

Sistemi di basi di dati

Fondamenti

Indice

Presentazione all'edizione italiana	XVII
Prefazione	XIX
Struttura dell'edizione italiana	XXIII
Capitolo 1 Basi di dati e utenti di basi di dati	1
1.1 Introduzione	2
1.2 Un esempio	4
1.3 Caratteristiche dell'approccio con basi di dati	6
1.3.1 Natura autodescrittiva di un sistema di basi di dati	6
1.3.2 Separazione tra programmi e dati e astrazione dei dati	7
1.3.3 Supporto di viste multiple dei dati	9
1.3.4 Condivisione dei dati ed elaborazione delle transazioni con utenti multipli	10
1.4 Gli attori in scena	10
1.4.1 Amministratori	10
1.4.2 Progettisti	11
1.4.3 Utenti finali	11
1.4.4 Analisti di sistema e programmatore di applicazioni (ingegneri del software)	12
1.5 I lavoratori dietro le quinte	12
1.6 Vantaggi dell'uso di un DBMS	13
1.6.1 Controllo della ridondanza	13
1.6.2 Divieto all'accesso non autorizzato	14
1.6.3 Memorizzazione persistente di oggetti e strutture di dati	15
1.6.4 Definizione di regole di inferenza e regole che usano azioni	15

1.6.5 Disponibilità di numerose interfacce utente	16
1.6.6 Rappresentazione di associazioni complesse fra dati	16
1.6.7 Imposizione di vincoli di integrità	16
1.6.8 Fornitura di backup e recovery	17
1.7 Implicazioni dell'approccio con basi di dati	17
1.8 Quando non usare un DBMS	18
Sommario	19
Questionario di verifica	20
Esercizi	20
Bibliografia selezionata	21
Capitolo 2 Concetti e architettura di un sistema di basi di dati	23
2.1 Modelli di dati, schemi e istanze	24
2.1.1 Categorie di modelli di dati	24
2.1.2 Schemi, istanze e stato di una base di dati	25
2.2 Architettura di un DBMS e indipendenza dei dati	27
2.2.1 L'architettura a tre livelli	27
2.2.2 Indipendenza dei dati	29
2.3 Linguaggi e interfacce di basi di dati	30
2.3.1 Linguaggi dei DBMS	30
2.3.2 Interfacce dei DBMS	32
2.4 L'ambiente di un sistema di basi di dati	33
2.4.1 Moduli componenti un DBMS	33
2.4.2 Programmi di utilità di un sistema di basi di dati	35
2.4.3 Strumenti, ambienti applicativi e funzioni di comunicazione	36
2.5 Classificazione dei sistemi di gestione di basi di dati	36
Sommario	38
Questionario di verifica	39
Esercizi	40
Bibliografia selezionata	40
Capitolo 3 Uso del modello Entità-Associazione per modellare i dati	41
3.1 Uso di modelli di dati concettuali di alto livello per la progettazione di basi di dati	42
3.2 Un'applicazione esemplificativa di basi di dati	44
3.3 Tipi di entità, insiemi di entità, attributi e chiavi	46
3.3.1 Entità e attributi	46
3.3.2 Tipi di entità, insiemi di entità, chiavi e insiemi di valori	49
3.3.3 Progettazione concettuale iniziale della base di dati AZIENDA	52
3.4 Associazioni, tipi di associazioni, ruoli e vincoli strutturali	53
3.4.1 Tipi di associazioni, insiemi di associazioni e istanze di associazione	53
3.4.2 Grado di un'associazione, nomi di ruolo e associazioni ricorsive	54
3.4.3 Vincoli sui tipi di associazione	57
3.4.4 Attributi di tipi di associazione	59
3.5 Tipi di entità debole	59

3.6 Raffinamento della progettazione ER per la base di dati AZIENDA	61
3.7 Diagrammi ER, convenzioni di denominazione e problemi di progettazione	62
3.7.1 Riepilogo della notazione dei diagrammi ER	62
3.7.2 Denominazione propria dei costrutti dello schema	64
3.7.3 Scelte per la progettazione concettuale ER	64
3.7.4 Notazioni alternative per i diagrammi ER	65
Sommario	66
Questionario di verifica	67
Esercizi	68
Bibliografia selezionata	73
Capitolo 4 Modellazione Entità-Associazione estesa e modellazione a oggetti	75
4.1 Sottoclassi, superclassi ed ereditarietà	76
4.2 Specializzazione e generalizzazione	78
4.3 Vincoli e caratteristiche di specializzazione e generalizzazione	81
4.4 Modellazione di tipi UNIONE attraverso l'uso di categorie	88
4.5 Uno schema EER esemplificativo UNIVERSITÀ e definizioni formali per il modello EER	92
4.6 Uso dei diagrammi delle classi UML per la modellazione concettuale a oggetti	95
4.7 Tipi di associazione di grado maggiore di due	98
4.8 Concetti di astrazione dei dati e di rappresentazione della conoscenza	102
4.8.1 Classificazione e istanziazione	103
4.8.2 Identificazione	104
4.8.3 Specializzazione e generalizzazione	104
4.8.4 Aggregazione e associazione	105
Sommario	107
Questionario di verifica	107
Esercizi	108
Bibliografia selezionata	113
Capitolo 5 Memorizzazione dei record e organizzazioni primarie dei file	115
5.1 Introduzione	116
5.1.1 Gerarchie di memoria e dispositivi di memorizzazione	116
5.1.2 Memorizzazione di basi di dati	118
5.2 Dispositivi di memoria secondaria	119
5.2.1 Descrizione dell'hardware di dispositivi a disco	119
5.2.2 Dispositivi di memorizzazione a nastro magnetico	124
5.3 Rendere parallelo l'accesso al disco attraverso l'uso della tecnologia RAID	125
5.3.1 Miglioramento dell'affidabilità	126
5.3.2 Miglioramento delle prestazioni	127
5.3.3 Organizzazioni e livelli RAID	128
5.4 Bufferizzazione di blocchi	130
5.5 Collocazione su disco dei record di un file	131
5.5.1 Record e tipi di record	131

5.5.2	File, record a lunghezza fissa e record a lunghezza variabile	132
5.5.3	Ripartizione dei record in blocchi e confronto tra record con spanning e record senza spanning	134
5.5.4	Allocazione dei blocchi di un file su disco	135
5.5.5	Header dei file	135
5.6	Operazioni sui file	136
5.7	File di record non ordinati (file heap)	138
5.8	File di record ordinati (file sorted)	140
5.9	Tecniche hash	143
5.9.1	Hash interno	143
5.9.2	Hash esterno per file su disco	146
5.9.3	Tecniche hash che consentono un'espansione dinamica dei file	148
5.10	Altre organizzazioni primarie dei file	152
5.10.1	File di record misti	152
5.10.2	Alberi B e altre strutture dati	153
Sommario	153	
Questionario di verifica	154	
Esercizi	155	
Bibliografia selezionata	158	
 Capitolo 6 Le strutture di indici per i file		
6.1	Tipi di indici ordinati a un solo livello	161
6.1.1	Indici primari	162
6.1.2	Indici di cluster	163
6.1.3	Indici secondari	165
6.2	Indice multilivello	168
6.3	Indici dinamici multilivello implementati da alberi B e alberi B ⁺	171
6.3.1	Alberi di ricerca e alberi B	175
6.3.2	Alberi B ⁺	176
6.4	Indice su chiavi multiple	181
6.4.1	Indice ordinato su più attributi	189
6.4.2	Hash partizionato	190
6.4.3	File a griglia	190
6.5	Altri tipi di indici	191
6.5.1	Utilizzo di hash e di altre strutture di dati come indici	192
6.5.2	Confronto tra indici logici e indici fisici	192
6.5.3	Discussione	193
Sommario	194	
Questionario di verifica	194	
Esercizi	195	
Bibliografia selezionata	198	
 Capitolo 7 Il modello di dati, i vincoli e l'algebra relazionali		
7.1	Concetti del modello relazionale	199
7.1.1	Domini, attributi, tuple e relazioni	200

7.1.2	Caratteristiche delle relazioni	202
7.1.3	Notazione del modello relazionale	205
7.2	Vincoli relazionali e schemi di basi di dati relazionali	206
7.2.1	Vincoli sul dominio	206
7.2.2	Vincoli di chiave e vincoli sui valori nulli	206
7.2.3	Basi di dati e schemi di basi di dati relazionali	208
7.2.4	Integrità dell'entità, integrità referenziale e chiavi esterne	210
7.3	Operazioni di aggiornamento e gestione delle violazioni dei vincoli	212
7.3.1	L'operazione di inserimento	213
7.3.2	L'operazione di cancellazione	214
7.3.3	L'operazione di modifica	215
7.4	Operazioni di base dell'algebra relazionale	215
7.4.1	L'operazione di SELEZIONE	216
7.4.2	L'operazione di PROIEZIONE	218
7.4.3	Sequenze di operazioni e operazione di RIDENOMINAZIONE	219
7.4.4	Operazioni insiemistiche	221
7.4.5	L'operazione di JOIN	225
7.4.6	Un insieme completo di operazioni dell'algebra relazionale	228
7.4.7	L'operazione di DIVISIONE	229
7.5	Altre operazioni relazionali	231
7.5.1	Funzioni aggregate e raggruppamento	231
7.5.2	Operazioni di chiusura ricorsiva	233
7.5.3	Operazioni di JOIN ESTERNO e di UNIONE ESTERNA	234
7.6	Esempi di interrogazioni in algebra relazionale	236
Sommario	238	
Questionario di verifica	240	
Esercizi	241	
Bibliografia selezionata	247	
 Capitolo 8 SQL - Lo standard delle basi di dati relazionali		
8.1	La definizione dei dati e le modifiche agli schemi in SQL2	249
8.1.1	I concetti di schema e di catalogo	251
8.1.2	Il comando CREATE TABLE e i vincoli di tipi di dati	251
8.1.3	I comandi DROP SCHEMA e DROP TABLE	252
8.1.4	Il comando ALTER TABLE	256
8.2	Interrogazioni fondamentali in SQL	257
8.2.1	La struttura SELECT-FROM-WHERE delle interrogazioni SQL	258
8.2.2	La problematica dei nomi di attributi ambigui e la loro ridenominazione (assegnando pseudonimi)	258
8.2.3	La clausola WHERE non specificata e l'utilizzo dell'asterisco (*)	261
8.2.4	Le tabelle come insiemi in SQL	262
8.2.5	Confronti di sottostringhe, operatori aritmetici e ordinamento	263
8.3	Interrogazioni SQL più complesse	265
8.3.1	Interrogazioni nidificate e confronti di insiemi	267
8.3.2	Funzioni EXISTS e UNIQUE	267

8.3.3 Insiemi esplicativi e NULLS	273
8.3.4 Ridenominazione degli attributi e tabelle collegate con i join	274
8.3.5 Funzioni di aggregazione e raggruppamento	276
8.3.6 Discussione e riepilogo delle interrogazioni SQL	281
8.4 Le istruzioni Insert, Delete e Update in SQL	282
8.4.1 Il comando INSERT	282
8.4.2 Il comando DELETE	284
8.4.3 Il comando UPDATE	284
8.5 Viste (tabelle virtuali) in SQL	285
8.5.1 Il concetto di vista	285
8.5.2 Specificazione delle viste	286
8.5.3 Implementazione e aggiornamento delle viste	287
8.6 Specificazione di vincoli generali come asserzioni	289
8.7 Ulteriori funzioni di SQL	290
Sommario	291
Questionario di verifica	292
Esercizi	293
Bibliografia selezionata	295
Capitolo 9 Traduzione da schemi ER e EER a schemi relazionali	297
9.1 Progettazione di basi di dati relazionali attraverso la traduzione da ER a relazionale	298
9.1.1 Algoritmo di traduzione da ER a relazionale	298
9.1.2 Sommario della traduzione per costrutti e vincoli del modello ER	302
9.2 Traduzione dei concetti del modello EER in relazioni	303
9.2.1 Associazioni superclasse/sottoclasse e specializzazione (o generalizzazione)	303
9.2.2 Traduzione delle sottoclassi condivise	306
9.2.3 Traduzione di categorie	306
Sommario	307
Questionario di verifica	308
Esercizi	308
Capitolo 10 Dipendenze funzionali e normalizzazione per basi di dati relazionali	309
10.1 Linee guida informali di progettazione di schemi di relazione	311
10.1.1 Semantica degli attributi di una relazione	311
10.1.2 Informazioni ridondanti nelle tuple e anomalie di aggiornamento	314
10.1.3 Valori nulli nelle tuple	316
10.1.4 Generazione di tuple spurie	317
10.1.5 Sommario ed esame delle linee guida di progettazione	320
10.2 Dipendenze funzionali	320
10.2.1 Definizione di dipendenza funzionale	320
10.2.2 Regole di inferenza per dipendenze funzionali	323
10.2.3 Equivalenza di insiemi di dipendenze funzionali	326

10.2.4 Insiemi minimali di dipendenze funzionali	326
10.3 Forme normali basate su chiavi primarie	327
10.3.1 Introduzione alla normalizzazione	328
10.3.2 Prima forma normale	329
10.3.3 Seconda forma normale	333
10.3.4 Terza forma normale	334
10.4 Definizioni generali di seconda e terza forma normale	335
10.4.1 Definizione generale di seconda forma normale	337
10.4.2 Definizione generale di terza forma normale	337
10.4.3 Interpretazione della definizione generale di 3NF	338
10.5 Forma normale di Boyce e Codd	338
Sommario	341
Questionario di verifica	342
Esercizi	342
Bibliografia selezionata	345
Capitolo 11 Algoritmi per la progettazione di basi di dati relazionali	347
11.1 Algoritmi per la progettazione di schemi di basi di dati relazionali	348
11.1.1 Decomposizione delle relazioni e insufficienza delle forme normali	348
11.1.2 Decomposizione e conservazione delle dipendenze	349
11.1.3 Decomposizione e join senza perdita (non-additivi)	351
11.1.4 Problemi con valori nulli e tuple dangling	357
11.1.5 Analisi degli algoritmi di normalizzazione	359
Sommario	360
Questionario di verifica	360
Esercizi	360
Bibliografia selezionata	361
Capitolo 12 Esempi di sistemi di gestione di basi di dati relazionali: Oracle e Microsoft Access	363
12.1 Sistemi di gestione di basi di dati relazionali: una prospettiva storica	364
12.2 Struttura base del sistema Oracle	365
12.2.1 Struttura della base di dati	365
12.2.2 Processi	367
12.2.3 Avvio	368
12.3 Struttura della base di dati e sua manipolazione in Oracle	369
12.3.1 Oggetti dello schema	369
12.3.2 Dizionario di dati di Oracle	371
12.3.3 SQL in Oracle	372
12.3.4 Metodi di Oracle	373
12.3.5 Trigger	373
12.4 Organizzazione della memoria in Oracle	374
12.4.1 Blocchi di dati	375
12.4.2 Estensioni	376
12.4.3 Segmenti	376

12.5 La programmazione delle applicazioni Oracle	377
12.5.1 Programmazione in PL/SQL	378
12.5.2 Cursori in PL/SQL	380
12.5.3 Un esempio in PRO*C	382
12.6 Strumenti di sviluppo di Oracle	385
12.7 Una visione d'insieme di Microsoft Access	386
12.7.1 Architettura di Access	386
12.7.2 Definizione dei dati delle basi di dati di Access	387
12.7.3 Definizione delle relazioni e dei vincoli di integrità referenziale	388
12.7.4 Manipolazione dei dati in Access	391
12.8 Caratteristiche e funzionalità di Access	393
12.8.1 Le maschere	393
12.8.2 I report	394
12.8.3 Le macro e Access Basic	395
12.8.4 Altre caratteristiche	396
Sommario	397
Bibliografia selezionata	397

Capitolo 13 Analisi multidimensionale dei dati e data mining	399
13.1 Analisi multidimensionale dei dati	399
13.1.1 Terminologia e definizioni	400
13.1.2 Le caratteristiche dei data warehouse	401
13.1.3 Modelli dei dati per i data warehouse	402
13.1.4 La creazione di un data warehouse	407
13.1.5 Funzionalità tipiche dei data warehouse	410
13.1.6 Difficoltà di implementazione dei data warehouse	411
13.1.7 Problemi aperti nelle tecniche di memorizzazione dei dati	413
13.2 Il data mining	413
13.2.1 Una visione d'insieme della tecnologia del data mining	414
13.2.2 Regole di associazione	418
13.2.3 Approcci ad altri problemi di data mining	423
13.2.4 Applicazioni del data mining	427
13.2.5 Stato dell'arte degli strumenti commerciali di data mining	427
Sommario	430
Esercizi	430
Bibliografia selezionata	431

Capitolo 14 Tecnologie e applicazioni emergenti delle basi di dati	433
14.1 Basi di dati e World Wide Web	434
14.1.1 Accesso Web alle basi di dati	435
14.1.2 Tecnica di integrazione Web di INFORMIX	436
14.1.3 WebServer di ORACLE	437
14.1.4 Problemi aperti con le basi di dati accessibili via Web	438
14.1.5 Bibliografia selezionata per le basi di dati accessibili via Web	440

14.2 Basi di dati multimediali	440
14.2.1 Natura dei dati e delle applicazioni multimediali	440
14.2.2 Problemi di gestione dati	442
14.2.3 Problemi di ricerca aperti	443
14.2.4 Applicazione delle basi di dati multimediali	445
14.2.5 Bibliografia selezionata per le basi di dati multimediali	446
14.3 Basi di dati mobili	446
14.3.1 Architettura di calcolo mobile	447
14.3.2 Tipi di dati delle applicazioni mobili	449
14.3.3 Problemi di gestione dei dati	449
14.3.4 Basi di dati mobili sincronizzate a intermittenza	450
14.3.5 Bibliografia selezionata per le basi di dati mobili	451
14.4 Sistemi informativi geografici	452
14.4.1 Applicazioni GIS	452
14.4.2 Requisiti di gestione dei dati dei GIS	453
14.4.3 Operazioni specifiche dei dati GIS	455
14.4.4 Un esempio di GIS: ARČ/INFO	456
14.4.5 Problemi e questioni future relative ai GIS	457
14.4.6 Bibliografia selezionata per i GIS	458
14.5 La gestione dei dati genetici	459
14.5.1 Le scienze biologiche e la genetica	459
14.5.2 Le caratteristiche dei dati biologici	460
14.5.3 Il progetto del genoma umano e le basi di dati biologiche esistenti	462
14.5.4 Bibliografia selezionata per le basi di dati del genoma	466
14.6 Biblioteche digitali	466
14.6.1 L'Iniziativa delle Biblioteche Digitali	467
14.6.2 Bibliografia selezionata per le biblioteche digitali	468
Appendice A Notazioni diagrammatiche alternative	469
Appendice B Parametri dei dischi	473
Bibliografia	477
Indice analitico	493



Presentazione all'edizione italiana

Ricercatori e studiosi italiani di informatica hanno dedicato grande attenzione al settore delle basi di dati fin dal suo nascere, contribuendo a produrre risultati di ricerca importanti a livello internazionale. Così non è stato per l'insegnamento di questa disciplina nell'ambito dei corsi universitari, in particolare per le Facoltà di Ingegneria, che solo con l'avvio dei corsi di laurea previsti dall'Ordinamento del 1989 hanno attivato il Corso di Laurea in Ingegneria Informatica nel cui ambito i corsi di basi di dati hanno trovato naturale collocazione. Di conseguenza in tutta Italia non sono stati resi disponibili, se non in anni recenti, libri in italiano idonei a essere adottati per l'insegnamento delle basi di dati all'università.

Chi, come me, ha avviato esperienze di insegnamento delle basi di dati fin dagli anni '80, spesso ha adottato e utilizzato il libro *Fundamentals of Database Systems* di Ramez Elmasri e Shamkant B. Navathe, che fin dalle sue prime edizioni si è rivelato fra i migliori – se non il migliore – dei testi disponibili. Questo libro combina il necessario rigore scientifico della disciplina, che sembra facile a chi non ne conosce la complessità e l'articolazione, alla semplicità e completezza di presentazione. Di conseguenza, da molti anni si pensava di rendere disponibile questo testo in italiano; l'occasione di realizzare la prima edizione italiana si è presentata con la riforma degli ordinamenti didattici ora in atto nelle università italiane. La riforma, meglio conosciuta con la formula del 3+2, che prende il via in quasi tutti gli Atenei italiani a partire dall'anno accademico 2001-2002, costituisce un cambiamento radicale del sistema universitario italiano. Molti docenti stanno facendo uno sforzo considerevole per rivedere i programmi degli insegnamenti, per innovarli e renderli più coerenti con lo spirito della riforma, che permette anche una maggiore autonomia.

Questo testo, che si basa sulla lunga esperienza come ricercatori e docenti degli autori, viene reso disponibile in italiano in una forma più snella rispetto all'originale e diventa un buon candidato per essere utilizzato come testo didattico, andandosi ad affiancare ad altri validi testi ora disponibili nel panorama nazionale. Nell'edizione italiana vengono presentati tutti gli aspetti di base della disciplina, insieme a quegli argomenti che più di recente sono stati sviluppati.

luppati nel settore e che forniscono al lettore gli elementi per continuare ad ampliare la propria conoscenza. In questo modo il testo può essere utilmente adottato per corsi che fanno parte di percorsi didattici anche molto diversi; per queste stesse ragioni esso si presta ad essere utilizzato come testo di base e di aggiornamento anche da parte del professionista informatico e da chi opera in azienda e si avvicina o lavora già in questo settore dell'informatica, perché utilizzandolo può acquisire anche competenze su nuove aree quali sono quelle del data warehouse e del data mining insieme a quella delle biblioteche digitali.

Molti sono i percorsi di lettura e di studio con i quali può esser utilmente utilizzato questo volume; oltre allo studio del testo nella sua interezza, sono possibili tre interessanti percorsi.

1. Un primo percorso fornisce una solida conoscenza delle basi di dati relazionali e della loro progettazione; esso è costituito dai Capitoli 1, 2, 3, 4, 7, 8, 9, 10 e 11 e, con l'aggiunta del Capitolo 12, fornisce anche le conoscenze relative a due sistemi commerciali di basi di dati di largo impiego.
2. Il percorso n.1, integrato con i Capitoli 5 e 6, permette di acquisire anche competenze relative agli aspetti implementativi e di progettazione fisica delle basi di dati.
3. Il percorso n.1, integrato con i Capitoli 13 e 14, permette di acquisire anche competenze relative a tematiche specialistiche e innovative del settore, quali sono le già citate tematiche del data warehouse, del data mining e delle biblioteche digitali, ma anche delle basi di dati e Web, delle basi di dati geografiche e di quelle per la genetica.

Il lettore interessato troverà nella prefazione degli autori altre informazioni che gli permetteranno di costruire ulteriori percorsi, più aderenti alle sue specifiche esigenze.

Ringraziamenti

Prima di tutto un ringraziamento particolare alla dott.ssa Donatella Pepe, Publishing Director di Addison Wesley Longman Italia, divisione di Pearson Education, che ha avuto e ha tenacemente perseguito l'idea di rendere disponibile in italiano questo testo, per la competenza e la sensibilità con la quale ha costantemente seguito e sostenuto le numerose e impegnative attività che hanno portato alla preparazione di questa edizione.

Un grazie di cuore all'ingegnere Luca Pretto, che ha tradotto la parte più consistente dell'opera, corrispondente ai Capitoli 1, 2, 3, 4, 5, 7, 9, 10 e 11 insieme alle Appendici A e B e alla quasi totalità dell'indice analitico. Luca ha dedicato a questa attività tutta la sua competenza di studioso, portando a termine un lavoro estremamente impegnativo con tutto il rigore e l'approfondimento necessario.

Un grazie anche alla dott.ssa Cristina Monteverdi e al Prof. Ernesto Damiani per aver accettato e aver portato a termine in tempi ristretti la traduzione dei Capitoli 6, 8, 12, 13 e 14.

Il ringraziamento finale alla dott.ssa Gabriella Piazza di Epitesto che, come copy editor del volume, è riuscita nel difficile compito di rispettare il rigore della trattazione, rendendo però il testo più scorrevole e piacevole da leggere.

Padova, ottobre 2001

Maristella Agosti
Dipartimento di Elettronica e Informatica
Università degli Studi di Padova

Prefazione

Questo libro introduce i concetti fondamentali necessari per progettare, usare e implementare sistemi di basi di dati e relative applicazioni. La nostra presentazione dà particolare enfasi ai fondamenti della modellazione e della progettazione di basi di dati, e ai linguaggi e alle funzioni forniti dai sistemi di basi di dati. Il libro è stato scritto per essere usato come testo per un corso sui sistemi di basi di dati di uno o due semestri a livello intermedio o avanzato, o come libro di consultazione. Si suppone che il lettore abbia una certa familiarità con concetti elementari di programmazione e strutture dati, e che conosca gli aspetti basilari dell'architettura dei calcolatori.

Si comincerà con l'introduzione e la presentazione dei concetti basilari per le due estremità dello spettro di una base di dati: principi di modellazione concettuale e tecniche di memorizzazione fisica dei file. Si concluderà il libro con una visione d'insieme di tecnologie e applicazioni emergenti, come il data mining, data warehousing e le basi di dati su Web. Nel corso di questo cammino forniremo un'approfondita trattazione dei più importanti aspetti dei fondamenti delle basi di dati.

La terza edizione presenta le seguenti caratteristiche fondamentali:

- tutto il libro ha un'organizzazione autosufficiente e flessibile che può essere adattata a esigenze individuali;
- è fornita una copertura completa e aggiornata del modello relazionale – compresa una nuova parte su Oracle e Microsoft Access come esempi di sistemi relazionali;
- un'esposizione aggiornata della modellazione concettuale EER è stata presentata nel Capitolo 4, in modo da seguire la modellazione ER di base presente nel Capitolo 3, e comprende un nuovo paragrafo sulla notazione usata per i diagrammi delle classi UML;
- due esempi che si ripresentano in tutto il libro – AZIENDA e UNIVERSITÀ – consentono al lettore di confrontare approcci diversi che usano la stessa applicazione;
- è stata aggiornata la materia sulla progettazione di basi di dati, compresa la progettazione concettuale, le tecniche di normalizzazione e la progettazione fisica;

- è presente un'esposizione aggiornata dei recenti progressi nelle applicazioni di basi di dati per il supporto alle decisioni, che comprende illustrazioni introduttive di data warehousing/OLAP e di data mining;
 - viene fornita una trattazione completa dello stato dell'arte di nuove tecnologie di basi di dati, incluse le basi di dati su Web, mobili e multimediali;
 - viene infine rivolta l'attenzione su nuove importanti aree applicative delle basi di dati all'inizio del nuovo millennio: basi di dati geografiche, basi di dati per la genetica e biblioteche digitali.

Contenuto di questa edizione

Inizialmente vengono descritti i concetti fondamentali necessari per una buona comprensione della progettazione e implementazione delle basi di dati, nonché le tecniche di modellazione concettuale usate nei sistemi di basi di dati. I Capitoli 1 e 2 introducono le basi di dati, i loro utenti tipici e i concetti, la terminologia e l'architettura dei DBMS. Nel Capitolo 3 sono presentati, e usati per illustrare la progettazione concettuale delle basi di dati, i concetti del modello Entità-Associazione (ER, Entity-Relationship) e i diagrammi ER. Il Capitolo 4 focalizza l'attenzione sui concetti di astrazione e di modellazione semantica dei dati, ed estende il modello ER per includere queste idee, portando così al modello di dati ER-esteso (EER) e ai diagrammi EER. I concetti presentati comprendono le sottoclassi, la specializzazione, la generalizzazione e i tipi unione (categorie). Sono state anche introdotte le notazioni per i diagrammi delle classi UML, simili ai diagrammi EER e sempre più usati nella modellazione concettuale a oggetti. Questa prima parte si conclude con una descrizione della struttura fisica dei file e dei metodi di accesso usati nei sistemi di basi di dati. Il Capitolo 5 descrive i metodi primari di organizzazione dei file di record su disco, compreso l'hash statico e dinamico. Il Capitolo 6 descrive le tecniche di indicizzazione per i file, comprendenti le strutture dati albero B e albero B⁺ e i file a griglia.

Si passa quindi a descrivere il modello di dati relazionale e i DBMS relazionali. Il Capitolo 7 descrive il modello relazionale di base, i suoi vincoli di integrità e operazioni di aggiornamento, e le operazioni dell'algebra relazionale. Il Capitolo 8 fornisce una visione dettagliata del linguaggio SQL, comprendente lo standard SQL2, che è implementato nella maggior parte dei sistemi relazionali. Il Capitolo 9 consiste di due paragrafi che descrivono la progettazione di schemi relazionali a partire da un progetto concettuale di base di dati nel modello ER o EER. I Capitoli 10 e 11 riguardano i formalismi, la teoria e gli algoritmi sviluppati per la progettazione di basi di dati relazionali tramite normalizzazione. Ciò comprende le dipendenze funzionali e le forme normali. Nel Capitolo 10 viene presentata la normalizzazione in modo intuitivo e nel Capitolo 11 vengono forniti algoritmi per la progettazione relazionale. Il Capitolo 12 presenta una visione d'insieme dei sistemi di basi di dati Oracle e Microsoft Access come esempi di ben noti sistemi commerciali di gestione di basi di dati.

Infine vengono considerati numerosi argomenti avanzati. Nel Capitolo 13 sono esaminate le nuove tecnologie di data warehousing e data mining per applicazioni di supporto alle decisioni. Il Capitolo 14 riassume le nuove tendenze nella tecnologia delle basi di dati, comprendenti le basi di dati su Web, mobili e multimediali e fornisce una visione d'insieme di impre-

tanti applicazioni emergenti delle basi di dati: sistemi di informazione geografica (GIS), basi di dati per il genoma umano e biblioteche digitali.

L'Appendice A fornisce notazioni diagrammatiche alternative per rappresentare graficamente uno schema concettuale ER o EER. Queste notazioni possono sostituire la notazione usata da noi, se lo si ritiene opportuno. Nell'Appendice B sono presenti alcuni importanti parametri fisici dei dischi.

Ringraziamenti

È per noi un grande piacere ringraziare le numerose persone che ci hanno offerto aiuto e collaborazione per questo lavoro. Prima di tutto desideriamo ringraziare i curatori Maite Suarez-Rivas, Katherine Harutunian, Patricia Unubun e Bob Woodbury. Vorremmo ringraziare in particolare l'opera e l'aiuto di Katherine Harutunian, che è stata il nostro punto di riferimento principale per la terza edizione. Desideriamo anche ringraziare quelle persone che hanno contribuito alla terza edizione e hanno suggerito vari miglioramenti alla seconda edizione. Suzanne Dietrich ha scritto parte dei Capitoli 10 e 12*, e Ed Omiecinski ha contribuito ai Capitoli 17*-21*. Abbiamo anche particolarmente apprezzato il contributo dei seguenti revisori: François Bancilhon, Jose Blakeley, Rick Cattell, Suzanne Dietrich, David W. Embley, Henry A. Etlinger, Leonidas Fegaras, Farshad Fotouhi, Michael Franklin, Goetz Graefe, Richard Hull, Sushil Jajodia, Ramesh K. Karne, Vijay Kumar, Tarcisio Lima, Ramon A. Mata-Toledo, Dennis McLeod, Rokia Missaoui, Ed Omiecinski, Joan Peckham, Betty Salzberg, Ming-Chien Shan, Junping Sun, Rajshekhar Sunderraman ed Emilia E. Villareal. In particolare Henry A. Etlinger, Leonidas Fegaras ed Emilia E. Villareal hanno letto tutto il libro.

Sham Navathe desidera ringraziare i contributi sostanziali dei suoi studenti Sreejith Gopinath (Capitolo 10, 24*), Harish Kotbagi (Capitolo 25*), Jack McCaw (Capitoli 26, 27) e Magdi di Morsi (Capitolo 13*). Notevole è stato l'aiuto a questa revisione fornito da Rafi Ahmed, Ann Chervenak, Dan Forsyth, M. Narayanaswamy, Carlos Ordóñez e Aravindan Veerasamy. Gwen Baker, Amol Navathe e Aditya Navathe hanno, in modi diversi, prestato il loro aiuto con il manoscritto. Ramez Elmasri desidera ringraziare Katrina, Riyad e Thomas Elmasri per l'aiuto fornito con l'indice analitico e i suoi studenti dell'Università del Texas per i commenti al manoscritto. Desideriamo anche ringraziare gli studenti dell'Università del Texas ad Arlington e del Georgia Institute of Technology che hanno usato le bozze relative alla materia aggiuntiva presente nella terza edizione.

Vogliamo ripetere qui i nostri ringraziamenti a tutti coloro che hanno svolto opera di revisione e comunque hanno contribuito a entrambe le edizioni precedenti di *Fundamentals of Database Systems*. Per la prima edizione si ringrazia Alan Apt (curatore), Don Batory, Scott Downing, Dennis Heimbigner, Julia Hodges, Yannis Ioannidis, Jim Larson, Dennis McLeod, Per-Ake Larson, Rahul Patel, Nicholas Roussopoulos, David Stemple, Michael Stonebraker,

¹ I capitoli contrassegnati con un asterisco non sono stati inseriti nella presente edizione italiana. Inoltre la numerazione dei capitoli presente in questo paragrafo si riferisce all'edizione americana del testo, ed è diversa da quella italiana. In particolare, il Capitolo 10 dell'edizione americana corrisponde al 12 di quella italiana, e i Capitoli 26 e 27 corrispondono ai 13 e 14, rispettivamente.

Frank Tompa e Kyu-Young Whang; per la seconda edizione Dan Joraanstad (curatore), Rafi Ahmed, Antonio Albano, David Beech, Jose Blakeley, Panos Chrysanthis, Suzanne Dietrich, Vic Ghorpadey, Goetz Graefe, Eric Hanson, Junguk L. Kim, Roger King, Vram Kouramajian, Vijay Kumar, John Lowther, Sanjay Manchanda, Toshimi Minoura, Inderpal Mumick, Ed Omiecinski, Girish Pathak, Raghu Ramakrishnan, Ed Robertson, Eugene Sheng, David Stotts, Marianne Winslett e Stan Zdonick.

Infine ringraziamo affettuosamente le nostre famiglie per il sostegno, l'incoraggiamento e la pazienza con cui ci hanno seguiti.

R.E.
S.B.N.

Profilo degli autori

Ramez A. Elmasri è professore del dipartimento di Computer Science and Engineering dell'Università del Texas ad Arlington. Il professor Elmasri ha lavorato in precedenza per Honeywell e per l'Università di Houston. È stato condirettore del *Journal of Parallel and Distributed Databases* e membro del comitato direttivo dell'International Conference on Conceptual Modeling. È stato inoltre coordinatore del comitato di programma dell'International Conference on Entity Relationship Approach del 1993. Negli ultimi vent'anni ha svolto ricerche finanziate da NSF, NASA, ARRI, Texas Instruments, Honeywell, Digital Equipment Corporation e dallo Stato del Texas in molte aree dei sistemi di basi di dati e nell'area dell'integrazione dei sistemi e del software. Il professor Elmasri ha ricevuto il premio Robert Q. Lee per la sua attività di docente da parte del College of Engineering dell'Università del Texas ad Arlington. Ha conseguito il Ph.D. all'Università di Stanford ed è autore di più di 70 pubblicazioni su riviste e atti di congressi.

Shankant Navathe è professore e coordinatore del gruppo di ricerca sulle basi di dati al College of Computing del Georgia Institute of Technology. Il professor Navathe ha lavorato in precedenza con IBM e Siemens nei rispettivi centri di ricerca ed è stato consulente per varie aziende, fra cui Digital Equipment Corporation, Hewlett-Packard ed Equifax. È stato condirettore di *ACM Computing Surveys* e di *IEEE Transactions on Knowledge and Data Engineering*, e attualmente è nei comitati di redazione di *Information Systems* (Pergamon Press) e *Distributed and Parallel Databases* (Kluwer Academic Publishers). È coautore di *Conceptual Design: An Entity Relationship Approach* (Addison-Wesley, 1992) con Carlo Batini e Stefano Ceri. Il professor Navathe ha conseguito il Ph.D. all'Università del Michigan ed è autore di più di 100 pubblicazioni su riviste e atti di congressi.

Struttura dell'edizione italiana

Con questa prima edizione abbiamo scelto di offrire al lettore italiano un percorso didattico mirato alle nuove necessità delle più recenti tendenze del mondo accademico. Abbiamo quindi stabilito di focalizzare questa edizione sui temi maggiormente trattati nei corsi dedicati all'argomento, pensando soprattutto a quelli dei primi tre anni di studio.

Il testo dei professori Elmasri e Navathe rappresenta per tutti gli "addetti ai lavori" un indispensabile punto di riferimento didattico e professionale; riteniamo quindi opportuno, se non doveroso, riportare lo schema di lavorazione che abbiamo seguito in corso d'opera, così che quanti sono abituati a orientarsi sul testo originale americano possano rapidamente ritrovarsi anche all'interno di questa prima edizione italiana.

Fundamentals of Database Systems, 3rd ed.	Sistemi di basi di dati – Fondamenti 1a ed. ital.
Chapter 1	Capitolo 1
Chapter 2	Capitolo 2
Chapter 3	Capitolo 3
Chapter 4	Capitolo 4
Chapter 5	Capitolo 5
Chapter 6	Capitolo 6
Chapter 7	Capitolo 7
Chapter 8	Capitolo 8
Chapter 9	Capitolo 9 (§ 9.1 e 9.2)
Chapter 10	Capitolo 12
<i>Chapters 11 - 13</i>	<i>omessi</i>
Chapter 14	Capitolo 10
Chapter 15	Capitolo 11 (§ 15.1)
<i>Chapters 16 - 25</i>	<i>omessi</i>
Chapter 26	Capitolo 13
Chapter 27	Capitolo 14
Appendix A	Appendice A
Appendix B	Appendice B
<i>Appendix C - D</i>	<i>omesse</i>

Capitolo 1

Basi di dati e utenti di basi di dati

Nella società odierna le basi di dati e i sistemi di basi di dati sono diventati una componente essenziale della vita quotidiana. Nell'arco di una giornata molti di noi si imbattono in numerose attività che comportano una qualche interazione con una base di dati. Per esempio, se andiamo in banca per depositare o prelevare denaro, se effettuiamo una prenotazione aerea o alberghiera, se accediamo a un catalogo computerizzato di una biblioteca alla ricerca di una voce bibliografica o se ordiniamo a un editore un abbonamento a una rivista, le nostre azioni implicheranno l'accesso a una base di dati da parte di qualcuno. Oggigiorno perfino l'acquisto di articoli da un supermercato comporta in molti casi un aggiornamento automatico della base di dati che mantiene l'inventario degli articoli del magazzino.

Le interazioni sopra descritte sono esempi di ciò che è possibile chiamare applicazioni tradizionali di basi di dati, nelle quali gran parte dell'informazione che è memorizzata e a cui si accede è testuale o numerica. Negli ultimi anni i miglioramenti tecnologici hanno portato a nuove e suggestive applicazioni di sistemi di basi di dati. Le basi di dati multimediali possono ora memorizzare immagini, video e messaggi sonori. I sistemi di informazione geografica (GIS: geographic information systems) possono memorizzare e analizzare carte geografiche, dati meteorologici e immagini da satelliti. I sistemi di data warehouse (magazzino di raccolta di dati) e di on-line analytical processing (OLAP) (elaborazione e analisi dei dati in linea) sono usati in molte aziende per estrarre e analizzare informazioni utili da basi di dati molto ampie al fine di prendere decisioni. La tecnologia delle basi di dati in tempo reale e delle basi di dati attive è usata nel controllo di processi industriali e manifatturieri. Inoltre le tecniche di ricerca proprie delle basi di dati sono attualmente applicate al World Wide Web per migliorare la ricerca dell'informazione desiderata dagli utenti che navigano in Internet.

Ad ogni modo, per capire gli aspetti fondamentali della tecnologia delle basi di dati, occorre partire dai primi elementi delle loro applicazioni tradizionali. Nel Paragrafo 1.1 di questo capitolo, spiegheremo quindi che cos'è una base di dati e daremo le definizioni di alcuni termini fondamentali. Nel Paragrafo 1.2 forniremo un semplice esempio di base di dati, UNI-

VERSITÀ, per illustrare la nostra analisi. Nel Paragrafo 1.3 descriveremo alcune delle principali caratteristiche dei sistemi di basi di dati e nei Paragrafi 1.4 e 1.5 quelle categorie di personale il cui lavoro prevede l'uso e l'interazione con essi. Nei Paragrafi 1.6, 1.7 e 1.8 si offre un'analisi più approfondita delle varie possibilità fornite dai sistemi di basi di dati e delle implicazioni insite nel loro uso. Il lettore che desidera solo una rapida introduzione ai sistemi di basi di dati può concentrarsi sui Paragrafi 1.1-1.5, quindi saltare o dare solo una rapida scorsa ai Paragrafi 1.6-1.8, per passare direttamente al Capitolo 2.

1.1 Introduzione

Le basi di dati e la tecnologia delle basi di dati stanno esercitando un'influenza fondamentale nell'uso sempre più esteso del computer. È giusto dire che le basi di dati giocano un ruolo fondamentale in quasi tutti i campi in cui i computer sono utilizzati, tra cui, solo per citarne alcuni, il mondo degli affari, l'ingegneria, la medicina, la giurisprudenza, l'istruzione e la biblioteconomia. La locuzione base di dati è di uso così comune che occorre cominciare col darne una definizione, seppur abbastanza generale.

Una base di dati è una collezione di dati correlati.¹ Per dati si intendono fatti noti che possono essere memorizzati e che hanno un significato implicito. Considerate ad esempio i nomi, i numeri di telefono e gli indirizzi delle persone che conoscete. Potete aver riportato questi dati su una rubrica, o potete averli memorizzati in un dischetto, usando un personal computer e software come DBASE IV o V, Microsoft ACCESS, o EXCEL: si tratta di una collezione di dati correlati con un significato隐含的 e quindi è una base di dati.

La precedente definizione di base di dati è piuttosto generale; ad esempio si potrebbe considerare la collezione di parole che formano questa pagina di testo come dati correlati e quindi costituenti una base di dati. Peraltra l'uso comune della locuzione base di dati è di solito più restrittivo. Una base di dati ha le seguenti proprietà implicite:

- rappresenta un certo aspetto del mondo reale, talvolta detto il mini-mondo o l'Universo del Discorso (UoD: Universe of Discourse); cambiamenti nel mini-mondo si riflettono sulla base di dati;
- è una collezione di dati logicamente coerenti con un certo significato intrinseco; un assortimento casuale di dati non può essere correttamente considerato una base di dati.
- è progettata, costruita e popolata con dati per uno scopo specifico; ha uno specifico gruppo di utenti e alcune applicazioni che questi utenti hanno individuato in precedenza essere di loro interesse.

In altre parole, una base di dati ha una certa sorgente dalla quale i dati sono derivati, un certo grado di interazione con gli eventi nel mondo reale, e un pubblico che è attivamente interessato al suo contenuto.

Una base di dati può essere di qualsiasi dimensione e di diversa complessità. Ad esempio, la lista di nomi e indirizzi a cui si è fatto riferimento prima può consistere solamente di poche centinaia di record, ciascuno con una struttura semplice. D'altra parte lo schedario di una grande biblioteca può contenere mezzo milione di schede immagazzinate sotto diverse categorie – per cognome dell'autore principale, per soggetto, per titolo dei libri – con ciascuna categoria organizzata in ordine alfabetico. Una base di dati di anche maggior ampiezza e complessità è gestita dal servizio erariale statunitense (IRS: Internal Revenue Service) per tener traccia dei moduli delle tasse fatti archiviare dai contribuenti americani. Se si suppone che vi siano 100 milioni di contribuenti e che ciascun contribuente faccia archiviare una media di cinque moduli con approssimativamente 200 caratteri di informazione per modulo, si ottiene una base di dati di $100 \times (10^6) \times 200 \times 5$ caratteri (byte) di informazione. Se l'IRS mantiene le ultime tre dichiarazioni dei redditi per ciascun contribuente in aggiunta alla dichiarazione corrente, si otterrà una base di dati di $4 \times (10^{11})$ byte (400 gigabyte). Questo ampio ammontare di informazione deve essere organizzato e gestito in modo tale che gli utenti possano cercare, recuperare e aggiornare i dati secondo necessità.

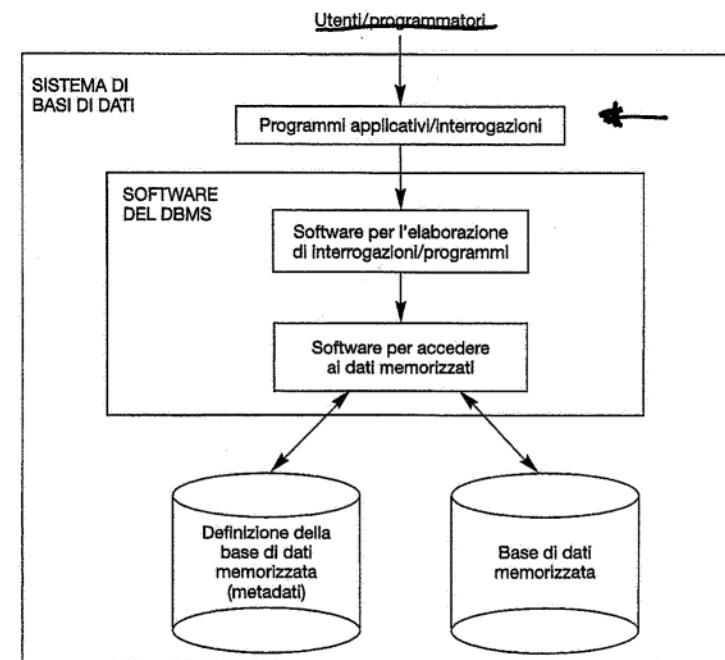


Figura 1.1 Un ambiente di sistema di basi di dati semplificato, che illustra i concetti e la terminologia esaminati nel Paragrafo 1.1.

¹ Nella letteratura delle basi di dati in lingua inglese il termine *data* è usato sia al singolare sia al plurale; il contesto permette al lettore di capire quando è singolare o plurale. Nell'inglese corrente *data* è usato solamente come *datum* singolare.

DEFINIRE COSTRUIRE MANIPOLARE

Una base di dati può essere generata e mantenuta manualmente oppure può essere computerizzata. Lo schedario di una biblioteca è un esempio di base di dati che può essere prodotta e mantenuta manualmente. Una base di dati computerizzata può essere prodotta e mantenuta da un gruppo di programmi applicativi scritti specificamente allo scopo o da un sistema di gestione di basi di dati.

Un sistema di gestione di basi di dati (DBMS: database management system) è un insieme di programmi che permettono agli utenti di creare e mantenere una base di dati. DBMS è perciò un sistema software con scopi generali (general-purpose software system) che facilita il processo di definire, costruire e manipolare basi di dati per varie applicazioni. Definire una base di dati implica specificare i tipi dei dati, le loro strutture e i vincoli per i dati che devono essere memorizzati nella base di dati. Costruire la base di dati significa immagazzinare i dati stessi entro un certo mezzo di memorizzazione che è controllato dal DBMS. Manipolare una base di dati include funzioni come la sua interrogazione per recuperare dati specifici, il suo aggiornamento per rispecchiare cambiamenti nel mini-mondo e la generazione di prospetti (reports) a partire dai dati.

Non è necessario usare software DBMS general-purpose per realizzare una base di dati computerizzata. È possibile scrivere un proprio insieme di programmi per produrre e mantenere la base di dati, creando un software DBMS personale con scopi specifici (special-purpose). In entrambi i casi — che si usi un general-purpose DBMS o no — occorre generalmente utilizzare una considerevole quantità di software per manipolare la base di dati. Nel seguito l'insieme costituito dalla base di dati e dal software DBMS verrà indicato come un sistema di basi di dati, illustrato in Figura 1.1.

1.2 Un esempio

Si consideri un esempio familiare a molti lettori: una base di dati UNIVERSITÀ che contenga informazioni riguardanti gli studenti, gli insegnamenti e le votazioni in ambiente universitario. In Figura 1.2 sono mostrati la struttura della base di dati e alcuni dati campione per questa. La base di dati è organizzata come cinque file, ciascuno dei quali contiene record di dati dello stesso tipo²: il file STUDENTE contiene i dati di ciascuno studente; il file INSEGNAMENTO contiene i dati di ciascun insegnamento; il file MODULO contiene i dati di ciascun modulo di un insegnamento; il file VOTAZIONE contiene i voti ottenuti dagli studenti nei vari moduli da eseguire; il file PROPEDEUTICITÀ contiene le propedeuticità di ciascun insegnamento.

Per definire questa base di dati è necessario specificare la struttura dei record di ciascun file specificando i diversi tipi di dati elementari da memorizzare in ciascun record. In Figura 1.2 ciascun record di STUDENTE contiene dati per rappresentare il Nome dello studente, il suo NumeroStudente, il suo AnnoCorso (primo o 1, secondo o 2 ecc.) e CorsoLaurea (MATH per Matematica, CS per Computer Science o Ingegneria informatica ecc.); ciascun re-

STUDENTE	Nome	NumeroStudente	AnnoCorso	CorsoLaurea
Smith		17	1	CS
Brown		8	2	CS

INSEGNAMENTO	NomeInsegnamento	CodiceInsegnamento	Ore	Dipartimento
Introduzione all'informatica		CS1310	4	CS
Struttura di dati		CS3320	4	CS
Matematica discreta		MATH2410	3	MATH
Basi di dati		CS3380	3	CS

MODULO	IdentificatoreModulo	CodiceInsegnamento	Semestre	Anno	Docente
85		MATH2410	Autunno	98	King
92		CS1310	Autunno	98	Anderson
102		CS3320	Primavera	99	Knuth
112		MATH2410	Autunno	99	Chang
119		CS1310	Autunno	99	Anderson
135		CS3380	Autunno	99	Stone

VOTAZIONE	NumeroStudente	IdentificatoreModulo	Voto
17		112	B
17		119	C
8		85	A
8		92	A
8		102	B
8		135	A

PROPEDEUTICITÀ	CodiceInsegnamento	CodicePropedeuticità
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

Figura 1.2 Esempio di una base di dati contenente record sugli studenti e i loro voti.

cord di INSEGNAMENTO contiene dati per rappresentare il NomeInsegnamento, il CodiceInsegnamento, le Ore di lezione settimanali e il Dipartimento (che organizza l'insegnamento); analogo discorso vale per gli altri file. Occorre pure specificare un tipo di dati per ciascun dato elementare all'interno di un record: ad esempio, specificare che Nome di STUDENTE è una stringa di caratteri alfabetici, NumeroStudente di STUDENTE è un intero e Voto di VOTAZIONE è un singolo carattere dell'insieme di caratteri {A, B, C, D, F, I}. Si può pure usare uno schema di codifica per rappresentare un'occorrenza di dato: ad esempio, in Figura 1.2 l'anno di corso di uno STUDENTE è rappresentato con 1 per primo, 2 per secondo, 3 per terzo e così via.

² A livello concettuale un file è una collezione di record, che può essere o non essere ordinata.

Per costruire la base di dati UNIVERSITÀ bisogna memorizzare i dati per rappresentare ogni studente, insegnamento, modulo, votazione e propedeuticità come un record nel file appropriato. Si noti che record diversi nei vari file possono essere collegati. Per esempio il record per "Smith" nel file STUDENTE è collegato con due record nel file VOTAZIONE che specificano i voti ottenuti da Smith in due moduli. Analogamente ciascun record nel file PROPEDEUTICITÀ collega due record di INSEGNAMENTO: uno che rappresenta un insegnamento e l'altro che rappresenta l'insegnamento ad esso propedeutico. Molte basi di dati di medie e grandi dimensioni comprendono molti tipi di record e presentano molte associazioni fra i record.

La manipolazione di basi di dati comprende l'interrogazione e l'aggiornamento. Esempi di interrogazione sono: "recupera la trascrizione del libretto universitario – un elenco di tutti gli esami sostenuti e dei voti ottenuti – di Smith"; "elenna i nomi degli studenti che hanno sostenuto l'esame relativo al modulo dell'insegnamento Basi di dati tenuto nell'autunno 2001 e i loro voti in questo modulo"; "quali sono gli insegnamenti propedeutici a Basi di dati?". Esempi di aggiornamento sono: "cambia l'anno di corso di Smith in secondo"; "crea un nuovo modulo per l'insegnamento Basi di dati per questo semestre"; "inserisci un voto A per Smith nel modulo di Basi di dati dell'ultimo semestre". Queste interrogazioni e aggiornamenti informali devono essere specificati esattamente nel linguaggio del sistema di basi di dati prima di essere eseguiti.

1.3 Caratteristiche dell'approccio con basi di dati

Un certo numero di caratteristiche distinguono l'approccio con basi di dati dall'approccio tradizionale della programmazione con file. Nella tradizionale gestione file ciascun utente definisce e implementa i file necessari per una specifica applicazione, come parte della programmazione dell'applicazione. Ad esempio, un utente, l'ufficio per la registrazione dei voti, può gestire un file sugli studenti e i loro voti. Sono quindi implementati programmi per stampare la trascrizione del libretto universitario di uno studente e per inserire nuovi voti nel file. Un secondo utente, l'ufficio contabilità, può tener traccia delle tasse d'iscrizione degli studenti e dei loro pagamenti. Per quanto entrambi gli utenti siano interessati a dati sugli studenti, ciascun utente mantiene file separati – e programmi per manipolare questi file – poiché ciascuno dichiede dei dati non disponibili nei file dell'altro utente. Questa ridondanza nel definire e memorizzare i dati ha come risultato uno spreco di spazio di memoria e un doppio sforzo per tenere aggiornati i dati comuni. Nell'approccio con basi di dati, invece, si mantiene un singolo magazzino di dati, definito una volta per tutte e al quale poi possono accedere vari utenti.

1.3.1 Natura autodescrittiva di un sistema di basi di dati

Una caratteristica fondamentale è che il sistema di basi di dati contiene non solo la base di dati ma anche un insieme autodescrittivo completo della sua struttura e dei suoi vincoli. Questo

sta definizione è memorizzata nel catalogo del sistema, che contiene informazioni come la struttura di ciascun file, il tipo e il formato di memorizzazione di ciascun dato e vari vincoli sui dati. Le informazioni memorizzate nel catalogo sono dette metadati e descrivono la struttura della base di dati principale (Figura 1.1).

Il catalogo è usato dal software del DBMS e anche dagli utenti della base di dati che hanno bisogno di informazioni sulla struttura della stessa. Un pacchetto software di un DBMS general-purpose non è scritto per una specifica applicazione di basi di dati, e di conseguenza deve riferirsi al catalogo per conoscere la struttura dei file in un database specifico, come il tipo e il formato dei dati a cui accederà. Il software del DBMS deve lavorare egualmente bene con qualsiasi tipo di applicazione di basi di dati – ad esempio una base di dati universitaria, una base di dati bancaria o una base di dati aziendale – per tutto il tempo in cui la definizione della base di dati permane nel catalogo.

Nella tradizionale gestione file, la definizione dei dati è tipicamente parte dei programmi applicativi. Di conseguenza questi programmi sono costretti a lavorare solo con una base di dati specifica, la cui struttura è dichiarata nei programmi applicativi. Ad esempio un programma PASCAL può avere dichiarate al suo interno strutture di record; un programma C++ può avere dichiarazioni di "struct" o "class" e un programma COBOL ha istruzioni della Sezione Dati per definire i suoi file. Mentre il software di gestione file può accedere solo a basi di dati specifiche, il software del DBMS può accedere a diverse basi di dati estrarrendone le definizioni dal catalogo e quindi usando queste definizioni.

Nell'esempio di Figura 1.2 il DBMS memorizza nel catalogo le definizioni di tutti i file mostrati. Ogni volta che viene fatta una richiesta di accedere, ad esempio, a Nome di un record di STUDENTE, il software del DBMS ricorre al catalogo per determinare la struttura del file STUDENTE e la posizione, nonché la dimensione del dato Nome all'interno di un record di STUDENTE. Al contrario, in una tipica applicazione di gestione file, la struttura del file e, al limite, la posizione esatta di Nome all'interno di un record di STUDENTE sono già codificate all'interno di ciascun programma che accede a questo dato.

1.3.2 Separazione tra programmi e dati e astrazione dei dati

Nella tradizionale gestione file la struttura dei file di dati è inserita nei programmi che devono accedervi e pertanto qualsiasi cambiamento alla struttura di un file può richiedere un cambiamento di tutti i programmi che accedono ad esso. Al contrario, i programmi di accesso del

Nome del dato	Posizione iniziale del record	Lunghezza in caratteri (byte)
Nome	1	30
NumeroStudente	31	4
AnnoCorso	35	4
CorsoLaurea	39	4

Figura 1.2 Formato di memorizzazione di un record di STUDENTE.

DBMS nella maggioranza dei casi non richiedono cambiamenti di questo tipo. La struttura dei file di dati è memorizzata nel catalogo del DBMS separatamente dai programmi di accesso. Questa proprietà è chiamata indipendenza tra programmi e dati. Ad esempio un programma di accesso a un file può essere scritto in modo da accedere solo a record di STUDENTE con la struttura mostrata in Figura 1.3. Se si vuole aggiungere un'altra parte di dati a ciascun record di STUDENTE, ad esempio DataNascita, un programma di questo tipo non funzionerà più e dovrà essere cambiato. In un ambiente DBMS, invece, c'è solo bisogno di cambiare la descrizione dei record di STUDENTE nel catalogo per rispecchiare l'inserimento del nuovo dato DataNascita; nessun programma viene modificato. La volta successiva che un programma del DBMS ricorrerà al catalogo accederà alla nuova struttura dei record di STUDENTE e la userà.

Nelle basi di dati orientate a oggetti e in quelle relazionali e a oggetti gli utenti possono definire le operazioni sui dati come parte delle definizioni della base di dati. Un'operazione (detta anche funzione) è specificata in due parti. L'interfaccia (o segnatura) di un'operazione comprende il nome dell'operazione e i tipi di dati dei suoi argomenti (o parametri). L'implementazione (o metodo) dell'operazione è specificata separatamente e può essere cambiata senza interessare l'interfaccia. I programmi applicativi dell'utente possono operare sui dati invocando queste operazioni attraverso i loro nomi e argomenti, indipendentemente da come le operazioni stesse siano implementate. Questo fatto può essere definito indipendenza tra programmi e operazioni.

La caratteristica che consente l'indipendenza tra programmi e dati e l'indipendenza tra programmi e operazioni è detta astrazione dei dati. Un DBMS fornisce agli utenti una rappresentazione concettuale dei dati che non comprende molti dettagli su come i dati sono memorizzati e su come le operazioni sono implementate. Informalmente, un modello di dati è un tipo di astrazione dei dati usato per fornire questa rappresentazione concettuale. Il modello di dati usa concetti logici, come oggetti, le loro proprietà e le loro interrelazioni, che sono per molti utenti più semplici da capire rispetto a nozioni di memorizzazione nel calcolatore. Di conseguenza il modello di dati nasconde dettagli di memorizzazione e implementazione che non sono di interesse per molti utenti di basi di dati.

Ad esempio, si consideri ancora la Figura 1.2. L'implementazione interna di un file può essere definita dalla lunghezza del suo record – il numero di caratteri (byte) in ciascun record – e ciascun dato può essere specificato dal suo byte iniziale all'interno del record e dalla sua lunghezza in byte. Il record di STUDENTE sarebbe perciò rappresentato come mostrato in Figura 1.3. Tuttavia un tipico utente di basi di dati non è interessato alla posizione di ciascun dato all'interno di un record o alla sua lunghezza; è piuttosto interessato al fatto che, qualora si faccia riferimento a Nome di STUDENTE, venga restituito il valore corretto. Una rappresentazione concettuale dei record di STUDENTE è mostrata in Figura 1.2. Molti altri dettagli dell'organizzazione della memorizzazione di file – come i percorsi di accesso specificati in un file – possono essere nascosti agli utenti della base di dati dal DBMS (si vedano i Capitoli 5 e 6).

Nell'approccio con basi di dati la struttura e l'organizzazione dettagliate di ciascun file sono memorizzate nel catalogo. Gli utenti fanno riferimento alla rappresentazione concettuale dei file, e il DBMS estrae dal catalogo i dettagli della loro memorizzazione quando questi siano necessari al software del DBMS. Molti modelli di dati diversi possono essere usati per fornire tale astrazione dei dati agli utenti di basi di dati: una parte importante di questo libro è

dedicata alla loro presentazione e ai concetti che essi usano per astrarre la rappresentazione dei dati.

Con la recente tendenza verso basi di dati orientate a oggetti e basi di dati relazionali e a oggetti, l'astrazione è portata a un livello superiore per includere non solo la struttura dei dati ma anche le operazioni sui dati. Queste operazioni forniscono un'astrazione di attività del mini-mondo comunemente comprensibili agli utenti. Ad esempio un'operazione CALCO-LA-MV può essere applicata a un oggetto studente per calcolare la media voti. Tali operazioni possono essere invocate dalle interrogazioni Q dai programmi dell'utente senza che l'utente conosca i dettagli su come esse siano internamente implementate. In questo senso, un'astrazione dell'attività del mini-mondo è resa disponibile all'utente come operazione astratta.

1.3.3 Supporto di viste multiple dei dati

Tipicamente una base di dati ha molti utenti, ciascuno dei quali può richiederne una diversa prospettiva o vista. Una vista può essere un sottoinsieme della base di dati o può contenere dati virtuali che sono derivati dai file della base di dati ma che non sono esplicitamente memorizzati. Talvolta può non essere necessario rendere consapevoli gli utenti del fatto che i dati a cui si riferiscono sono memorizzati piuttosto che derivati da altri. Un DBMS multietente i cui utenti hanno una molteplicità di applicazioni deve fornire funzioni per definire viste multiple. Ad esempio, un utente della base di dati di Figura 1.2 può essere interessato solo alla trascrizione del libretto universitario di ciascuno studente: la vista per questo utente è mostrata in Figura 1.4(a). Un secondo utente, che sia interessato solo a verificare che gli studenti abbiano sostenuto tutti gli esami relativi a insegnamenti propedeutici a ciascun insegnamento a cui si sono iscritti, può aver bisogno della vista illustrata in Figura 1.4(b).

Trascrizione del libretto universitario dello studente					
	NomeStudente	CodiceInsegnamento	Voto	Semestre	Anno
(a)	Smith	CS1310	C	Autunno	99
		MATH2410	B	Autunno	99
		MATH2410	A	Autunno	98
	Brown	CS1310	A	Autunno	98
		CS3320	B	Primavera	99
		CS3380	A	Autunno	99

PROPEDEUTICITÀ	NomeInsegnamento	CodiceInsegnamento	Propedeuticità
(b)	Basi di dati	CS3380	CS3320
		MATH2410	CS3320
	Strutture di dati	CS3320	CS1310

Figura 1.4 Due viste derivate dalla base di dati esemplificativa di Figura 1.2. (a) La vista della trascrizione del libretto universitario degli studenti. (b) La vista delle propedeuticità agli insegnamenti.

1.3.4 Condivisione dei dati ed elaborazione delle transazioni con utenti multipli

Un DBMS multutente, come dice il nome, deve consentire a più utenti di accedere contemporaneamente alla base di dati. La cosa è essenziale se dati per applicazioni multiple devono essere integrati e mantenuti in una singola base di dati. Il DBMS deve contenere un software per il controllo della concorrenza che garantisca che più utenti che cerchino di aggiornare gli stessi dati lo possano fare in maniera controllata, cosicché il risultato degli aggiornamenti sia corretto. Ad esempio, quando molti addetti alle prenotazioni cercano di assegnare un posto in un volo aereo, il DBMS dovrebbe garantire che ciascun posto possa essere preso in considerazione per essere assegnato a un passeggero da un solo addetto alla volta. Applicazioni di questo tipo sono generalmente dette applicazioni di elaborazione delle transazioni in linea (OLTP: on-line transaction processing). Una funzione fondamentale del software di un DBMS multutente è quella di assicurare che le transazioni concorrenti operino correttamente.
Le caratteristiche ora viste sono le principali per distinguere un DBMS dal software tradizionale di gestione file (per ulteriori approfondimenti si veda il Paragrafo 1.6).

1.4 Gli attori in scena

In una piccola base di dati a uso personale, come ad esempio l'elenco di indirizzi di cui si è parlato nel Paragrafo 1.1, tipicamente una sola persona definisce, costruisce e manipola la base di dati. Viceversa molte persone sono coinvolte nella progettazione, nell'uso e nella manutenzione di una grande base di dati con qualche centinaio di utenti. In questo paragrafo verranno individuate le persone la cui attività lavorativa prevede l'uso quotidiano di una base di dati di grandi dimensioni; chiameremo queste persone gli "attori in scena". Nel Paragrafo 1.5 verranno invece considerati i "lavoratori dietro le quinte", coloro che lavorano per la manutenzione dell'ambiente del sistema di basi di dati, ma che non sono direttamente interessati alla base di dati in sé.

1.4.1 Amministratori

In qualsiasi organizzazione in cui molte persone usano le stesse risorse c'è bisogno di un amministratore capo per sovrintendere e gestire queste risorse. In un contesto di basi di dati la risorsa primaria è la base di dati stessa e la risorsa secondaria è il DBMS e il software correlato. L'amministrazione di queste risorse è responsabilità dell'**amministratore della base di dati (DBA: database administrator)**. È responsabilità del DBA autorizzare l'accesso alla base di dati, coordinare e monitorare il suo uso e acquisire risorse software e hardware quando esse si rivelino necessarie. Il DBA risponde inoltre di problemi quali la violazione al sistema o tempi di risposta scadenti da parte di quest'ultimo. In organizzazioni di grandi dimensioni il DBA è supportato da uno staff che lo aiuta a svolgere tali mansioni.

1.4.2 Progettisti

I **progettisti di basi di dati** hanno la responsabilità di individuare i dati da memorizzare nella base di dati e di scegliere strutture adeguate per rappresentarli e memorizzarli. Questi compiti sono per lo più intrapresi prima che la base di dati sia realmente implementata e popolata di dati. È responsabilità dei progettisti di basi di dati comunicare con i futuri utenti della base di dati, per capirne le esigenze e per dar vita a un progetto che le soddisfi. In molti casi i progettisti fanno parte dello staff del DBA e possono essere assegnate loro altre responsabilità di staff dopo che è stata completata la progettazione della base di dati.

Essi tipicamente interagiscono con ciascun gruppo di utenti potenziali e sviluppano una vista della base di dati che si accordi con le richieste di dati e di elaborazioni del gruppo stesso. Queste viste sono poi analizzate e *integrate* con le viste di altri gruppi di utenti. Il progetto finale della base di dati deve essere in grado di soddisfare le richieste di tutti i gruppi di utenti.

1.4.3 Utenti finali

Gli utenti finali sono quelle persone le cui attività lavorative richiedono l'accesso alla base di dati per interrogazioni, aggiornamenti e generazione di prospetti: la base di dati esiste principalmente per loro. Ci sono diverse categorie di utenti finali.

- **Utenti casuali.** Essi accedono occasionalmente alla base di dati, ma possono aver bisogno ogni volta di informazioni diverse. Usano un linguaggio di interrogazione sofisticato per specificare le loro richieste e sono tipicamente manager di medio o alto livello o altre persone che accedono occasionalmente alla base di dati.
- **Utenti finali non sofisticati (o parametrici).** Costituiscono una porzione rimarchevole degli utenti finali della base di dati. La loro principale funzione lavorativa consiste nell'interrogare e aggiornare continuamente la stessa, usando tipi standard di interrogazioni e aggiornamenti – detti **transazioni standard (canned transactions)**: transazioni "in scatola", cioè preconfezionate – accuratamente programmati e testati. Svariati sono i compiti svolti da tali utenti. Ad esempio, i cassieri di banca controllano i bilanci dei conti e registrano prelievi e depositi; gli addetti alle prenotazioni per linee aeree, hotel e società di autonoleggio verificano la disponibilità in corrispondenza a una data richiesta ed effettuano le prenotazioni; gli impiegati nelle stazioni di ricevimento di un corriere inseriscono gli elementi identificativi di un pacco attraverso codici a barre e informazioni descrittive attraverso pulsanti, per aggiornare una base di dati centrale sui pacchi ricevuti e in transito.
- **Utenti finali sofisticati.** Comprendono ingegneri, scienziati, analisti commerciali e altre persone che acquisiscono completa familiarità con le funzioni del DBMS in modo da implementare le loro applicazioni per soddisfare le loro complesse esigenze.
- **Utenti indipendenti.** Mantengono basi di dati a uso personale usando pacchetti di programmi già pronti che forniscono interfacce a menu di facile uso o interfacce di tipo grafico. Un esempio è l'utente di un pacchetto tributario che memorizza una varietà di dati personali a scopo fiscale.

(a) VOTAZIONE	NumeroStudente	NomeStudente	IdentificatoreModulo	CodiceInsegnamento	Voto
17	Smith	112		MATH2410	B
17	Smith	119		CS1310	C
8	Brown	85		MATH2410	A
8	Brown	92		CS1310	A
8	Brown	102		CS3320	B
8	Brown	135		CS3380	A

(b) VOTAZIONE	NumeroStudente	NomeStudente	IdentificatoreModulo	CodiceInsegnamento	Voto
17	Brown	112		MATH2410	B

Figura 1.5 La memorizzazione ridondante di dati. (a) *Ridondanza controllata*: si includono NomeStudente e CodiceInsegnamento nel file VOTAZIONE. (b) *Ridondanza non controllata*: un record di VOTAZIONE che non è consistente con i record di STUDENTE di Figura 1.2, perché il nome dello studente numero 17 è Smith, non Brown.

Nell'approccio con basi di dati le viste di diversi gruppi di utenti vengono integrate durante la progettazione della base di dati. Per assicurare la consistenza si dovrebbe avere una progettazione che memorizza ciascun dato logico – come il nome o la data di nascita di uno studente – in *un solo posto* nella base di dati. Questo non permette inconsistenze e riduce l'occupazione di memoria. Peraltro in alcuni casi una **ridondanza controllata** può essere utile per migliorare le prestazioni delle interrogazioni. Ad esempio è possibile memorizzare NomeStudente e CodiceInsegnamento in maniera ridondante in un file VOTAZIONE (Figura 1.5a), se ogni volta che si recupera un record di VOTAZIONE si vogliono recuperare il nome dello studente e il codice dell'insegnamento insieme con il voto, il numero dello studente e l'identificatore del modulo. Gestendo tutti i dati insieme, per raccoglierli non è necessario ricercare più file. In questi casi il DBMS dovrebbe essere in grado di *controllare* la ridondanza in modo tale da evitare inconsistenze tra i file. Questo obiettivo può essere raggiunto verificando automaticamente che i valori di NomeStudente-Numerostudente in ogni record di VOTAZIONE in Figura 1.5(a) si accordino con uno dei valori di Nome-Numerostudente di un record di STUDENTE (Figura 1.2). Analogamente i valori di IdentificatoreModulo-CodiceInsegnamento in VOTAZIONE possono essere confrontati con i record di MODULO. Queste verifiche possono essere specificate al DBMS durante la progettazione della base di dati e automaticamente imposte dal DBMS ogni volta che il file VOTAZIONE è aggiornato. In Figura 1.5(b) è mostrato un record di VOTAZIONE che è inconsistente con il file STUDENTE di Figura 1.2 e che può essere inserito erroneamente se la ridondanza è *non controllata*.

1.6.2 Divieto all'accesso non autorizzato

Quando più utenti condividono una base di dati, è verosimile che ad alcuni non sarà permesso di accedere a tutte le informazioni in essa contenute. I dati finanziari, ad esempio, sono spesso considerati riservati e di conseguenza l'accesso è consentito solo a persone autorizzate a esserne permesso solo di recuperare dati, mentre ad altri sarà

permesso sia di recuperarli sia di aggiornarli. Di conseguenza deve essere controllato il tipo di operazione di accesso – recupero o aggiornamento. Tipicamente si forniscono agli utenti o a gruppi di utenti account protetti da password, che possono essere usati per ottenere l'accesso alla base di dati. Un DBMS dovrebbe fornire un sottosistema per la sicurezza e l'autorizzazione, usato dal DBA per creare gli account e per specificare le restrizioni per ciascun account. Il DBMS dovrebbe quindi imporre automaticamente queste restrizioni. Si noti che simili controlli si possono applicare al software del DBMS. Si può, ad esempio, consentire al solo staff di DBA di usare certo software privilegiato, come il software per creare nuovi account. Analogamente, agli utenti parametrici può essere concesso di accedere alla base di dati solo attraverso le transazioni standard sviluppate per essere usate da loro.

1.6.3 Memorizzazione persistente di oggetti e strutture di dati

Le basi di dati possono essere usate per fornire **memorizzazione persistente** di oggetti di programmi e strutture di dati. Questa è una delle ragioni principali per cui sono stati sviluppati i sistemi di basi di dati a oggetti. I linguaggi di programmazione hanno strutture dati complesse, come i tipi record in PASCAL o le definizioni di classe in C++. I valori di variabili di programma sono scartati una volta che un programma termina, a meno che il programmatore non le memorizzi esplicitamente in file persistenti, il che spesso comporta una conversione di queste complesse strutture in un formato adatto per la memorizzazione su file. Quando sorge la necessità di leggere questi dati un'altra volta, il programmatore deve attuare la conversione dal formato file alla struttura di variabile di programma. I sistemi di basi di dati a oggetti sono compatibili con linguaggi di programmazione come C++ e JAVA, e il software del DBMS esegue automaticamente ogni conversione necessaria. Di conseguenza un oggetto complesso in C++ può essere memorizzato persistentemente in un DBMS a oggetti, come ObjectStore o O2 (ora detto Ardent). Un oggetto di questo tipo è detto **persistente**, dal momento che sopravvive alla terminazione dell'esecuzione del programma e può in seguito essere direttamente recuperato da un altro programma C++.

La memorizzazione persistente di oggetti di programmi e strutture di dati è una funzione importante dei sistemi di basi di dati. I sistemi di basi di dati tradizionali hanno spesso sofferto del cosiddetto problema del conflitto di impedimento (**impedance mismatch problem**), dal momento che le strutture di dati fornite dal DBMS erano incompatibili con le strutture dati del linguaggio di programmazione. Tipicamente i sistemi di basi di dati a oggetti offrono compatibilità della struttura di dati con uno o più linguaggi di programmazione a oggetti.

1.6.4 Definizione di regole di inferenza e regole che usano azioni

Alcuni sistemi di basi di dati forniscono servizi per definire **regole di deduzione** in modo da *inferire* nuove informazioni dai fatti memorizzati nella base di dati. Sistemi di questo tipo so-

no detti **sistemi di basi di dati deduttive**. Ad esempio, ci possono essere regole complesse nell'applicazione del mini-mondo per determinare quando uno studente è sotto esame. Queste possono essere specificate con *dichiarazioni esplicite* come **regole**, che una volta compilate e mantenute dal DBMS sono in grado di determinare tutti gli studenti sotto esame. In un DBMS tradizionale si sarebbe dovuto scrivere un esplicito *codice di programma procedurale* per supportare tali applicazioni. Ma se le regole del mini-mondo cambiano, è generalmente più conveniente cambiare le regole di deduzione dichiarate che riscrivere il codice di programmi procedurali. Una funzionalità più potente è fornita dai **sistemi di basi di dati attive**, che forniscono regole attive che possono automaticamente dar inizio ad azioni.

1.6.5 Disponibilità di numerose interfacce utente

Poiché una base di dati è utilizzata da molti tipi di utenti, con svariati livelli di conoscenza tecnica, un DBMS dovrebbe fornire una molteplicità di interfacce utente. Queste includono linguaggi di interrogazione per utenti casuali, interfacce a linguaggi di programmazione per programmati di applicazioni, moduli (*forms*) e codici di comando per utenti parametrici e interfacce a menu e in linguaggio naturale per utenti indipendenti. Sia le interfacce a moduli sia quelle a menu sono comunemente note come **interfacce utente grafiche (GUIs: graphical user interfaces)**. Esistono molti linguaggi e ambienti specializzati per specificare GUI. Sempre più comuni sono inoltre servizi per fornire accesso via World Wide Web a una base di dati – o abilitazione via Web di una base di dati.

1.6.6 Rappresentazione di associazioni complesse fra dati

Una base di dati può comprendere numerose varietà di dati associati in molti modi. Si consideri l'esempio mostrato in Figura 1.2. Il record relativo a Brown nel file STUDENTE è associato a quattro record nel file VOTAZIONE. In modo simile, ciascun record di MODULO è associato a un record di INSEGNAMENTO così come a un certo numero di record di VOTAZIONE – uno per ciascuno studente che ha superato quel modulo. Un DBMS deve essere in grado di rappresentare una varietà di associazioni complesse fra i dati, così come di recuperare e aggiornare i dati correlati facilmente ed efficientemente.

1.6.7 Imposizione di vincoli di integrità

Molte applicazioni di basi di dati presentano **vincoli di integrità** che devono valere per i dati. Un DBMS dovrebbe fornire servizi per definire e imporre questi vincoli. Il tipo più semplice di vincolo di integrità comporta la specificazione di un tipo di dati per ciascun dato. Ad esempio in Figura 1.2 è possibile specificare che il valore del dato AnnoCorso per ciascun record "voto a un studente" esser intero tra 1 e 5 e che il valore di Nome deve esse-

re una stringa di non più di 30 caratteri alfabetici. Un tipo più complesso di vincolo, che si presenta frequentemente, comporta la specificazione che un record in un file deve essere correlato a record in altri file. Ad esempio, in Figura 1.2 è possibile specificare che "ogni record relativo a un modulo deve essere correlato a un record di insegnamento". Un altro tipo di vincolo specifica l'unicità di valori per certi dati, come ad esempio "ogni record di insegnamento deve assumere un unico valore per CodiceInsegnamento". Questi vincoli sono dedotti dal significato o semantica dei dati e del mini-mondo che essi rappresentano. È responsabilità dei progettisti della base di dati individuare i vincoli di integrità durante la fase di progettazione. Alcuni vincoli possono essere specificati al DBMS e imposti automaticamente. È possibile che altri vincoli vengano controllati dai programmi di aggiornamento o nell'istante in cui vengono inseriti i dati.

Un dato può essere stato inserito erroneamente e tuttavia soddisfare i vincoli di integrità specificati. Ad esempio, se uno studente ottiene come voto un A, ma nella base di dati viene inserito un C, il DBMS non può scoprire questo errore automaticamente, perché C è un valore valido per il tipo di dati di Voto. Errori nell'inserimento dei dati di questo tipo possono essere scoperti solo manualmente (quando lo studente legge il voto e protesta) e corretti in seguito aggiornando la base di dati. Però un voto Z può essere automaticamente rifiutato dal DBMS, perché Z non è un valore valido per il tipo di dati di Voto.

1.6.8 Fornitura di backup e recovery

Un DBMS deve fornire funzioni di ripristino da guasti hardware e software. Il **sottosistema di backup (salvataggio)** e **recovery (ripristino)** del DBMS è responsabile del ripristino. Ad esempio, se il sistema di computer si guasta a metà di un complesso programma di aggiornamento, il sottosistema di recovery ha la responsabilità di assicurare che la base di dati venga ripristinata nello stato in cui era prima che il programma iniziasse l'esecuzione. Alternativamente il sottosistema di recovery potrebbe assicurare che il programma venga ripreso dal punto in cui era stato interrotto, in modo tale che venga registrato nella base di dati il suo effetto completo.

1.7 Implicazioni dell'approccio con basi di dati

In aggiunta ai punti discussi nel paragrafo precedente ci sono altre implicazioni, nell'uso dell'approccio con basi di dati, di cui possono giovarsi molte organizzazioni.

Potenziale per imporre standard. L'approccio con basi di dati permette al DBA di definire e imporre standard fra utenti di basi di dati in una grande organizzazione. Ciò facilita la comunicazione e la cooperazione tra vari dipartimenti, progetti e utenti all'interno dell'organizzazione. Possono essere definiti standard per nomi e formati di dati elementari, formati di visualizzazione, strutture di prospetti, terminologia e così via. Il DBA può imporre gli standard

~~no detti sistemi di basi di dati deduttive. Ad esempio, ci possono essere regole complesse nell'applicazione del mini-mondo per determinare quando uno studente è sotto esame. Queste possono essere specificate con dichiarazioni esplicite come regole che una volta compilate e mantenute dal DBMS sono in grado di determinare tutti gli studenti sotto esame. In un DBMS tradizionale si sarebbe dovuto scrivere un esplicito codice di programma procedurale per supportare tali applicazioni. Ma se le regole del mini-mondo cambiano, è generalmente più conveniente cambiare le regole di deduzione dichiarate che riscrivere il codice di programmi procedurali. Una funzionalità più potente è fornita dai sistemi di basi di dati attive, che forniscono regole attive che possono automaticamente dar inizio ad azioni.~~

1.6.5 Disponibilità di numerose interfacce utente

Poiché una base di dati è utilizzata da molti tipi di utenti, con svariati livelli di conoscenza tecnica, un DBMS dovrebbe fornire una molteplicità di interfacce utente. Queste includono linguaggi di interrogazione per utenti casuali, interfacce a linguaggi di programmazione per programmatore di applicazioni, moduli (*forms*) e codici di comando per utenti parametrici e interfacce a menu e in linguaggio naturale per utenti indipendenti. Sia le interfacce a moduli sia quelle a menu sono comunemente note come interfacce utente grafiche (GUIs: graphical user interfaces). Esistono molti linguaggi e ambienti specializzati per specificare GUI. Sempre più comuni sono inoltre servizi per fornire accesso via World Wide Web a una base di dati – o abilitazione via Web di una base di dati.

1.6.6 Rappresentazione di associazioni complesse fra dati

Una base di dati può comprendere numerose varietà di dati associati in molti modi. Si consideri l'esempio mostrato in Figura 1.2. Il record relativo a Brown nel file STUDENTE è associato a quattro record nel file VOTAZIONE. In modo simile, ciascun record di MODULO è associato a un record di INSEGNAMENTO così come a un certo numero di record di VOTAZIONE – uno per ciascuno studente che ha superato quel modulo. Un DBMS deve essere in grado di rappresentare una varietà di associazioni complesse fra i dati, così come di recuperare e aggiornare i dati correlati facilmente ed efficientemente.

1.6.7 Imposizione di vincoli di integrità

Molte applicazioni di basi di dati presentano vincoli di integrità che devono valere per i dati. Un DBMS dovrebbe fornire servizi per definire e imporre questi vincoli. Il tipo più semplice di vincolo di integrità comporta la specificazione di un tipo di dati per ciascun dato. Ad esempio in Figura 1.2 è possibile specificare che il valore del dato AnnoCorso per ciascun record relativo a uno studente deve essere un intero tra 1 e 5 e che il valore di Nome deve esse-

re una stringa di non più di 30 caratteri alfabetici. Un tipo più complesso di vincolo, che si presenta frequentemente, comporta la specificazione che un record in un file deve essere corrisposto a record in altri file. Ad esempio, in Figura 1.2 è possibile specificare che "ogni record relativo a un modulo deve essere correlato a un record di insegnamento". Un altro tipo di vincolo specifica l'unicità di valori per certi dati, come ad esempio "ogni record di insegnamento deve assumere un unico valore per CodiceInsegnamento". Questi vincoli sono dedotti dal significato o semantica dei dati e del mini-mondo che essi rappresentano. È responsabilità dei progettisti della base di dati individuare i vincoli di integrità durante la fase di progettazione. Alcuni vincoli possono essere specificati al DBMS e imposti automaticamente. È possibile che altri vincoli vengano controllati dai programmi di aggiornamento o nell'istante in cui vengono inseriti i dati.

Un dato può essere stato inserito erroneamente e tuttavia soddisfare i vincoli di integrità specificati. Ad esempio, se uno studente ottiene come voto un A, ma nella base di dati viene inserito un C, il DBMS non può scoprire questo errore automaticamente, perché C è un valore valido per il tipo di dati di Voto. Errori nell'inserimento dei dati di questo tipo possono essere scoperti solo manualmente (quando lo studente legge il voto e protesta) e corretti in seguito aggiornando la base di dati. Per un voto Z può essere automaticamente rifiutato dal DBMS, perché Z non è un valore valido per il tipo di dati di Voto.

1.6.8 Fornitura di backup e recovery

Un DBMS deve fornire funzioni di ripristino da guasti hardware e software. Il sottosistema di backup (salvataggio) e recovery (ripristino) del DBMS è responsabile del ripristino. Ad esempio, se il sistema di computer si guasta a metà di un complesso programma di aggiornamento, il sottosistema di recovery ha la responsabilità di assicurare che la base di dati venga ripristinata nello stato in cui era prima che il programma lasciasse l'esecuzione. Alternativamente il sottosistema di recovery potrebbe assicurare che il programma venga ripreso dal punto in cui era stato interrotto, in modo tale che venga registrato nella base di dati il suo effetto completo.

1.7 Implicazioni dell'approccio con basi di dati

In aggiunta ai punti discussi nel paragrafo precedente ci sono altre implicazioni, nell'uso dell'approccio con basi di dati, di cui possono giovare molte organizzazioni.

Potenziale per imporre standard. L'approccio con basi di dati permette al DBA di definire e imporre standard fra utenti di basi di dati in una grande organizzazione. Ciò facilita la comunicazione e la cooperazione tra vari dipartimenti, progetti e utenti all'interno dell'organizzazione. Possono essere definiti standard per nomi e formati di dati elementari, formati di visualizzazione, strutture di prospetti, terminologia e così via. Il DBA può imporre gli standard

più facilmente in un ambiente con una base di dati centralizzata che in un ambiente in cui ciascun gruppo di utenti ha il controllo dei propri file e del proprio software.

Tempo ridotto per lo sviluppo di applicazioni. Una ragione di successo fondamentale dell'approccio con basi di dati è che lo sviluppo di una nuova applicazione – come il recupero di certi dati dalla base di dati per stampare un nuovo prospetto – richiede molto poco tempo. La progettazione e l'implementazione da zero di una nuova base di dati può richiedere molto più tempo che non scrivere una singola e specializzata applicazione su file. Però, una volta che una base di dati è realizzata e in funzione, generalmente è richiesto decisamente meno tempo per creare nuove applicazioni usando i servizi del DBMS (da un sesto a un quarto di quello richiesto usando un file system tradizionale).

Flessibilità. Quando cambiano le esigenze può essere necessario cambiare la struttura di una base di dati. Per esempio, può sorgere un nuovo gruppo di utenti che abbisogna di informazioni al momento non presenti in essa. In risposta a ciò può essere necessario aggiungere un file alla base di dati o aumentare i dati elementari in un file esistente. I DBMS moderni consentono alcuni cambiamenti alla struttura della base di dati senza coinvolgere i dati memorizzati e i programmi applicativi esistenti.

Disponibilità di informazioni aggiornate. Un DBMS rende la base di dati disponibile a tutti gli utenti. Non appena in essa viene effettuato un aggiornamento di un utente, tutti gli altri utenti possono immediatamente vederlo. Questa disponibilità di informazioni aggiornate è essenziale per molte applicazioni che elaborano transazioni, come sistemi di prenotazione o basi di dati bancarie, ed è resa possibile dal controllo della concorrenza e dai sottosistemi di ripristino disponibili in un DBMS.

Economie di scala. L'approccio con DBMS permette l'unificazione dei dati e delle applicazioni, riducendo così l'ammontare di dispendiosa sovrapposizione tra le attività del personale di elaborazione dati in diversi progetti o dipartimenti. Questo consente all'intera organizzazione di investire in più potenti processori, dispositivi di memoria o meccanismi di comunicazione piuttosto che dover far acquistare a ciascun dipartimento la propria attrezzatura (meno potente). Ciò riduce i costi operativi e di gestione complessivi.

1.8 Quando non usare un DBMS

Nonostante i vantaggi di usare un DBMS, ci sono alcune situazioni nelle quali un tale sistema può comportare spese generali non necessarie a cui non ci si esporrebbe nella tradizionale gestione file. Le spese generali nell'uso di un DBMS sono dovute alle seguenti ragioni:

- alti investimenti iniziali in hardware, software e formazione;
- generalità fornita da un DBMS per definire ed elaborare dati;

- spese generali per assicurare le funzioni di sicurezza, controllo della concorrenza, ripristino e integrità.

Problemi aggiuntivi possono sorgere se i progettisti e il DBA non progettano correttamente la base di dati o se le applicazioni dei sistemi di basi di dati non sono implementate correttamente. Perciò può essere conveniente usare solo file nelle seguenti circostanze:

- la base di dati e le applicazioni sono semplici, ben definite e non si prevede che debbano cambiare;
- ci sono stringenti necessità di tempo reale per alcuni programmi che non possono essere soddisfatte a causa del DBMS sovrastante;
- non è richiesto un accesso ai dati da parte di più utenti contemporaneamente.

Sommario

In questo capitolo abbiamo definito una base di dati come una collezione di dati correlati, dove per *dati* si intende fatti registrati. Una tipica base di dati rappresenta alcuni aspetti del mondo reale ed è usata per scopi specifici da uno o più gruppi di utenti. Un DBMS è un pacchetto software di uso generale per implementare e mantenere una base di dati computerizzata. La base di dati e il software nel complesso formano un sistema di basi di dati. Abbiamo individuato diverse caratteristiche che distinguono l'approccio con basi di dati dalle tradizionali applicazioni di gestione file:

- esistenza di un catalogo;
- indipendenza tra programmi e dati e indipendenza tra programmi e operazioni;
- astrazione dei dati;
- mantenimento di viste per più utenti;
- condivisione dei dati tra transazioni multiple.

Quindi abbiamo esaminato le principali categorie di utenti di basi di dati, o gli "attori in scena":

- amministratori;
- progettisti;
- utenti finali;
- analisti di sistema e programmatore di applicazioni.

Abbiamo notato che, in aggiunta agli utenti di basi di dati, ci sono alcune categorie di personale di supporto, o "lavoratori dietro le quinte":

- progettisti e implementatori di sistema DBMS;
- sviluppatori di strumenti;
- operatori e personale per la manutenzione.



Abbiamo poi presentato un elenco di servizi che dovrebbero essere forniti dal software del DBMS a DBA, progettisti e utenti per aiutarli a progettare, amministrare e usare una base di dati:

- controllo della ridondanza;
- divieto all'accesso non autorizzato;
- memorizzazione persistente di oggetti e strutture di dati;
- definizione di regole di inferenza e regole che usano azioni;
- disponibilità di numerose interfacce utente;
- rappresentazione di associazioni complesse fra dati;
- imposizione di vincoli di integrità;
- fornitura di backup e recovery.

Abbiamo elencato alcuni vantaggi aggiuntivi dell'approccio con basi di dati rispetto a sistemi tradizionali di gestione file:

- potenziale per imporre standard;
- tempo ridotto per lo sviluppo di applicazioni;
- flessibilità;
- disponibilità di informazioni aggiornate a tutti gli utenti;
- economie di scala.

Infine abbiamo esaminato le spese generali legate all'uso di un DBMS e abbiamo discusso alcune situazioni nelle quali può non essere vantaggioso ricorrere ad esso.

Questionario di verifica

- 1.1. Si definiscano i seguenti termini: *dati, base di dati, DBMS, sistema di basi di dati, catalogo di basi di dati, indipendenza tra programmi e dati, vista d'utente, DBA, utente finale, transazione standard, sistema di basi di dati deduttivo, oggetto persistente, metadati, applicazione di elaborazione di transazione.*
- 1.2. Quali sono i tre tipi principali di azioni che coinvolgono le basi di dati? Le si tratti brevemente una ad una.
- 1.3. Si esaminino le principali caratteristiche dell'approccio con basi di dati e come esso differisce dai file system tradizionali.
- 1.4. Quali sono le responsabilità del DBA e dei progettisti di basi di dati?
- 1.5. Quali sono i diversi tipi di utenti finali di basi di dati? Si discutano le principali attività di ciascuno di essi.
- 1.6. Si esaminino i servizi che dovrebbero essere forniti da un DBMS.

Esercizi

- 1.7. Si identifichino alcune interrogazioni e operazioni di aggiornamento informali che si ritiene ragionevole indirizzare alla base di dati mostrata in Figura 1.2.

- 1.8. Qual è la differenza tra ridondanza controllata e non controllata? La si illustri con alcuni esempi.
- 1.9. Si definiscano tutte le associazioni fra i record della base di dati mostrata in Figura 1.2.
- 1.10. Si forniscano alcune viste aggiuntive che possono essere necessarie ad altri gruppi di utenti per la base di dati mostrata in Figura 1.2.
- 1.11. Si citino alcuni esempi di vincoli di integrità che dovrebbero verosimilmente sussistere per la base di dati di Figura 1.2.

Bibliografia selezionata

Il numero di ottobre 1991 di *Communications of the ACM* e Kim (1995) comprendono svariati articoli che descrivono DBMS "della prossima generazione"; molte delle caratteristiche delle basi di dati discusse in questo numero sono ora disponibili a livello commerciale. Il numero di marzo 1976 di *ACM Computer Surveys* offre una delle prime introduzioni esistenti ai sistemi di basi di dati e può fornire al lettore interessato una prospettiva storica.



Capitolo 2

Concetti e architettura di un sistema di basi di dati

L'architettura dei DBMS è evoluta dai primi sistemi monolitici, dove l'intero pacchetto software costituiva un solo sistema strettamente integrato, ai moderni pacchetti di DBMS, che sono a progetto modulare, con un'architettura di sistema di tipo client-server (cliente-servente). Questa evoluzione rispecchia la tendenza attualmente esistente nella scienza dei calcolatori, dove i grandi mainframe centralizzati vengono man mano sostituiti da centinaia di workstation e personal computer distribuiti, connessi da reti di comunicazione. In un'architettura client-server di base la funzionalità del sistema è distribuita tra due tipi di moduli. Un modulo client è di norma progettato in modo tale da poter essere eseguito sulla workstation o sul personal computer dell'utente. Tipicamente i programmi applicativi e le interfacce utente che accedono alla base di dati vengono eseguiti nel modulo client. Perciò quest'ultimo gestisce l'interazione con l'utente e fornisce le interfacce amichevoli come moduli o interfacce utente grafiche (GUI) a menu. L'altro tipo di modulo, detto modulo server, tratta la memorizzazione dei dati, l'accesso, la ricerca e altre funzioni.

Ora però occorre introdurre concetti più basilari che consentiranno di raggiungere una migliore comprensione delle moderne architetture di basi di dati. In questo capitolo presenteremo quindi la terminologia e i concetti base che saranno usati per tutto il libro. Cominceremo nel Paragrafo 2.1, esaminando i modelli di dati e definendo i concetti di schema e istanza, fondamentali per lo studio dei sistemi di basi di dati. Tratteremo poi l'architettura a tre livelli dei DBMS e l'indipendenza dei dati nel Paragrafo 2.2; questo fornisce una prospettiva d'utente su ciò che ci si aspetta che un DBMS faccia. Nel Paragrafo 2.3 descriveremo i tipi di interfacce e linguaggi che un DBMS tipicamente rende disponibili, nel Paragrafo 2.4 l'ambiente software di un sistema di basi di dati e infine, nel Paragrafo 2.5, daremo una classificazione dei DBMS (questi ultimi due paragrafi forniscono concetti più dettagliati che possono essere considerati un'integrazione al materiale introduttivo di base).

2.1 Modelli di dati, schemi e istanze

Una caratteristica fondamentale dell'approccio con basi di dati è che esso fornisce un certo livello di astrazione dei dati nascondendo quei dettagli sulla memorizzazione degli stessi che non sono necessari alla maggior parte degli utenti. Un modello di dati – un insieme di concetti che possono essere usati per descrivere la struttura di una base di dati – fornisce i mezzi necessari per raggiungere questa astrazione.¹ Per struttura di una base di dati si intendono i tipi di dati, le associazioni e i vincoli che dovrebbero valere su di essi. La maggior parte dei modelli di dati comprende pure un insieme di operazioni di base per specificare recuperi e aggiornamenti sulla base di dati.

In aggiunta alle operazioni di base fornite dal modello di dati sta diventando sempre più comune includere in esso concetti per specificare l'aspetto dinamico o il comportamento di un'applicazione di basi di dati. Ciò consente al progettista della base di dati di specificare un insieme di operazioni definite dall'utente valide, permesse sugli oggetti della base di dati.² Un esempio di operazione definita dall'utente potrebbe essere CALCOLA_MV, che può essere applicata a un oggetto di STUDENTE. D'altra parte, operazioni generiche per inserire, cancellare, modificare o recuperare qualsiasi tipo di oggetto sono spesso comprese nelle operazioni di base del modello di dati. Concetti per specificare il comportamento sono fondamentali per i modelli di dati a oggetti, ma si stanno anche incorporando in modelli di dati più tradizionali – attraverso la loro estensione –, ad esempio nei modelli relazionali e a oggetti.

2.1.1 Categorie di modelli di dati

Sono stati proposti molti modelli di dati, classificabili in base ai tipi di concetti da essi utilizzati per descrivere la struttura di una base di dati. I modelli di dati di alto livello o concettuali forniscono concetti che sono vicini al modo in cui molti utenti percepiscono i dati, mentre i modelli di dati di basso livello o fisici forniscono concetti che descrivono i dettagli sul modo in cui i dati sono memorizzati nel computer. I concetti forniti dai modelli di dati di basso livello sono generalmente destinati a specialisti di computer, non a utenti finali tipici. Tra questi due estremi c'è una classe di modelli di dati implementabili che forniscono concetti che possono essere compresi dagli utenti finali ma che non sono troppo lontani dal modo in cui i dati sono organizzati entro il computer: essi nascondono alcuni dettagli di memorizzazione dei dati, ma proprio perché implementabili si possono considerare più vicini al computer stesso.

¹ Talora la parola *modello* è usata per indicare una descrizione, o schema, di una specifica base di dati, per esempio, il "modello di dati per il marketing". Qui non è data al termine tale interpretazione.

² L'inclusione di concetti per descrivere il comportamento riflette una tendenza secondo la quale la progettazione di basi di dati e le attività di progettazione del software vengono via via combinate in una singola attività. Tradizionalmente la specificazione del comportamento è associata con la progettazione del software.

I modelli di dati concettuali usano concetti come entità, attributi e associazioni. Un'entità rappresenta un oggetto o concetto del mondo reale, come un impiegato o un progetto, che è descritto nella base di dati. Un attributo rappresenta una qualche proprietà di interesse che descrive più a fondo un'entità, come il nome o il salario dell'impiegato. Un'associazione tra due o più entità rappresenta un'interazione tra le entità; per esempio un'associazione lavorativa tra un impiegato e un progetto. Nel Capitolo 3 verrà presentato il modello Entità-Associazione (Entity-Relationship), un popolare modello di dati concettuale di alto livello; nel Capitolo 4 vengono descritti altri concetti riguardanti i modelli di dati, come la generalizzazione, la specializzazione e le categorie.

I modelli di dati implementabili sono i modelli usati più frequentemente nei tradizionali DBMS commerciali, e includono il diffuso modello di dati relazionale, così come modelli di dati molto usati in passato e che ora costituiscono una legacy (eredità), cioè i modelli reticolare e gerarchico. I Capitoli 7, 8 e 9 sono dedicati al modello di dati relazionale, alle sue operazioni e linguaggi, mentre il Capitolo 12 offre una visione d'insieme di due sistemi relazionali.³ Lo standard SQL per basi di dati relazionali è descritto in particolare nel Capitolo 8. I modelli di dati implementabili rappresentano dati usando strutture di record e perciò sono talora chiamati modelli di dati basati su record.

È possibile considerare i modelli di dati a oggetti come una nuova famiglia di modelli di dati implementabili a più alto livello che sono più vicini ai modelli di dati concettuali. Essi sono spesso anche utilizzati come modelli concettuali di alto livello, in particolar modo nel campo dell'ingegneria del software.

I modelli di dati fisici descrivono il modo in cui i dati sono memorizzati nel computer rappresentando informazioni come i formati e gli ordinamenti dei record e i cammini di accesso. Un cammino di accesso è una struttura che rende efficiente la ricerca di particolari record della base di dati. Per una trattazione dettagliata delle tecniche di memorizzazione fisica e delle strutture di accesso si rimanda ai Capitoli 5 e 6.

2.1.2 Schemi, istanze e stato di una base di dati

Qualsiasi sia il modello di dati considerato, è importante distinguere tra la descrizione della base di dati e la base di dati stessa. La descrizione è detta schemi della base di dati (database schema)⁴; esso è specificato durante la progettazione della base di dati e non ci si aspetta che cambi frequentemente.⁵ La maggior parte dei modelli di dati seguono certe convenzioni per rappresentare gli schemi come diagrammi. Una rappresentazione grafica di uno schema

³ I modelli di dati reticolare e gerarchico sono descritti nella seconda edizione originale (in lingua inglese) di questo libro, e sono accessibili in Internet nel relativo sito Web.

⁴ Nel gergo in lingua inglese delle basi di dati è consuetudine usare *schemas* come plurale per *schema*, anche se *schemata* sarebbe forma più corretta. La parola *schemi* è talora usata al posto di *schema*.

⁵ I cambiamenti nello schema sono generalmente necessari quando i requisiti delle applicazioni della base di dati cambiano. I sistemi di basi di dati più recenti comprendono operazioni che li rendono possibili, per quanto tale processo sia più complicato dei semplici aggiornamenti della base di dati.

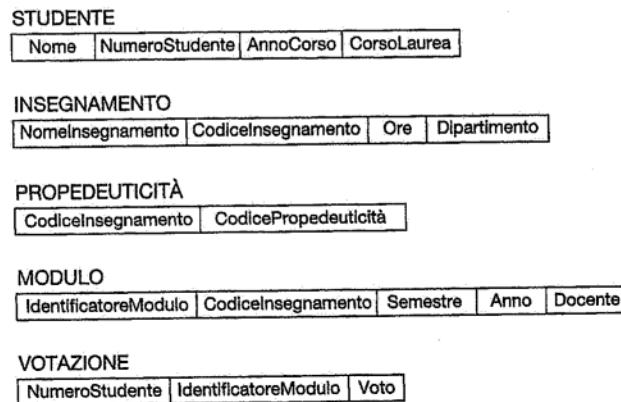


Figura 2.1 Diagramma di schema per la base di dati di Figura 1.2.

è detta diagramma di schema. In Figura 2.1 è mostrato un diagramma di schema per la base di dati di Figura 1.2; il diagramma rappresenta la struttura di ciascun tipo di record ma non le istanze effettive dei record. Si chiamerà ciascun oggetto dello schema – come STUDENTE o INSEGNAMENTO – costrutto dello schema.

Un diagramma di schema rappresenta solo certi aspetti di uno schema, come i nomi dei vari tipi di record e dei dati, e alcuni tipi di vincoli. Altri aspetti non vengono specificati; ad esempio la Figura 2.1 non mostra né il tipo di dati di ciascun dato né le associazioni tra i vari file. Molti tipi di vincoli non sono rappresentati nei diagrammi di schema; per esempio, un vincolo come “gli studenti che stanno frequentando Ingegneria informatica devono superare l’esame dell’insegnamento di codice CS1310 prima della fine del secondo anno di corso” è piuttosto difficile da rappresentare.

I dati veri e propri in una base di dati possono cambiare piuttosto frequentemente; ad esempio, la base di dati mostrata in Figura 1.2 cambia ogni volta che viene aggiunto uno studente o inserito un nuovo voto per uno studente. I dati nella base di dati in un particolare istante di tempo costituiscono lo stato della base di dati o l’istantanea della base di dati. Si parla anche di insieme corrente di occorrenze o istanze nella base di dati. In un certo stato della base di dati ciascun costrutto dello schema ha il proprio insieme corrente di istanze; ad esempio il costrutto STUDENTE conterrà l’insieme di entità (record) di ciascuno studente come sue istanze. Si possono costruire molti stati della base di dati che si accordino a un particolare schema della base di dati stessa. Ogni volta che viene inserito o cancellato un record, o cambiato il valore di un dato in un record, si passa da uno stato della base di dati a un altro.

La distinzione tra schema e stato di una base di dati è molto importante. Quando si definisce una nuova base di dati, si specifica il suo schema solo al DBMS. A questo punto il corrispondente stato della base di dati è lo stato vuoto, senza dati. Si raggiunge lo stato iniziale de... se di ... andando ... per la prima volta popolata e caricata con i dati iniziali. Da

allora in poi, ogni volta che viene eseguita un’operazione di aggiornamento, si ottiene un altro stato della base di dati. Qualsiasi sia l’istante considerato nel tempo, la base di dati ha uno stato corrente.⁶ È in parte responsabilità del DBMS garantire che ogni stato della base di dati sia uno stato valido, cioè uno stato che soddisfi la struttura e i vincoli specificati nello schema. Di conseguenza, specificare uno schema corretto al DBMS è estremamente importante, e lo schema deve essere progettato con la massima cura. Il DBMS memorizza le descrizioni dei costrutti dello schema e dei suoi vincoli – detti anche metadati – nel suo catalogo cosicché il software del DBMS può riferirsi allo schema ogni volta che ne ha bisogno. Lo schema è talora detto intensione, e uno stato della base di dati estensione dello schema.

Per quanto non ci si aspetti, come detto in precedenza, che lo schema cambi frequentemente, non è poi così eccezionale che sia necessario applicare ad esso cambiamenti quelle rare volte in cui cambiano i requisiti dell’applicazione. Ad esempio, si potrebbe decidere che deve essere memorizzato un altro dato in ciascun record di un file, quale la DataNascita nello schema di STUDENTE di Figura 2.1. Ciò è noto come evoluzione dello schema. La maggior parte dei DBMS moderni include alcune operazioni per l’evoluzione dello schema che possono essere eseguite mentre la base di dati è operativa.

2.2 Architettura di un DBMS e indipendenza dei dati

Tre importanti caratteristiche dell’approccio con basi di dati, elencate nel Paragrafo 1.3, sono: (1) isolamento tra programmi e dati (indipendenza programmi-dati e programmi-operazioni); (2) supporto di più viste d’utente; (3) uso di un catalogo per memorizzare la descrizione (schema) della base di dati. In questo paragrafo verrà analizzata un’architettura per sistemi di basi di dati, detta architettura a tre livelli,⁷ che è stata proposta per aiutare a raggiungere e visualizzare tali caratteristiche. Verrà in secondo luogo discusso il concetto di indipendenza dei dati.

2.2.1 L’architettura a tre livelli

L’obiettivo dell’architettura a tre livelli, illustrata in Figura 2.2, è quello di separare le applicazioni dell’utente dalla base di dati fisica. In questa architettura possono essere definiti schemi ai seguenti tre livelli:

1. Il livello interno ha uno schema interno, che descrive la struttura di memorizzazione fisica della base di dati. Lo schema interno usa un modello fisico di dati e descrive i dettagli completi della memorizzazione dei dati e dei cammini di accesso per la base di dati.

⁶ Lo stato corrente è anche detto istantanea corrente della base di dati.

⁷ Si parla anche di architettura ANSI/SPARC, dal nome del comitato che l’ha proposta (Teichritzis e Klug 1978).

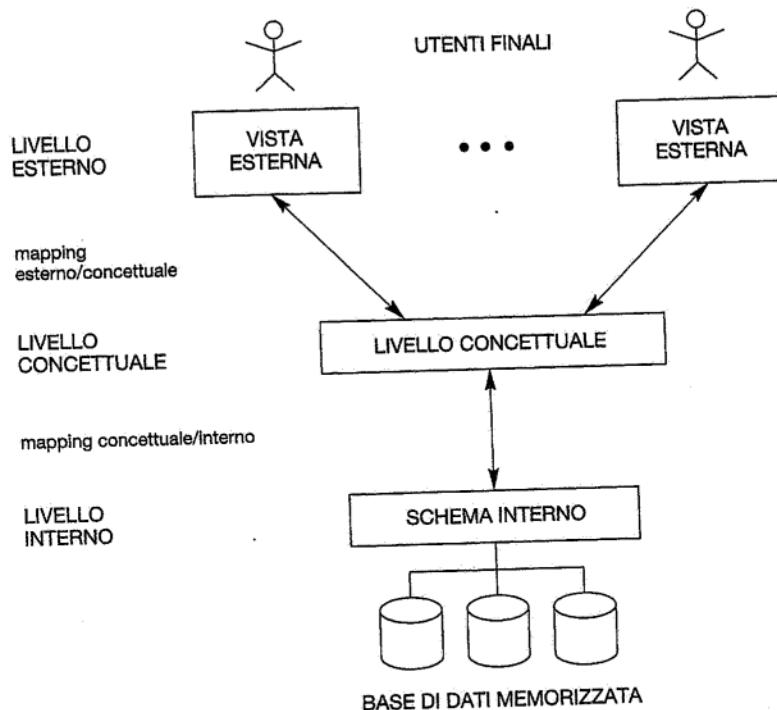


Figura 2.2 Illustrazione dell'architettura a tre livelli.

2. Il livello concettuale ha uno schema concettuale,⁸ che descrive la struttura dell'intera base di dati per una comunità di utenti. Lo schema concettuale nasconde i dettagli delle strutture di memorizzazione fisica e si concentra sulla descrizione di entità, tipi di dati, associazioni, operazioni degli utenti e vincoli. A questo livello può essere usato un modello di dati di alto livello o un modello di dati implementabile.
3. Il livello esterno o di vista comprende un certo numero di schemi esterni o viste d'utente. Ciascuno schema esterno descrive la parte della base di dati a cui è interessato un particolare gruppo di utenti, nascondendo il resto. A questo livello possono essere usati un modello di dati di alto livello o un modello di dati implementabile.

⁸ Anziché di livello concettuale e di schema concettuale in letteratura si parla più frequentemente di livello logico e di schema logico rispettivamente; qui si è preferito rimanere aderenti al testo originale in lingua inglese. (N.d.T.)

L'architettura a tre livelli è uno strumento comodo per l'utente per visualizzare i livelli di schema in un sistema di basi di dati. La maggior parte dei DBMS non separa completamente i tre livelli, ma supporta solo fino a un certo punto questa architettura. Alcuni possono includere dettagli di livello fisico nello schema concettuale. Nella maggior parte di quelli che supportano viste d'utente, gli schemi esterni sono specificati nello stesso modello di dati che descrive le informazioni di livello concettuale. Altri ancora consentono di usare modelli di dati diversi passando da uno schema concettuale a uno schema esterno.

Si noti che i tre schemi sono solo descrizioni di dati; i soli dati che esistono realmente sono a livello fisico. In un DBMS basato sull'architettura a tre livelli ciascun gruppo di utenti fa riferimento solo al proprio schema esterno. Quindi il DBMS deve trasformare una richiesta specificata su uno schema esterno in una richiesta verso lo schema concettuale, e poi in una richiesta verso lo schema interno perché avvenga l'elaborazione sulla base di dati memorizzata. Se la richiesta è un recupero dalla base di dati, i dati estratti dalla base di dati memorizzata devono essere riformattati per accordarsi con la vista esterna dell'utente. I processi di trasformazione di richieste e risultati tra livelli sono detti mapping (trasformazioni). Questi possono essere dispendiosi in termini di tempo, cosicché alcuni DBMS – specialmente quelli destinati a supportare piccole basi di dati – non supportano viste esterne. Pure in tali sistemi, tuttavia, una certa quantità di mapping è necessaria per trasformare le richieste tra i livelli concettuale e interno.

2.2.2 Indipendenza dei dati

L'architettura a tre livelli può essere usata per spiegare il concetto di indipendenza dei dati, che può essere definita come la capacità di cambiare lo schema a un certo livello di un sistema di basi di dati senza dover cambiare lo schema al livello immediatamente superiore. È possibile definire due tipi di indipendenza dei dati.

1. Indipendenza logica dei dati. È la capacità di cambiare lo schema concettuale senza dover cambiare gli schemi esterni o i programmi applicativi. Si può cambiare lo schema concettuale per espandere la base di dati (aggiungendo un tipo di record o un campo) o per ridurla (rimuovendo un tipo di record o un campo). In quest'ultimo caso gli schemi esterni che si riferiscono solo ai dati rimanenti non dovrebbero essere intaccati. Ad esempio lo schema esterno di Figura 1.4(a) non dovrebbe essere influenzato dal cambiamento del file VOTAZIONE di Figura 1.2 in quello mostrato in Figura 1.5(a). Solo la definizione della vista e i mapping devono essere modificati in un DBMS che supporta l'indipendenza logica dei dati. I programmi applicativi che fanno riferimento ai costrutti dello schema esterno devono lavorare come prima, dopo che lo schema concettuale è stato sottoposto a una riorganizzazione logica. Si possono anche eseguire cambiamenti ai vincoli nello schema concettuale senza per questo influenzare gli schemi esterni o i programmi applicativi.
2. Indipendenza fisica dei dati. È la capacità di cambiare lo schema interno senza dover cambiare lo schema concettuale (o gli schemi esterni). Cambiamenti allo schema interno possono essere necessari perché alcuni file fisici devono essere riorganizzati – ad esempio creando strutture di accesso addizionali – per migliorare le prestazioni di recupero o di ag-

giornamento. Se rimangono nella base di dati gli stessi dati di prima, non dovrebbe essere necessario cambiare lo schema concettuale. Ad esempio, fornire un cammino di accesso per migliorare il recupero di record di MODULO (Figura 1.2) tramite Semestre e Anno non dovrebbe richiedere il cambiamento di un'interrogazione come "elena tutti i moduli attivati nell'autunno 1998", anche se l'interrogazione verrebbe eseguita dal DBMS più efficientemente tramite l'utilizzo del nuovo cammino di accesso.

Ogni volta che si ha un DBMS a più livelli, il suo catalogo deve essere ampliato per aggiungere informazioni su come mappare richieste e dati tra i vari livelli. Il DBMS usa software aggiuntivo per realizzare questi mapping riferendosi alle informazioni di mapping nel catalogo. L'indipendenza dei dati è realizzata perché, quando lo schema è cambiato a un certo livello, lo schema al livello immediatamente superiore rimane immutato; solo il mapping tra i due livelli viene cambiato. Perciò i programmi applicativi che fanno riferimento allo schema di più alto livello non devono essere cambiati.

L'architettura a tre livelli può rendere più semplice raggiungere la vera indipendenza dei dati, sia logica sia fisica. Tuttavia, i due livelli di mapping creano un aumento di elaborazione durante la compilazione o l'esecuzione di un'interrogazione o di un programma, portando a inefficienze nel DBMS. A causa di ciò, pochi DBMS hanno implementato completamente tale architettura.

2.3 Linguaggi e interfacce di basi di dati

Nel Paragrafo 1.4 è stata presentata la varietà di utenti di un DBMS. Il DBMS deve fornire linguaggi e interfacce appropriati per ciascuna categoria di utenti. In questo paragrafo verranno esaminati i tipi di linguaggi e interfacce forniti da un DBMS e le categorie di utenti a cui è destinata ciascuna interfaccia.

2.3.1 Linguaggi dei DBMS

Una volta che la progettazione di una base di dati è stata completata e che è stato scelto un DBMS per implementarla, il principale obiettivo diventa quello di specificare gli schemi concettuale e interno e ogni mapping tra i due. In molti DBMS nei quali non è mantenuta una separazione netta tra livelli, un solo linguaggio, detto linguaggio di definizione dei dati (DDL: *data definition language*), è usato dal DBA e dai progettisti per definire entrambi gli schemi. Il DBMS avrà un compilatore DDL la cui funzione sarà quella di elaborare le istruzioni DDL per identificare descrizioni dei costrutti dello schema e per memorizzare la descrizione dello schema nel catalogo del DBMS.

In quei DBMS in cui è mantenuta una netta distinzione tra i livelli concettuale e interno, il DDL è usato per specificare il solo schema concettuale. Un altro linguaggio, il linguaggio di memorizzazione (SDL: *storage definition language*), è usato per spe-

cificare lo schema interno. I mapping tra questi due schemi possono essere specificati nell'uno o nell'altro di questi linguaggi. Per una vera architettura a tre livelli si avrà bisogno di un terzo linguaggio, il linguaggio di definizione delle viste (VDL: *view definition language*), per specificare le viste d'utente e i loro mapping nello schema concettuale, ma nella maggior parte dei DBMS il DDL è usato per definire sia lo schema concettuale sia lo schema esterno.

Una volta che gli schemi sono stati compilati e la base di dati è stata popolata, gli utenti devono avere degli strumenti per manipolare quest'ultima. Le manipolazioni tipiche comprendono il recupero, l'inserimento, la cancellazione e la modifica dei dati. Il DBMS fornisce un linguaggio di manipolazione dei dati (DML: *data manipulation language*) per questi scopi.

Nei DBMS attuali i tipi precedentemente visti di linguaggi di solito non sono considerati linguaggi distinti; piuttosto, è usato un linguaggio integrato globale, che include costrutti per la definizione dello schema concettuale, per la definizione delle viste e per la manipolazione dei dati. La definizione della memorizzazione è tipicamente tenuta separata, dal momento che è usata per definire strutture di memorizzazione fisica al fine di mettere a punto le prestazioni del sistema di basi di dati, ed è usualmente utilizzata dallo staff del DBA. Un tipico esempio di linguaggio globale è il linguaggio per basi di dati relazionali SQL (si veda il Capitolo 8), che rappresenta una combinazione di DDL, VDL e DML, così come istruzioni per la specificazione di vincoli e l'evoluzione dello schema. SDL era un componente di SQL nelle sue prime versioni, ma è stato rimosso per mantenere il linguaggio solo ai livelli concettuale ed esterno.

Ci sono due tipi principali di DML. Un DML di alto livello o nonprocedurale può essere usato da solo per specificare in modo conciso operazioni complesse sulle basi di dati. Molti DBMS consentono che istruzioni DML di alto livello vengano inserite interattivamente da un terminale (o monitor) oppure vengano incapsulate (*embedded*) in un linguaggio di programmazione general-purpose. In quest'ultimo caso le istruzioni DML devono essere riconosciute all'interno del programma così da poter essere estratte da un precompilatore ed elaborate dal DBMS. Un DML di basso livello o procedurale deve essere incapsulato in un linguaggio di programmazione general-purpose. Questo tipo di DML tipicamente recupera singoli record od oggetti della base di dati e li elabora separatamente uno ad uno. Ha quindi bisogno di usare costrutti per linguaggi di programmazione, come l'iterazione, per recuperare ed elaborare ciascun record, prendendolo da un insieme di record. I DML di basso livello sono anche detti DML record-at-a-time (DML un-record-all-at-once) proprio per questa proprietà. I DML di alto livello, come SQL, possono specificare e recuperare molti record in una singola istruzione DML e sono perciò detti set-at-a-time o set-oriented DML (DML un-insieme-all-a-volta od orientati all'insieme). Un'interrogazione in un DML di alto livello spesso specifica quali dati recuperare piuttosto che come recuperarli; perciò questi linguaggi sono anche detti dichiarativi.

Ogni volta che comandi DML, di alto o di basso livello, sono incapsulati in un linguaggio di programmazione general-purpose, quel linguaggio viene detto linguaggio ospite (*host language*) e il DML è detto linguaggio specializzato per la gestione dati.⁹ D'altronde un DML

⁹ Nelle basi di dati a oggetti, il linguaggio ospite e il linguaggio specializzato per la gestione dati formano tipicamente un solo linguaggio integrato – per esempio, C++ con alcune estensioni per supportare la funzionalità per basi di dati. Anche alcuni sistemi relazionali forniscono linguaggi integrati – ad esempio il linguaggio PL/SQL di ORACLE.

di alto livello usato in modo interattivo isolato è detto linguaggio di interrogazione (query language). In generale, sia i comandi di recupero sia quelli di aggiornamento di un DML di alto livello possono essere usati interattivamente e sono perciò considerati parte di un linguaggio di interrogazione.¹⁰

Gli utenti casuali tipicamente usano un linguaggio di interrogazione di alto livello per specificare le loro richieste, mentre i programmati usano il DML nella sua forma encapsulata. Come si vedrà nel prossimo paragrafo, per utenti non sofisticati e parametrici ci sono di norma interfacce amichevoli per interagire con la base di dati; esse possono essere anche usate da utenti casuali o da altri utenti che non vogliono imparare i dettagli di un linguaggio di interrogazione di alto livello.

2.3.2 Interfacce dei DBMS

Le interfacce amichevoli fornite da un DBMS possono comprendere le interfacce seguenti.

Interfacce a menu per la navigazione. Offrono all'utente elenchi di opzioni, detti menu, che guidano l'utente nella formulazione di una richiesta. I menu eliminano la necessità di memorizzare i comandi e la sintassi specifici di un linguaggio di interrogazione; piuttosto, l'interrogazione è composta passo passo selezionando opzioni da un menu che è mostrato dal sistema. L'uso dei menu pull-down (a discesa) sta diventando una tecnica molto popolare per interfacce utente basate su finestre. Sono spesso usati in interfacce per la navigazione, che consentono a un utente di scorrere il contenuto di una base di dati in modo esplorativo e destrutturato.

Interfacce a moduli. Visualizzano un modulo per ciascun utente. Gli utenti possono compilare tutti i campi del modulo per inserire nuovi dati, oppure solo certi campi, nel qual caso il DBMS recupererà, per quanto concerne i campi rimanenti, tutti i dati che si accordano con quanto specificato. I moduli sono usualmente progettati e programmati per utenti non sofisticati come interfacce alle transazioni standard. Molti DBMS dispongono di linguaggi per la specificazione dei moduli, linguaggi speciali che aiutano i programmati a specificare questi moduli. Alcuni sistemi hanno programmi di utilità che consentono di definire un modulo tramite la costruzione interattiva, da parte dell'utente finale, di un modulo campione sullo schermo.

Interfacce grafiche d'utente. Tipicamente mostrano all'utente uno schema in forma di diagramma. L'utente può allora specificare un'interrogazione manipolando quest'ultimo. In molti casi, le interfacce grafiche utilizzano sia menu sia moduli. La maggior parte di esse usa un dispositivo di puntamento, come un mouse, per selezionare certe parti del diagramma di schema visualizzato.

¹⁰ Se si dovesse guardare il significato della parola *query* in inglese (*interrogazione* in italiano), il linguaggio dovrebbe in realtà essere usato per descrivere solo recuperi, non aggiornamenti.

Interfacce in linguaggio naturale. Accettano richieste scritte in inglese o in qualche altra lingua e cercano di "capiile". Un'interfaccia in linguaggio naturale di solito ha un proprio "schema", che è simile allo schema concettuale della base di dati, e fa riferimento alle parole in esso contenute, così come a un insieme di parole standard, per interpretare la richiesta. Se l'interpretazione ha avuto successo, l'interfaccia genera un'interrogazione di alto livello corrispondente alla richiesta in linguaggio naturale e la sottopone al DBMS per l'elaborazione; altrimenti inizia un dialogo con l'utente per chiarire la richiesta.

Interfacce per utenti parametrici. Gli utenti parametrici, ad esempio i cassieri di banca, spesso si trovano a dover eseguire ripetutamente un piccolo insieme di operazioni. Gli analisti di sistema e i programmati progettano e implementano un'interfaccia speciale per una classe nota di utenti non sofisticati. Di solito viene inserito un piccolo insieme di comandi abbreviati, con lo scopo di minimizzare il numero di battute necessarie per ciascuna richiesta (ad esempio, i tasti funzionali in un terminale possono essere programmati per dare inizio ai vari comandi).

Interfacce per il DBA. La maggior parte dei sistemi di basi di dati contiene comandi privilegiati che possono essere usati solo dallo staff del DBA. Essi includono comandi per creare account, per fissare parametri del sistema, per concedere autorizzazioni agli account, per cambiare uno schema e riorganizzare le strutture di memorizzazione di una base di dati.

2.4 L'ambiente di un sistema di basi di dati

Un DBMS è un sistema software complesso. In questo paragrafo verranno analizzati i tipi di componenti software che costituiscono un DBMS e i tipi di software di sistema con cui il DBMS interagisce.

2.4.1 Moduli componenti un DBMS

In Figura 2.3 sono illustrati, in forma semplificata, i tipici componenti di un DBMS. La base di dati e il catalogo del DBMS sono solitamente memorizzati su disco. L'accesso al disco è controllato essenzialmente dal sistema operativo (OS: operating system), che ne pianifica l'input/output. Un gestore dei dati memorizzati, modulo del DBMS a più alto livello, controlla l'accesso alle informazioni del DBMS memorizzate su disco, siano esse parte della base di dati o del catalogo. Le linee tratteggiate e i cerchi contrassegnati con A, B, C, D ed E in Figura 2.3 illustrano accessi che sono sotto il suo controllo. Ancora, esso può usare servizi di base del sistema operativo per effettuare trasferimenti dati di basso livello tra il disco e la memoria esterna principale del computer, ma controlla pure altri aspetti del trasferimento dati, come la gestione di buffer in memoria principale. Una volta che i dati sono nei buffer di me-

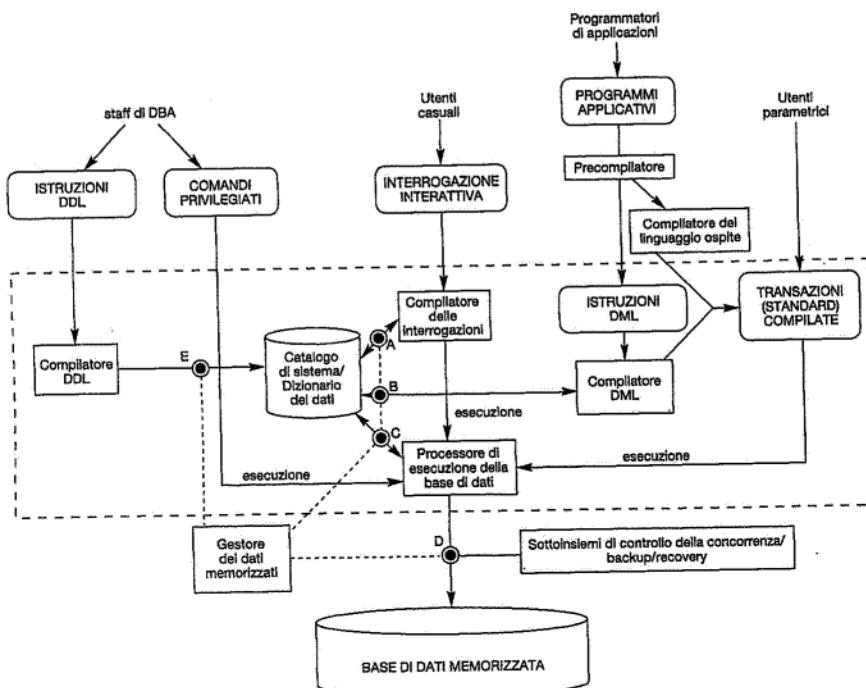


Figura 2.3 Tipici moduli componenti un DBMS. Le linee tratteggiate indicano accessi che sono sotto il controllo del gestore dei dati memorizzati.

memoria principale, essi possono essere elaborati da altri moduli del DBMS, così come da programmi applicativi.

Il compilatore DDL elabora definizioni di schema, specificate nel DDL, e memorizza descrizioni degli schemi (metadati) nel catalogo del DBMS. Il catalogo include informazioni come i nomi dei file, quelli dei dati, i dettagli di memorizzazione di ciascun file, informazioni sul mapping tra i vari schemi, e vincoli, in aggiunta a molti altri tipi di informazioni necessarie ai moduli del DBMS. I moduli software del DBMS, dunque, cercano le informazioni di catalogo quando ne hanno bisogno.

Il processore di esecuzione della base di dati gestisce gli accessi alla base di dati durante l'esecuzione: riceve operazioni di recupero o aggiornamento e le esegue sulla base di dati. L'accesso al disco passa attraverso il gestore dei dati memorizzati. Il compilatore delle interrogazioni gestisce le interrogazioni di alto livello che sono inserite interattivamente. Esso scandisce, analizza e compila o interpreta un'interrogazione creando codice di accesso alla base di dati e quindi genera chiamate al processore di esecuzione per eseguire il codice.

Il precompilatore estrae comandi DML da un programma applicativo scritto in un linguaggio ospite di programmazione. Questi comandi sono inviati al compilatore DML per la compilazione in codice oggetto per l'accesso alla base di dati. Il resto del programma è inviato al compilatore del linguaggio ospite. Quindi il codice oggetto per i comandi DML e quello per il resto del programma sono messi in collegamento (linked), formando una transazione standard il cui codice eseguibile include chiamate al processore di esecuzione della base di dati.

Con la Figura 2.3 non si è inteso descrivere uno specifico DBMS, quanto piuttosto illustrare tipici moduli DBMS. Il DBMS interagisce con il sistema operativo quando sono necessari accessi al disco – alla base di dati o al catalogo. Se il sistema di computer è condiviso da molti utenti, il sistema operativo pianificherà le richieste di accesso al disco e le elaborazioni del DBMS insieme con altri processi. Il DBMS interagisce anche con i compilatori per i linguaggi ospite di programmazione general-purpose. Interfacce utente amichevoli al DBMS possono essere fornite per aiutare ogni tipo di utente di Figura 2.3 a specificare le sue esigenze.

2.4.2 Programmi di utilità di un sistema di basi di dati

Oltre a disporre dei moduli software appena descritti, la maggior parte dei DBMS dispone di programmi di utilità per basi di dati che aiutano il DBA a gestire il sistema di basi di dati. I comuni programmi di utilità svolgono i seguenti tipi di funzioni.

1. Caricamento. Un'utilità di caricamento è usata per caricare file di dati già esistenti – come file di testo o file sequenziali – nella base di dati. Di solito il formato corrente (sorgente) dei file di dati e la struttura desiderata (obiettivo) dei file della base di dati sono specificati al programma di utilità, che quindi traduce automaticamente i dati in formati diversi e li memorizza nella base di dati. Con la proliferazione dei DBMS il trasferimento dei dati da un DBMS a un altro sta diventando comune in molte organizzazioni. Alcune aziende offrono prodotti che generano programmi di caricamento appropriati, date le descrizioni esistenti della modalità di memorizzazione nella base di dati sorgente e obiettivo (schemi interni). Strumenti di questo tipo sono anche chiamati strumenti di conversione.
2. Backup. Un'utilità di backup crea una copia di backup della base di dati, di solito riversando l'intera base di dati su nastro. La copia di backup può essere usata per ripristinare la base di dati in caso di guasto catastrofico. Spesso si usano anche backup incrementali, nei quali sono registrati solo i cambiamenti dal backup precedente. Il backup incrementale è più complesso ma consente di risparmiare spazio.
3. Riorganizzazione dei file. Questo programma di utilità può essere usato per cambiare l'organizzazione di un file della base di dati, al fine di incrementare le prestazioni.
4. Monitoraggio delle prestazioni. Una tale utilità controlla sistematicamente l'uso della base di dati e fornisce statistiche al DBA, il quale ricorre ad esse per prendere decisioni, come quella se riorganizzare o meno i file per migliorare le prestazioni.

Altre utilità possono essere disponibili, tra l'altro, per ordinare i file, gestire la compressione dei dati, controllare l'accesso da parte degli utenti.

2.4.3 Strumenti, ambienti applicativi e funzioni di comunicazione

Spesso sono disponibili altri strumenti per i progettisti e per gli utenti di basi di dati, nonché per i DBA. Strumenti CASE¹¹ sono usati nella fase di progettazione di sistemi di basi di dati. Un altro strumento che può avere una certa utilità per grandi organizzazioni è un più ampio sistema di dizionario dei dati (o deposito di dati). Oltre a immagazzinare informazioni di catalogo sugli schemi e i vincoli, il dizionario dei dati immagazzina altre informazioni, come decisioni di progettazione, standard d'uso, descrizioni di programmi applicativi e informazioni per gli utenti. Un sistema di questo tipo è anche detto deposito di informazioni. A queste informazioni possono accedere direttamente gli utenti o il DBA quando esse si rendano necessarie. Un'utilità di dizionario dei dati è simile al catalogo del DBMS, ma include una più ampia varietà di informazioni, e ad essa accedono principalmente gli utenti piuttosto che il software del DBMS.

Ambienti di sviluppo applicazioni, come il sistema PowerBuilder, stanno diventando piuttosto popolari. Questi sistemi forniscono un ambiente per lo sviluppo di applicazioni di basi di dati e includono funzioni che sono di utilità per molti aspetti relativi ai sistemi di basi di dati, tra cui la progettazione di basi di dati, lo sviluppo di interfacce grafiche, le interrogazioni e gli aggiornamenti, nonché lo sviluppo di programmi applicativi.

Il DBMS ha anche bisogno di interfacciarsi con il software di comunicazione, la cui funzione è quella di consentire agli utenti in postazioni remote rispetto al sito del sistema di basi di dati di accedere alla base di dati tramite terminali di computer, workstation, o i loro personal computer locali. Questi sono connessi al sito della base di dati tramite hardware per il trasferimento dati come linee telefoniche, reti a lungo raggio, reti locali, o dispositivi di comunicazione via satellite. Molti sistemi di basi di dati commerciali hanno pacchetti di comunicazione che lavorano con il DBMS. Il sistema integrato DBMS e trasferimento dati è detto sistema DB/DC (database/data communications). Inoltre, alcuni DBMS distribuiti sono ripartiti fisicamente tra più macchine. In questo caso, per connettere le macchine sono necessarie reti di comunicazione, spesso reti locali (LANs: local area networks), ma anche di altro tipo.

2.5 Classificazione dei sistemi di gestione di basi di dati

Possono essere usati svariati criteri per classificare i DBMS. Il primo è quello del modello di dati su cui è basato il DBMS. I due tipi di modelli di dati usati attualmente in molti DBMS commerciali sono il modello di dati relazionale e il modello di dati a oggetti. Molte applicazioni lasciateci in eredità operano ancora su sistemi di basi di dati basati sul modello di dati.

ti gerarchico e su quello reticolare. I DBMS relazionali stanno evolvendo continuamente e, in particolare, stanno via via incorporando molti dei concetti sviluppati nelle basi di dati a oggetti. Ciò ha portato a una nuova classe di DBMS: si parla infatti di DBMS relazionali e a oggetti. Basandosi sul modello di dati è perciò possibile classificare i DBMS in: relazionali, a oggetti, relazionali e a oggetti, gerarchici, reticolari e altri.

Il secondo criterio usato per classificare i DBMS è quello del numero di utenti supportato dal sistema. Sistemi a utente singolo supportano solo un utente alla volta e sono per lo più usati con personal computer. Sistemi multiutente, che includono la maggior parte dei DBMS, supportano più utenti simultaneamente.

Un terzo criterio è il numero di siti sui quali è distribuita la base di dati. Un DBMS è centralizzato se i dati sono memorizzati in un singolo sito di computer. Un DBMS centralizzato può supportare utenti multipli, ma il DBMS e la stessa base di dati risiedono totalmente in un singolo sito di computer. Un DBMS distribuito (DDBMS; distributed DBMS) può avere la base di dati vera e propria e il software distribuiti su molti siti, connessi da una rete di computer. DDBMS omogenei usano lo stesso software in più siti. Una recente tendenza è quella di sviluppare software per accedere a numerose basi di dati autonome preesistenti memorizzate sotto DBMS eterogenei. Ciò porta a un DBMS federato (o sistema a basi di dati multiple), in cui i DBMS partecipanti sono debolmente accoppiati e godono di un certo grado di autonomia locale. Molti DDBMS usano un'architettura client-server.

Un quarto criterio è il costo del DBMS. La maggior parte dei DBMS costa tra i 10.000 e i 100.000 dollari. Sistemi a utente singolo e a basso costo, che operano su microcomputer, costano tra 100 e 3000 dollari. Al lato opposto c'è qualche raro pacchetto particolarmente sofisticato che costa più di 100.000 dollari.

Si può classificare un DBMS anche sulla base delle opzioni per i tipi di cammini di accesso per memorizzare file. Una famiglia ben nota di DBMS è basata sulle strutture di file traspesi. Infine, un DBMS può essere general-purpose o special-purpose. Quando sono di primaria importanza le prestazioni, un DBMS special-purpose può essere progettato e costruito per una specifica applicazione; un tale sistema non può essere usato per altre applicazioni senza modifiche radicali. Molti sistemi di prenotazione aerea e di elenchi telefonici sviluppati nel passato sono DBMS special-purpose. Questi ricadono nella categoria dei sistemi di elaborazione delle transazioni in linea (OLTP), che devono supportare un gran numero di transazioni concorrenti senza poter imporre ritardi eccessivi.

Si ragiona ora brevemente sul principale criterio di classificazione dei DBMS: il modello di dati. Il modello di dati relazionale di base rappresenta una base di dati come una collezione di tabelle, dove ciascuna tabella può essere memorizzata come un file separato. La base di dati in Figura 1.2 è presentata in modo molto simile a una rappresentazione relazionale. La maggior parte delle basi di dati relazionali usa il linguaggio di interrogazione di alto livello detto SQL e supporta una forma limitata di viste d'utente (per l'analisi del modello relazionale, dei suoi linguaggi e delle sue operazioni, nonché di due sistemi commerciali campione, si rinvia ai Capitoli 7, 8, 9 e 12).

Il modello di dati a oggetti definisce una base di dati in termini di oggetti, delle loro proprietà e delle loro operazioni. Oggetti con la stessa struttura e comportamento appartengono a una classe, e le classi sono organizzate in gerarchie (o grafi aciclici). Le operazioni di ciascuna classe sono specificate in termini di procedure predefinite dette metodi. I DBMS relazionali hanno nel corso del tempo esteso il loro modello per incorporare concetti di basi di dati.

¹¹ Benché CASE stia per Computer Aided Software Engineering (ingegneria del software assistita dal computer), molti strumenti CASE sono usati principalmente per la progettazione di basi di dati.

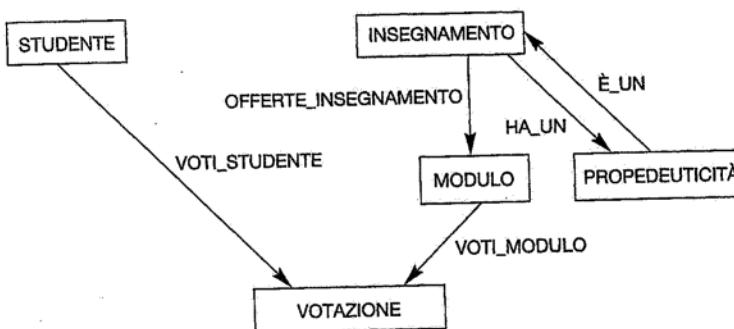


Figura 2.4 Lo schema di Figura 2.1 nella notazione del modello di dati reticolare.

ti a oggetti e altri servizi: questi sistemi sono detti relazionali e a oggetti o sistemi relazionali estesi.

Il modello reticolare rappresenta i dati come tipi di record e un tipo limitato di associazione 1:N, detta tipo insieme. In Figura 2.4 è mostrato un diagramma di schema reticolare per la base di dati di Figura 1.2, in cui i tipi record sono presentati come rettangoli e i tipi insieme come archi orientati etichettati. Il modello reticolare, noto anche come modello CODASYL DBTG,¹² ha un linguaggio associato record-at-a-time che deve essere incapsulato in un linguaggio di programmazione ospite. Il modello gerarchico rappresenta i dati come strutture ad albero gerarchiche. Ciascuna gerarchia rappresenta un certo numero di record correlati. Non c'è nessun linguaggio standard per il modello gerarchico, anche se la maggior parte dei DBMS gerarchici dispone di linguaggi record-at-a-time.¹³

Sommario

In questo Capitolo abbiamo introdotto i concetti fondamentali usati nei sistemi di basi di dati. Abbiamo dato la definizione di modello di dati, distinguendone tre categorie principali:

- modelli di dati di alto livello o concettuali (basati su entità e associazioni);
- modelli di dati di basso livello o fisici;
- modelli di dati implementabili (basati su record, a oggetti).

¹² CODASYL DBTG sta per Computer Data Systems Language Data Base Task Group (linguaggio per sistemi di dati per computer – task group per basi di dati), che è il comitato che ha specificato il modello reticolare e il suo linguaggio.

¹³ I capitoli completi sui modelli reticolare e gerarchico della seconda edizione originale (in lingua inglese) di questo libro sono disponibili in Internet nel relativo sito Web.

Abbiamo poi distinto lo schema, o descrizione di una base di dati, dalla base di dati stessa. Lo schema non cambia molto spesso, mentre invece lo stato della base di dati cambia ogni volta che vengono inseriti, cancellati o modificati dati. Abbiamo quindi descritto l'architettura a tre livelli di un DBMS, che consente tre livelli di schemi:

- uno schema interno descrive la struttura di memorizzazione fisica della base di dati;
- uno schema concettuale fornisce una descrizione di alto livello dell'intera base di dati;
- schemi esterni descrivono le viste di diversi gruppi di utenti.

Un DBMS che separa in modo netto i tre livelli deve disporre di mapping tra gli schemi per trasformare richieste e risultati da un livello al successivo. La maggior parte dei DBMS non separa completamente i tre livelli. Abbiamo usato l'architettura a tre livelli per definire i concetti di indipendenza dei dati logica e fisica.

Sono stati poi studiati i tipi principali di linguaggi e interfacce supportati dai DBMS. Un linguaggio di definizione dei dati (DDL) è usato per definire lo schema concettuale della base di dati. Nella maggior parte dei DBMS il DDL definisce anche le viste d'utente e, talora, le strutture di memorizzazione; in altri DBMS a tale scopo possono esistere linguaggi separati (VDL, SDL). Il DBMS compila tutte le definizioni di schemi e memorizza le loro descrizioni nel suo catalogo. Un linguaggio di manipolazione dati (DML) è usato per specificare recuperi e aggiornamenti effettuati sulla base di dati. I DML possono essere di alto livello (set-oriented, nonprocedurali) o di basso livello (record-oriented, procedurali). Un DML di alto livello può essere incapsulato in un linguaggio di programmazione ospite, o essere usato come linguaggio indipendente; in quest'ultimo caso è spesso detto linguaggio di interrogazione.

Si sono esaminati diversi tipi di interfacce fornite dai DBMS, e i tipi di utenti di DBMS ai quali ciascuna interfaccia è associata. Abbiamo poi esaminato l'ambiente del sistema di basi di dati, tipici moduli software di DBMS e i programmi di utilità dei DBMS che aiutano gli utenti e il DBA a eseguire i loro compiti.

Nell'ultimo paragrafo abbiamo classificato i DBMS seguendo diversi criteri: modello di dati, numero di utenti, numero di siti, costo, tipi di cammini di accesso e generalità. La principale classificazione dei DBMS è basata sul modello di dati. Abbiamo infine brevemente discusso i principali modelli di dati usati attualmente nei DBMS commerciali.

Questionario di verifica

- 2.1. Si definiscano i seguenti termini: *modello di dati, schema di una base di dati, stato di una base di dati, schema interno, schema concettuale, schema esterno, indipendenza dei dati, DDL, DML, SDL, VDL, linguaggio di interrogazione, linguaggio ospite, linguaggio specializzato per la gestione dati, programma di utilità per base di dati, catalogo, architettura client-server.*
- 2.2. Si discutano le principali categorie di modelli di dati.
- 2.3. Qual è la differenza tra uno schema di base di dati e uno stato di base di dati?
- 2.4. Si descriva l'architettura a tre livelli. Perché abbiamo bisogno di mapping tra i vari livelli di schemi? Come supportano questa architettura i diversi linguaggi di definizione di schemi?

- 2.5. Qual è la differenza tra indipendenza dei dati logica e indipendenza dei dati fisica?
- 2.6. Qual è la differenza tra DML procedurali e nonprocedurali?
- 2.7. Si presentino i diversi tipi di interfacce amichevoli e i tipi di utenti che solitamente li usano.
- 2.8. Con quale altro software di sistema interagisce un DBMS?
- 2.9. Si discutano alcuni tipi di programmi di utilità e di strumenti di una base di dati e le rispettive funzioni.

Esercizi

- 2.10. Si pensi a diversi utenti possibili per la base di dati di Figura 1.2. Di quali tipi di applicazioni avrebbe bisogno ciascun utente? A quale categoria di utenti apparterrebbe ciascuno di essi e di quale tipo di interfaccia necessiterebbe?
- 2.11. Si scelga un'applicazione di basi di dati con cui si abbia familiarità. Si progetti uno schema e si mostri una base di dati campione per quella applicazione, usando le notazioni delle Figure 2.1 e 1.2. Quali altri tipi di informazioni e vincoli si potrebbero rappresentare nello schema? Si pensi a diversi utenti per la base di dati e si progettino una vista per ciascuno di essi.

Bibliografia selezionata

Molti libri di testo sulle basi di dati, tra cui Date (1995), Silberschatz e altri (1998), Ramakrishnan (1997), Ullman (1988, 1989) e Abiteboul e altri (1995), forniscono un'analisi dei vari concetti sulle basi di dati qui presentati. Quello di Tsichritzis e Lochovsky (1982) è uno dei primi libri di testo sui modelli di dati. Tsichritzis e Klug (1978) e Jardine (1977) presentano l'architettura a tre livelli, che è stata suggerita per la prima volta nel rapporto DBTG CODASYL (1971) e più tardi in un rapporto dell'American National Standards Institute (ANSI, l'Istituto nazionale americano per gli standard) (1975). Un'analisi approfondita del modello di dati relazionale e di alcune delle sue possibili estensioni è fornita in Codd (1990). Lo standard proposto per le basi di dati a oggetti è descritto in Cattell (1997).

Un esempio di programmi di utilità per basi di dati è l'Extract Toolkit di ETI (www.eti.com) e lo strumento di amministrazione di basi di dati DB Artisan di Embarcadero Technologies (www.embarcadero.com).

Capitolo 3

Uso del modello Entità-Associazione per modellare i dati

La modellazione concettuale costituisce una fase importante nella progettazione di una buona applicazione di basi di dati. In genere la locuzione applicazione di basi di dati si riferisce a una base di dati specifica – ad esempio, una base di dati BANCA che tiene traccia dei conti dei clienti – e ai programmi associati che ne implementano le interrogazioni e gli aggiornamenti – ad esempio, programmi che implementano aggiornamenti in corrispondenza ai depositi e ai prelievi dei clienti. Questi programmi spesso forniscono interfacce grafiche d'utente (GUI) amichevoli, che utilizzano moduli e menu. Quindi parte dell'applicazione di basi di dati richiederà la progettazione, l'implementazione e il test di questi programmi applicativi. Tradizionalmente la progettazione e il test di programmi applicativi sono stati considerati come facenti parte del dominio dell'ingegneria del software più che delle basi di dati. Oggi però sta diventando sempre più chiaro che c'è una certa vicinanza tra le metodologie di progettazione delle basi di dati e quelle proprie dell'ingegneria del software. Dal momento che le prime tendono sempre più a includere concetti per specificare operazioni su oggetti di basi di dati, e che le seconde specificano in sempre maggior dettaglio la struttura delle basi di dati che saranno usate e a cui accederanno i programmi software, è certo che tale vicinanza aumenterà. Discuteremo brevemente alcuni dei concetti per specificare le operazioni sulle basi di dati nel Capitolo 4.

In questo Capitolo seguiremo l'approccio tradizionale, che consiste nel concentrarsi sulle strutture e sui vincoli della base di dati durante la sua progettazione. Presenteremo i concetti di modellazione propri del modello Entità-Associazione (ER: Entity-Relationship),¹ un popolare modello di dati concettuale di alto livello. Questo modello e le sue variazioni sono frequentemente usati per la progettazione concettuale di applicazioni di basi di dati, e molti stru-

¹ Nella letteratura delle basi di dati in lingua italiana si usa anche parlare di modello Entità-Relazione; qui non verrà mai adottata questa terminologia per non ingenerare confusione con concetti che sono propri del modello di dati relazionale e che saranno introdotti nel seguito. (N.d.T.)

menti di progettazione di basi di dati impiegano i suoi concetti. Noi descriveremo i concetti di base di strutturazione dei dati e i vincoli del modello ER e discuteremo il loro uso nella progettazione di schemi concettuali per applicazioni di basi di dati.

Il presente capitolo è organizzato come segue. Nel Paragrafo 3.1 discuteremo il ruolo dei modelli di dati concettuali di alto livello nella progettazione di basi di dati. Introdurremo i requisiti per un'applicazione di basi di dati esemplificativa nel Paragrafo 3.2, per illustrare l'uso dei concetti del modello ER. Questa base di dati sarà usata anche nei capitoli successivi. Nel Paragrafo 3.3 presenteremo i concetti di entità e attributo, e introdurremo gradualmente la tecnica diagrammatica per la rappresentazione grafica di uno schema ER. Nel Paragrafo 3.4 introduciamo i concetti relativi alle associazioni binarie e ai loro ruoli e quelli relativi ai vincoli strutturali, nel Paragrafo 3.5 i tipi di entità debole. Il Paragrafo 3.6 mostra come un progetto di uno schema venga raffinato per includere associazioni. Il Paragrafo 3.7 esamina la notazione per i diagrammi ER, riassume i problemi che sorgono nella progettazione di uno schema e studia come scegliere i nomi per i costrutti dello schema della base di dati.

Il materiale dei Paragrafi 3.3 e 3.4 fornisce una descrizione piuttosto dettagliata, e può in parte essere omesso in un corso introduttivo. Viceversa, se il lettore desidera una copertura più approfondita dei concetti relativi alla costruzione dei modelli di dati e della progettazione concettuale di basi di dati, dopo aver concluso il capitolo, dovrebbe continuare con il Capitolo 4. Qui descriveremo estensioni del modello ER che portano al modello ER Esteso (EER: Enhanced – ER), che comprende concetti come la specializzazione, la generalizzazione, l'ereditarietà e i tipi unione (categorie), e introdurremo anche la modellazione a oggetti e la notazione del linguaggio universale di modellazione (UML: Universal Modeling Language), che è stato proposto come uno standard per la modellazione a oggetti.

3.1 Uso di modelli di dati concettuali di alto livello per la progettazione di basi di dati

In Figura 3.1 viene mostrata una descrizione semplificata del processo di progettazione di una base di dati. Il primo passo è quello di raccolta e analisi dei requisiti. Durante questa fase i progettisti intervistano i futuri utenti della base di dati per capire e documentare i loro requisiti sui dati. Il risultato di questa fase è un insieme, descritto concisamente, di requisiti degli utenti, che dovrebbero essere specificati nella forma più dettagliata e completa possibile. In parallelo è utile specificare i requisiti funzionali noti dell'applicazione. Questi consistono nelle operazioni (o transazioni) definite dall'utente che saranno eseguite sulla base di dati, e comprendono sia le operazioni di recupero sia quelle di aggiornamento. Nella progettazione del software per specificare i requisiti funzionali è comune usare diagrammi di flusso dati, diagrammi di sequenza, scenari e altre tecniche.

Una volta che tutti i requisiti sono stati raccolti e analizzati, il passo successivo consiste nel creare uno schema concettuale per la base di dati, usando un modello di dati concettuali di alto livello. Questa fase è detta di progettazione concettuale. Lo schema concettuale è una descrizione concisa dei requisiti sui dati degli utenti e comprende descrizioni dettagliate dei tipi di entità, delle associazioni e dei vincoli: questi sono espressi usando i concetti forniti dal

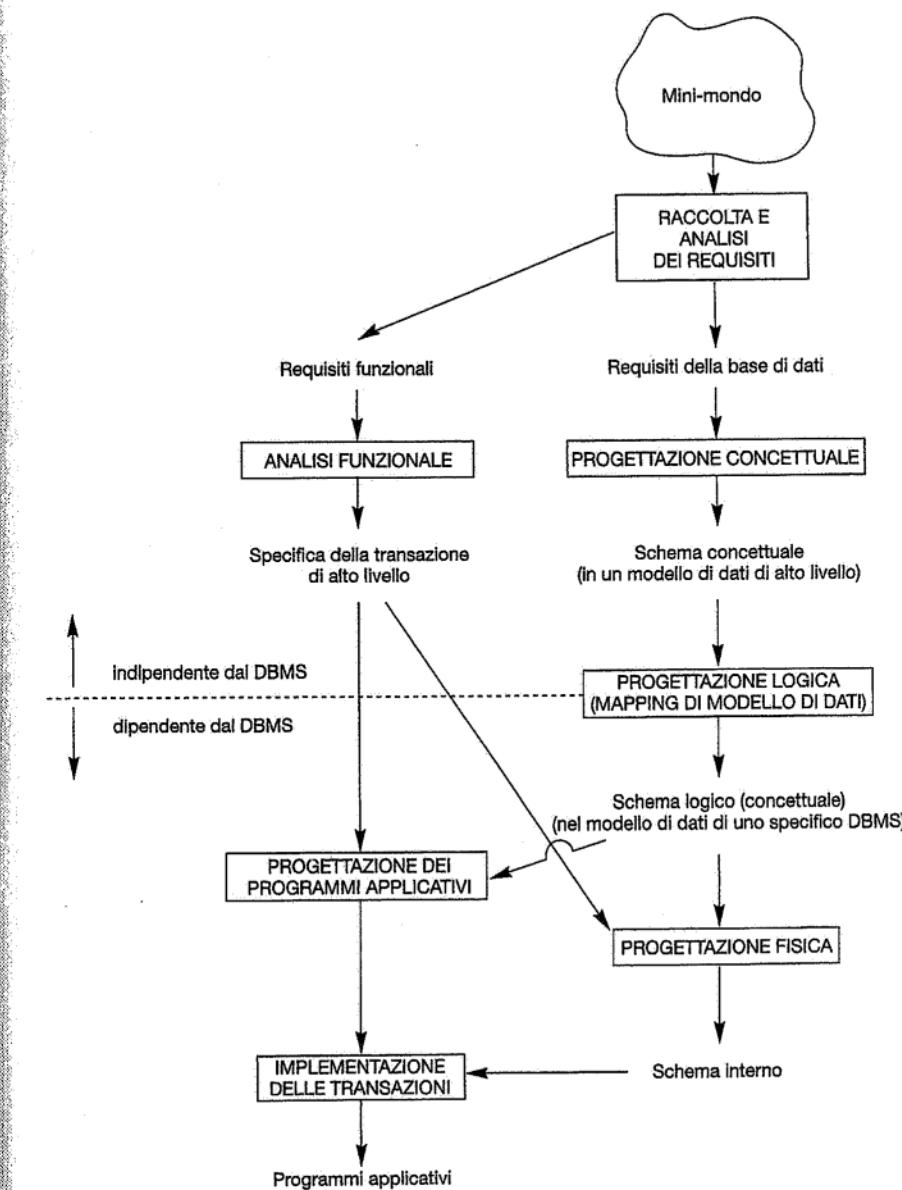


Figura 3.1 Diagramma semplificato per l'illustrazione delle fasi principali della progettazione di una base di dati.

modello di dati di alto livello. Poiché non comprendono dettagli implementativi, essi sono generalmente più semplici da comprendere e possono essere usati per comunicare con utenti non tecnici. Lo schema concettuale di alto livello può anche essere usato come riferimento per assicurarsi che si siano considerati tutti i requisiti degli utenti sui dati e che i requisiti non prevedano conflitti. Questo approccio consente al progettista della base di dati di concentrarsi sulla specificazione delle proprietà dei dati, senza doversi preoccupare dei dettagli di memorizzazione. Di conseguenza è più facile che egli possa arrivare a un buon progetto concettuale della base di dati.

Nel corso della progettazione dello schema concettuale o dopo di essa, le operazioni di base del modello di dati possono essere usate per specificare le operazioni d'utente di alto livello individuate durante l'analisi funzionale. Ciò serve anche per confermare che lo schema concettuale soddisfi tutti i requisiti funzionali individuati. Possono essere introdotte modifiche allo schema concettuale se alcuni requisiti funzionali non possono essere specificati nello schema iniziale.

Il passo successivo nella progettazione di basi di dati consiste nell'implementazione effettiva della base di dati, usando un DBMS commerciale. La maggior parte dei DBMS commerciali attuali usa un modello di dati implementabile, come il modello relazionale o quello di basi di dati a oggetti - cosicché lo schema concettuale subisce una trasformazione dal modello di dati di alto livello nel modello di dati implementabile. Questa fase è detta progettazione logica o mapping di modelli di dati, e il suo risultato è uno schema della base di dati nel modello di dati implementabile del DBMS.

Infine, l'ultimo passo consiste nella progettazione fisica, durante la quale sono specificate le strutture di memorizzazione interna, i cammini di accesso e le organizzazioni dei file della base di dati. In parallelo con queste attività i programmi applicativi sono progettati e implementati come transazioni della base di dati corrispondenti alle specifiche delle transazioni di alto livello.

In questo capitolo saranno presentati solo i concetti del modello ER per la progettazione dello schema concettuale. L'inserimento di operazioni definite dall'utente sarà discussa nel Capitolo 4, quando verrà introdotta la modellazione a oggetti.

3.2 Un'applicazione esemplificativa di basi di dati

In questo paragrafo verrà descritta l'applicazione di basi di dati AZIENDA, la quale servirà a illustrare i concetti del modello ER e il loro uso nella progettazione dello schema. Qui saranno elencati i requisiti sui dati per la base di dati, mentre il suo schema concettuale verrà costruito gradualmente, man mano che verranno introdotti i concetti di rappresentazione del modello ER. La base di dati AZIENDA tiene traccia degli impiegati,² dei dipartimenti e dei progetti di

² Il termine utilizzato dagli autori (*employee*) sarebbe reso più correttamente in italiano con *dipendente*, ma l'esempio proposto nel testo si focalizza poi in particolare sul tipo di dipendenti *impiegati*. (N.d.T.)

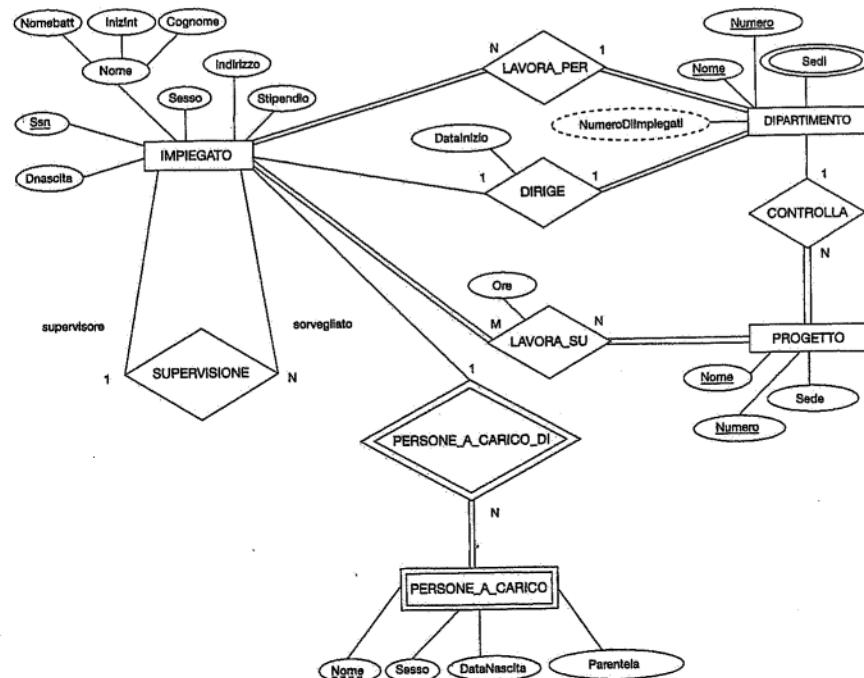


Figura 3.2 Diagramma di schema ER per la base di dati AZIENDA.

un'azienda. Si supponga che, dopo la fase di raccolta e analisi dei requisiti, i progettisti della base di dati abbiano dato del "mini-mondo", la parte dell'azienda che deve essere rappresentata con la base di dati, la descrizione seguente.

1. L'azienda è organizzata in dipartimenti, ciascuno dei quali ha un nome e un numero univoci, più sedi e un particolare impiegato da cui viene gestito. Si registrerà la data in cui quell'impiegato ha cominciato a gestire il dipartimento.
2. Un dipartimento controlla un certo numero di progetti, ciascuno dei quali ha un nome e un numero univoci e una singola sede.
3. Per ciascun impiegato verrà memorizzato il nome, il numero di previdenza sociale,³ l'indirizzo, lo stipendio, il sesso e la data di nascita. Un impiegato è assegnato a un solo dipartimento.

³ Il numero di previdenza sociale, o SSN (*social security number*), è un identificatore univoco a nove cifre assegnato negli Stati Uniti a ciascun individuo per tener traccia del suo impiego, delle sue indennità e delle sue imposte. Altri Stati possono avere schemi identificativi simili, come ad esempio il numero della carta d'identità.

partimento ma può lavorare su molti progetti, non necessariamente controllati dallo stesso dipartimento. Si terrà traccia del numero di ore settimanali lavorate da un impiegato su ciascun progetto e del diretto supervisore di ciascun impiegato.

4. Si registreranno anche le persone a carico di ciascun impiegato a scopi assicurativi e per ciascuna di esse verranno memorizzati il nome di battesimo, il sesso, la data di nascita e il rapporto di parentela con l'impiegato.

In Figura 3.2 è illustrato come lo schema per questa applicazione di basi di dati possa essere rappresentato con notazioni grafiche note come **diagrammi ER**.

3.3 Tipi di entità, insiemi di entità, attributi e chiavi

3.3.1 Entità e attributi

Entità e loro attributi. L'oggetto base rappresentato dal modello ER è l'**entità**, che è una "cosa" del mondo reale con un'esistenza indipendente. Un'entità può essere un oggetto con un'esistenza fisica – una particolare persona, automobile, casa o un certo impiegato – o essere un oggetto che esiste a livello concettuale – un'azienda, un lavoro o un corso universitario. Ciascuna entità ha degli **attributi** – le proprietà particolari che la descrivono. Ad esempio, un'entità impiegato può essere descritta da nome, età, indirizzo, stipendio e lavoro dell'impiegato. Un'entità particolare avrà un **valore** per ciascuno dei suoi attributi. I valori degli attributi che descrivono ciascuna entità divengono parte importante dei dati memorizzati nella base di dati.

In Figura 3.3 sono mostrati due entità e i valori dei rispettivi attributi. L'entità impiegato e_1 ("e" come employee) ha quattro attributi: Nome, Indirizzo, Età e TelefonoDiCasa; i loro valori sono "John Smith", "2311 Kirby, Houston, Texas 77001", "55" e "713-749-2630", ri-

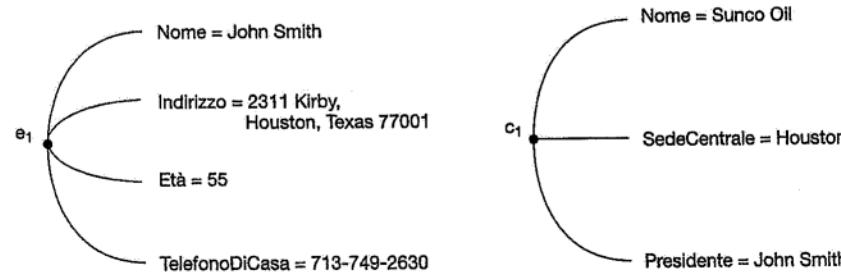


Figura 3.3 Due entità, un impiegato e_1 e un'azienda c_1 , e i valori dei loro attributi.



Figura 3.4 Una gerarchia di attributi composti; il componente IndirizzoVia di Indirizzo è ulteriormente composto da Numero, Via e NumeroAppartamento.

spettivamente. L'entità azienda c_1 ("c" come *company*) ha tre attributi: Nome, SedeCentrale e Presidente; i loro valori sono "Sunco Oil", "Houston" e "John Smith", rispettivamente.

Nel modello ER esistono diversi tipi di attributi: **semplice** in contrapposizione a **composto**; **a valore singolo** in contrapposizione a **multivalore**; **memorizzato** in contrapposizione a **derivato**. Nel seguito verranno definiti questi tipi di attributi e illustrato il loro uso tramite esempi. Quindi sarà introdotto il concetto di **valore nullo** (*null value*) per un attributo.

Attributi composti e attributi semplici (atomici). Gli **attributi composti** possono essere divisi in parti più piccole, che rappresentano più attributi di base con significati indipendenti. Ad esempio, l'attributo Indirizzo dell'entità impiegato mostrata in Figura 3.3 può essere diviso in IndirizzoVia, Città, Stato e CAP,⁴ con i valori "2311 Kirby", "Houston", "Texas" e "77001". Gli attributi che non sono divisibili sono detti **attributi semplici** o **atomici**. Gli attributi composti possono formare una gerarchia; ad esempio, IndirizzoVia può essere diviso in tre attributi semplici, Numero, Via e NumeroAppartamento, come mostrato in Figura 3.4. Il valore di un attributo composto è la concatenazione dei valori degli attributi semplici che lo costituiscono.

Gli attributi composti sono utili per modellare situazioni nelle quali un utente qualche volta fa riferimento all'attributo composto come a un tutt'uno, ma altre volte fa riferimento specificamente ai suoi componenti. Se all'attributo composto ci si riferisce solo come a un tutto, non c'è alcun bisogno di dividerlo negli attributi componenti. Ad esempio, se non è necessario riferirsi ai singoli componenti di un indirizzo (CAP, Via, ecc.), allora l'intero indirizzo è considerato un attributo semplice.

Attributi a valore singolo e attributi multivalore. La maggior parte degli attributi ha un valore singolo per una particolare entità; attributi di questo tipo sono detti a **valore singolo**. Ad esempio, Età è un attributo a valore singolo di persona. In certi casi un attributo può avere un in-

⁴ Per gli Stati Uniti si parla di *zip code*.

sieme di valori per la stessa entità – ad esempio, un attributo Colori per un'automobile, o un attributo Lauree per una persona. Le automobili con un solo colore hanno un valore singolo, mentre le automobili bicolori hanno due valori per Colori. Analogamente, una persona può non avere una laurea, un'altra può averne una e una terza può averne due o più; pertanto persone diverse possono avere *numeri diversi di valori* per l'attributo Lauree. Attributi di questo tipo sono detti **multivaleore**. Un attributo multivaleore può avere limiti inferiori e superiori per il numero di valori consentiti per ciascuna singola entità. Ad esempio, l'attributo Colori di un'automobile può avere tra uno e tre valori, se si suppone che un'automobile possa avere al più tre colori.

Attributi memorizzati e attributi derivati. In certi casi i valori di due o più attributi sono correlati – ad esempio, gli attributi Età e DataNascita di una persona. Per una particolare entità persona, il valore di Età può essere determinato a partire dalla data corrente e dal valore di DataNascita. L'attributo Età è perciò detto **attributo derivato**, e si dice che è **derivabile** dall'attributo DataNascita, che è detto **attributo memorizzato**. Alcuni valori di attributi possono essere derivati da *entità correlate*; ad esempio, un attributo NumeroDiImpiegati di un'entità dipartimento può essere derivato tramite il conteggio del numero di impiegati collegati a (che lavorano per) quel dipartimento.

Valori nulli. In certi casi una particolare entità può non avere un valore adatto per un attributo. Ad esempio, l'attributo NumeroAppartamento di un indirizzo ha senso solo per quegli indirizzi che sono relativi a edifici con appartamenti, e non ad altri tipi di abitazioni, come ad esempio case monofamiliari. Analogamente, un attributo Lauree ha senso solo per persone che effettivamente hanno una laurea. Per queste situazioni è stato definito un valore speciale detto **null**. Un indirizzo di una casa monofamiliare avrà null come valore del suo attributo NumeroAppartamento, e una persona senza lauree avrà null come valore di Lauree. Null può anche essere usato se non è noto il valore di un attributo per un'entità particolare – ad esempio, se non si conosce il numero di telefono di casa di "John Smith" in Figura 3.3. Il significato del primo tipo di null è *non applicabile*, mentre il significato del secondo è *sconosciuto*. La categoria dei null sconosciuti può essere ulteriormente suddivisa in due casi distinti: il primo caso si presenta quando è noto che il valore per quell'attributo esiste ma è *mancante* – ad esempio, se l'attributo Altezza di una persona è indicato come null; il secondo quando *non è noto* se il valore dell'attributo esista – ad esempio, se l'attributo TelefonoDiCasa di una persona è null.

Attributi complessi. Si noti che gli attributi composti e multivaleore possono essere annidati in modo arbitrario. È possibile rappresentare un annidamento arbitrario raggruppando i componenti di un attributo composto tra parentesi tonde () e separando i componenti con virgolette, nonché indicando gli attributi multivaleore tra parentesi graffe {}. Attributi di questo tipo sono detti **attributi complessi**. Ad esempio, se una persona può avere più di una residenza e ciascuna residenza può avere più telefoni, un attributo IndirizzoTelefono per un tipo di entità PERSONA può essere specificato come mostrato in Figura 3.5.

```
{IndirizzoTelefono{ {Telefono(Prefisso, NumeroTelefono)},  
Indirizzo{IndirizzoVia(Numeri, Via, NumeroAppartamento), Città, Stato, CAP) } }
```

Figura 3.5 Un attributo complesso IndirizzoTelefono con componenti multivaleore e composti.

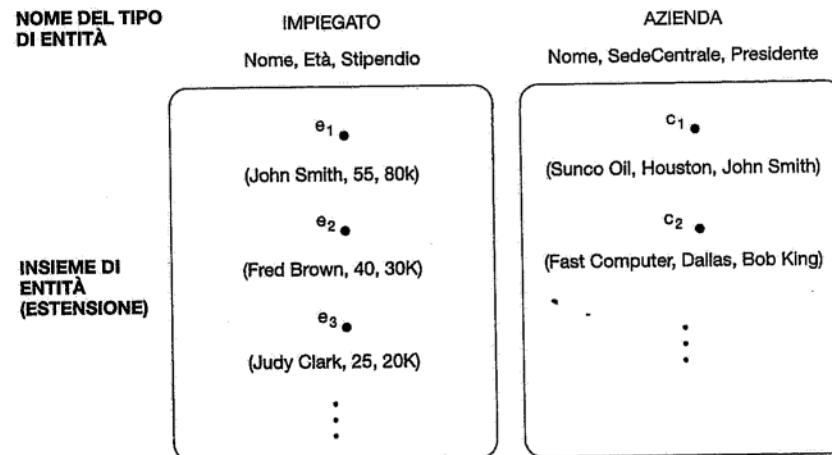


Figura 3.6 Due tipi di entità, detti IMPIEGATO e AZIENDA, e alcune delle entità membri della collezione di entità (o insieme di entità) di ciascun tipo.

3.3.2 Tipi di entità, insiemi di entità, chiavi e insiemi di valori

Tipi di entità e insiemi di entità. Una base di dati di solito contiene gruppi di entità simili. Ad esempio, un'azienda che occupa centinaia di impiegati può aver bisogno di memorizzare informazioni simili su ciascun impiegato. Queste entità impiegato condividono gli stessi attributi, ma ciascuna entità ha *i propri valori* per ciascun attributo. Un **tipo di entità** definisce una **collezione** (o *insieme*) di entità le quali possiedono gli stessi attributi. Ciascun tipo di entità nella base di dati è descritto dal suo nome e dai suoi attributi. In Figura 3.6 sono mostrati due tipi di entità, detti IMPIEGATO e AZIENDA, e un elenco di attributi per ciascuno di essi. Sono anche illustrate alcune entità di ciascun tipo, insieme con i valori dei loro attributi. La collezione di tutte le entità di un particolare tipo di entità nella base di dati in un istante di tempo qualsiasi è detta **insieme di entità**; di solito ci si riferisce all'insieme di entità usando lo stesso nome del tipo di entità. Ad esempio, IMPIEGATO si riferisce sia a un *tipo di entità* sia al corrente *insieme di tutte le entità impiegato* nella base di dati.

Un tipo di entità è rappresentato nei diagrammi ER⁵ (Figura 3.2) come un rettangolo con all'interno il nome del tipo di entità. I nomi degli attributi sono posti dentro degli ovali e

⁵ Si sta qui usando per i diagrammi ER una notazione simile a quella proposta originariamente (Chen 1976). Purtroppo sono in uso anche molte altre notazioni (si veda l'Appendice A e più avanti in questo capitolo).

sono uniti ai loro tipi di entità tramite linee rette. Gli attributi composti sono uniti ai loro attributi componenti tramite linee rette. Gli attributi multivaleure sono rappresentati con ovali doppi.

Un tipo di entità descrive lo **schema o intensione** per un *insieme di entità* che condividono la stessa struttura. La raccolta delle entità di un particolare tipo di entità è raggruppata in un insieme di entità, che è chiamato anche **estensione** del tipo di entità.

Attributi chiave di un tipo di entità. Un vincolo importante sulle entità di un tipo di entità è il **vincolo di chiave** o di **univocità** sugli attributi. Un tipo di entità di solito ha un attributo i cui valori sono distinti per ciascuna singola entità della collezione. Un attributo con queste caratteristiche è detto **attributo chiave**, e i suoi valori possono essere usati per identificare univocamente ciascuna entità. Ad esempio, l'attributo Nome è una chiave del tipo di entità AZIENDA in Figura 3.6, perché non è consentito che due aziende abbiano lo stesso nome. Per il tipo di entità PERSONA un tipico attributo chiave è NumeroPrevidenzaSociale. Talora molti attributi insieme formano una chiave, nel senso che la *combinazione* dei valori degli attributi deve essere distinta per ciascuna entità. Se un insieme di attributi possiede questa proprietà, è possibile definire un **attributo composto** che diventa un attributo chiave del tipo di entità. Si noti che una chiave composta deve essere *minimale*, cioè, tutti gli attributi componenti devono far parte dell'attributo composto per avere la proprietà di univocità.⁶ Nella notazione dei diagrammi ER ciascun attributo chiave ha il suo nome **sottolineato** dentro l'ovale, come illustrato in Figura 3.2.

La specificazione che un attributo è chiave di un tipo di entità significa che la precedente proprietà di univocità deve valere per *ogni estensione* del tipo di entità. Di conseguenza, è un vincolo che proibisce a qualsiasi coppia di entità di avere contemporaneamente lo stesso valore per l'attributo chiave. Non è la proprietà di una particolare estensione; piuttosto è un vincolo su *tutte le estensioni* del tipo di entità. Questo vincolo di chiave (come altri vincoli che verranno discussi in seguito) è derivato dai vincoli del mini-mondo che la base di dati rappresenta.

Alcuni tipi di entità hanno *più di un* attributo chiave. Ad esempio, sia l'attributo IDVeicolo (identificazione del veicolo) che l'attributo Registrazione del tipo di entità AUTOMOBILE (Figura 3.7) costituiscono una chiave a tutti gli effetti. L'attributo Registrazione è un esempio di chiave composta formata da due attributi componenti semplici, NumeroRegistrazione e Stato, nessuno dei quali è chiave, se preso singolarmente. Un tipo di entità può anche non avere *nessuna chiave*, nel qual caso è chiamato **tipo di entità debole** (si veda il Paragrafo 3.5).

Insiemi di valori (domini) degli attributi. Ciascun attributo semplice di un tipo di entità è associato a un **insieme di valori** (o **dominio** di valori), che specifica l'insieme di valori che possono essere assegnati a quell'attributo per ciascuna entità. In Figura 3.6, se la gamma di età consentite per gli impiegati va da 16 a 70, è possibile specificare l'insieme di valori del-

AUTOMOBILE
Registrazione(Numeroregistrazione, Stato), IDVeicolo, Marca, Modello, Anno, {Colore}

automobile₁ •
((ABC 123, TEXAS), TK629, Ford Mustang, cabriolet, 1998, {rosso, nero})

automobile₂ •
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4 porte, 1999, {blu})

automobile₃ •
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4 porte, 1995, {bianco, blu})

⋮

Figura 3.7 Il tipo di entità AUTOMOBILE, con due attributi chiave Registrazione e IDVeicolo. Gli attributi multivaleure sono indicati tra parentesi graffe {}. I componenti di un attributo composto sono indicati tra parentesi tonde () .

l'attributo Età di IMPIEGATO come l'insieme dei numeri interi tra 16 e 70. Analogamente, si può stabilire che l'insieme di valori per l'attributo Nome sia l'insieme delle stringhe di caratteri alfabetici separate da caratteri spazio e così via. Gli insiemi di valori non sono rappresentati nei diagrammi ER.

Matematicamente un attributo A di un tipo di entità E il cui insieme di valori è V può essere definito come una **funzione** da E all'insieme potenza⁷ P(V) di V:

$$A : E \rightarrow P(V)$$

Ci si riferirà qui al valore dell'attributo A per l'entità e come $A(e)$. La definizione precedente copre sia gli attributi a valore singolo sia quelli multivaleure, così come i null. Un valore null è rappresentato dall'insieme vuoto. Per attributi a valore singolo, $A(e)$ è vincolato a essere un insieme con un solo elemento (valore) per ciascuna entità e di E, mentre non c'è alcuna restrizione per attributi multivaleure.⁸ Per un attributo composto A l'insieme di valori V è il prodotto cartesiano di $P(V_1), P(V_2), \dots, P(V_n)$, dove V_1, V_2, \dots, V_n sono gli insiemi di valori degli attributi semplici componenti che formano A:

$$V = P(V_1) \times P(V_2) \times \dots \times P(V_n)$$

⁷ L'insieme potenza $P(V)$ di un insieme V è l'insieme di tutti i sottoinsiemi di V.

⁸ Nella letteratura in lingua inglese, un insieme con un solo elemento è detto singleton.

⁶ Gli attributi superflui non devono far parte di una chiave; però una superchiave può includere attributi superflui, come verrà spiegato nel Capitolo 7.

3.3.3 Progettazione concettuale iniziale della base di dati AZIENDA

Secondo i requisiti elencati nel Paragrafo 3.2, è ora possibile identificare per la base di dati AZIENDA quattro tipi di entità – uno per ciascun punto delle specifiche (Figura 3.8).

1. Un tipo di entità DIPARTIMENTO con attributi Nome, Numero, Sedi, Direttore e DataInizioDirettore. Sedi è l'unico attributo multivale. Si può specificare che sia Nome sia Numero sono attributi chiave (separati), perché si è stabilito che ciascuno dei due deve essere univoco.
2. Un tipo di entità PROGETTO con attributi Nome, Numero, Sede e DipartimentoControllante. Sia Nome sia Numero sono attributi chiave (separati).
3. Un tipo di entità IMPIEGATO con attributi Nome, SSN, Sesso, Indirizzo, Stipendio, DataNascita, Dipartimento e Supervisore. Sia Nome sia Indirizzo possono essere attributi composti; comunque ciò non è stato specificato nei requisiti. Occorre ritornare dagli utenti per vedere se qualcuno di loro farà riferimento ai singoli componenti di Nome – NomeDiBattesimo, InizialeIntermedia, Cognome – o di Indirizzo.
4. Un tipo di entità PERSONA_A_CARICO con attributi Impiegato, NomePersonaACarico, Sesso, DataNascita e Parentela (con l'impiegato).

Finora non è stato rappresentato il fatto che un impiegato può lavorare su più progetti, né il numero di ore a settimana che un impiegato lavora su ciascun progetto. Questa caratteristica è elencata come parte del requisito 3 del Paragrafo 3.2, e può essere rappresentata da un attributo composto multivale di IMPIEGATO detto LavoraSu con componenti semplici (Progetto, Ore). Alternativamente, può essere rappresentata come un attributo composto multivale di PROGETTO, detto Lavoratori, con componenti semplici (Impiegato, Ore). In Figura 3.8 viene scelta la prima possibilità che mostra ciascuno dei tipi di entità sopra descritti. L'attributo Nome di IMPIEGATO è presentato come attributo composto, presumibilmente dopo una consultazione con gli utenti.

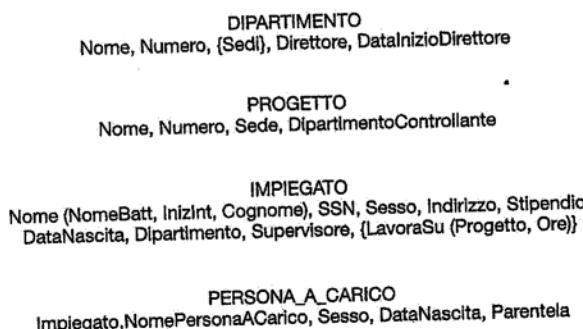


Figura 3.8 Progettazione preliminare di tipi di entità per la base di dati AZIENDA i cui requisiti sono descritti nel Paragrafo 3.2.

3.4 Associazioni, tipi di associazione, ruoli e vincoli strutturali

In Figura 3.8 ci sono diverse *associazioni implicite* tra i vari tipi di entità. Infatti, ogni volta che un attributo di un tipo di entità si riferisce a un altro tipo di entità sussiste una qualche associazione. Ad esempio, l'attributo Direttore di DIPARTIMENTO fa riferimento a un impiegato che dirige il dipartimento; l'attributo DipartimentoControllante di PROGETTO fa riferimento al dipartimento che controlla il progetto; l'attributo Supervisore di IMPIEGATO fa riferimento a un altro impiegato (quello che sovraintende a questo impiegato); l'attributo Dipartimento di IMPIEGATO si riferisce al dipartimento per il quale l'impiegato lavora, e così via. Nel modello ER questi riferimenti non dovrebbero essere rappresentati come attributi ma come associazioni (lo schema della base di dati AZIENDA sarà raffinato nel Paragrafo 3.6 per rappresentare esplicitamente le associazioni). Nel progetto iniziale dei tipi di entità le associazioni sono tipicamente colte come attributi. Man mano che il progetto viene raffinato, questi attributi sono convertiti in associazioni tra tipi di entità.

3.4.1 Tipi di associazione, insiemi di associazioni e istanze di associazione

Un tipo di associazione R tra n tipi di entità E_1, E_2, \dots, E_n definisce un insieme di legami – o un insieme di associazioni – tra entità di questi tipi. Come per i tipi di entità e gli insiemi di entità, un tipo di associazione e il corrispondente insieme di associazioni sono normalmente indicati con lo stesso nome R . Matematicamente, l'insieme di associazioni R è un insieme di istanze di associazione r_i , dove ciascun r_i lega n entità individuali (e_1, e_2, \dots, e_n) , e ciascuna entità e_j in r_i è un membro di un tipo di entità E_j , $1 \leq j \leq n$. Perciò un tipo di associazione è una relazione matematica su E_1, E_2, \dots, E_n , o alternativamente può essere definito come un sottoinsieme del prodotto cartesiano $E_1 \times E_2 \times \dots \times E_n$. Si dice che ciascun tipo di entità E_1, E_2, \dots, E_n partecipa al tipo di associazione R , e analogamente si dice che ciascuna delle singole entità e_1, e_2, \dots, e_n partecipa all'istanza di associazione $r_i = (e_1, e_2, \dots, e_n)$.

Informalmente, ciascuna istanza di associazione r_i in R è un legame tra entità, dove il legame include esattamente un'entità per ciascun tipo di entità partecipante. Ciascuna istanza di associazione r_i con queste caratteristiche rappresenta il fatto che le entità partecipanti a r_i sono correlate in qualche modo nella corrispondente situazione del mini-mondo. Ad esempio, si consideri un tipo di associazione LAVORA_PER tra i due tipi di entità IMPIEGATO e DIPARTIMENTO, che associa ciascun impiegato con il dipartimento per cui l'impiegato lavora. Ciascuna istanza di associazione nell'insieme di associazioni LAVORA_PER lega un'entità impiegato con un'entità dipartimento. Questo esempio è illustrato in Figura 3.9, dove ciascuna istanza di associazione r_i è mostrata connessa alle entità impiegato e dipartimento che partecipano a r_i . Nel mini-mondo rappresentato dalla Figura 3.9, gli impiegati e_1, e_3 ed e_6 lavorano per il dipartimento d_1 ; e_2 ed e_4 lavorano per d_2 ; infine e_5 ed e_7 lavorano per d_3 .

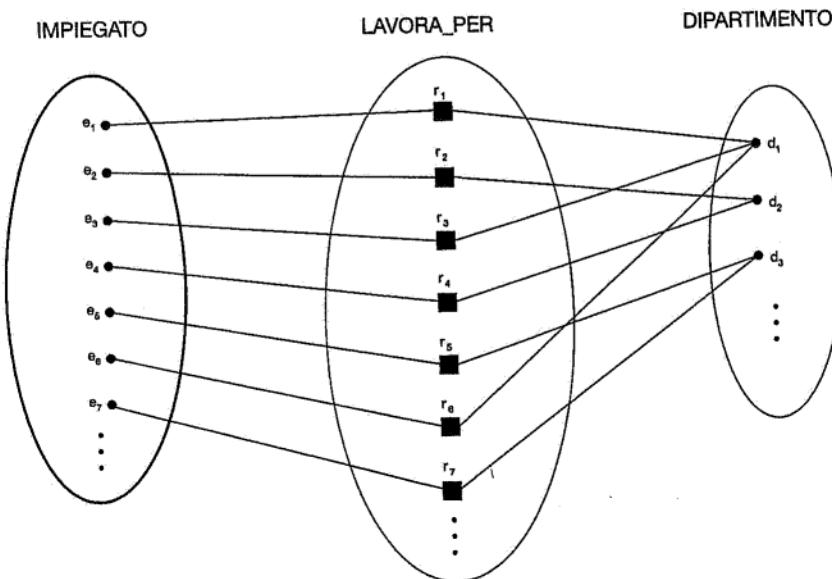


Figura 3.9 Alcune istanze dell'associazione LAVORA_PER tra IMPIEGATO e DIPARTIMENTO.

Nei diagrammi ER, i tipi di associazione sono rappresentati come rombi, collegati con linee rette ai rettangoli che rappresentano i tipi di entità partecipanti. Il nome dell'associazione è rappresentato dentro il relativo rombo (Figura 3.2).

3.4.2 Grado di un'associazione, nomi di ruolo e associazioni ricorsive

Grado di un tipo di associazione. Il grado di un tipo di associazione è il numero dei tipi di entità che vi partecipano. Perciò, l'associazione LAVORA_PER è di grado due. Un tipo di associazione di grado due è detto **binario**, e uno di grado tre è detto **ternario**.

Un esempio di associazione ternaria è FORNITURA, mostrata in Figura 3.10, dove ciascuna istanza di associazione r_i lega tre entità – un fornitore s ("s" come *supplier*) una parte p e un progetto j – ogni volta che s fornisce parte p al progetto j . In generale le associazioni possono essere di qualsiasi grado, ma quelle più comuni sono le associazioni binarie. Le associazioni di grado più alto sono generalmente più complesse delle associazioni binarie (si veda il Capitolo 4).

Associazioni come attributi. È talora conveniente pensare a un tipo di associazione in termini di attributi, come visto nel sottoparagrafo 3.3.3. Si consideri il tipo di associazione

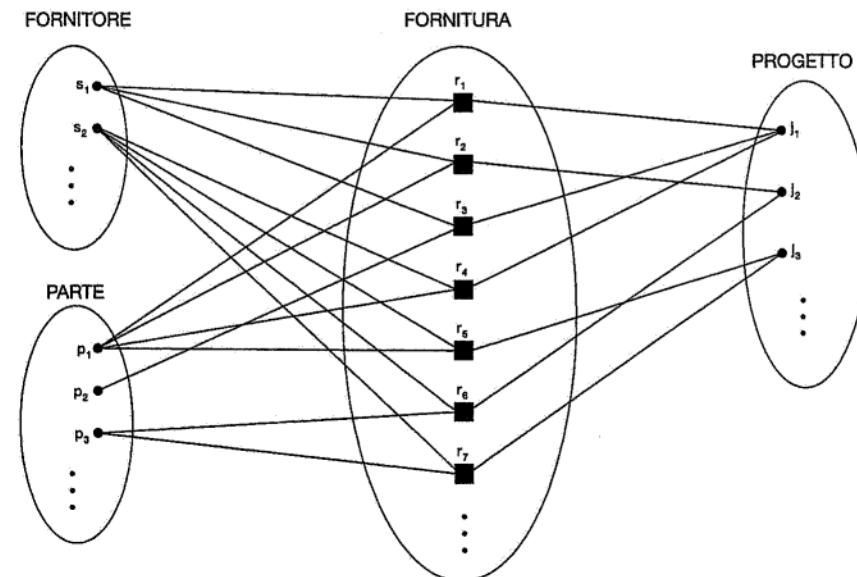


Figura 3.10 Alcune istanze di associazione per un'associazione ternaria FORNITURA.

LAVORA_PER di Figura 3.9. Si può pensare a un attributo, chiamiamolo Dipartimento, del tipo di entità IMPIEGATO il cui valore per ciascuna entità impiegato è (un riferimento a) l'*entità dipartimento* per cui lavora l'impiegato. Perciò, l'insieme di valori per questo attributo Dipartimento è l'*insieme di tutte le entità DIPARTIMENTO*. Ciò è quanto è stato fatto in Figura 3.8 quando si è specificato il progetto iniziale del tipo di entità IMPIEGATO per la base di dati AZIENDA. Comunque, quando si pensa a un'associazione binaria come a un attributo, esistono sempre due possibilità. In questo esempio l'alternativa è quella di pensare a un attributo multivaleore Impiegati per il tipo di entità DIPARTIMENTO, i cui valori per ciascuna entità dipartimento costituiscono l'*insieme delle entità impiegato* che lavorano per quel dipartimento. L'insieme di valori di questo attributo Impiegati è l'*insieme di entità IMPIEGATO*. Ciascuno di questi due attributi – Dipartimento di IMPIEGATO o Impiegati di DIPARTIMENTO – può rappresentare il tipo di associazione LAVORA_PER. Se sono rappresentati entrambi, essi sono vincolati ad essere fra di loro in direzioni opposte.⁹

⁹ L'idea di rappresentare tipi di associazione come attributi è usata in una classe di modelli di dati detti **modelli di dati funzionali**. Nelle basi di dati a oggetti, le associazioni possono essere rappresentate da attributi di riferimento, o in una sola direzione o in entrambe le direzioni, come opposti. Nelle basi di dati relazionali (si vedano i Capitoli 7 e 8), le chiavi esterne sono un tipo di attributo che attua riferimenti/collegamenti, e sono usate per rappresentare associazioni.

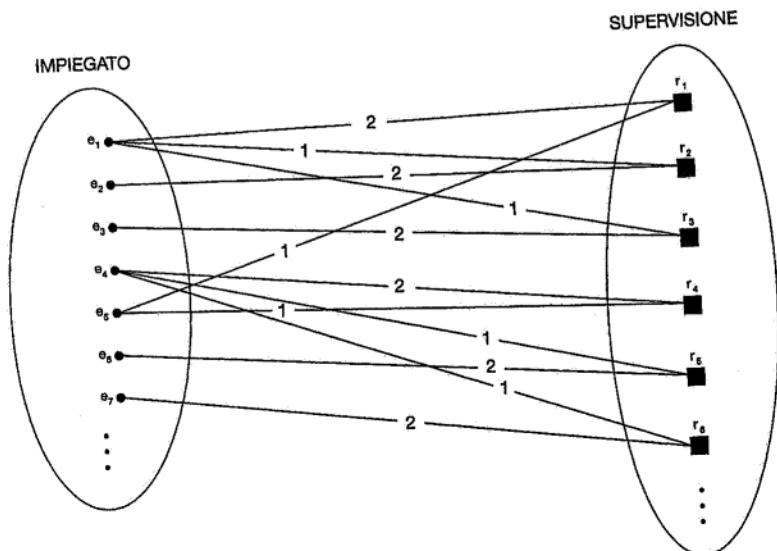


Figura 3.11 L'associazione ricorsiva SUPERVISIONE, in cui il tipo di entità IMPiegato svolge i due ruoli di supervisore (1) e di chi è sorvegliato (2).

Nomi di ruolo e associazioni ricorsive. Ogni tipo di entità che partecipa a un tipo di associazione recita in essa un **ruolo** particolare. Il **nome di ruolo** indica il ruolo che un'entità partecipante, facente parte del tipo di entità, recita in ciascuna istanza di associazione e aiuta a spiegare cosa indica quell'associazione. Ad esempio, nel tipo di associazione LAVORA_PER, IMPiegato sostiene il ruolo di *impiegato* o *lavoratore* e DIPARTIMENTO sostiene il ruolo di *dipartimento* o *datore di lavoro*.

I nomi di ruolo non sono tecnicamente necessari nei tipi di associazione in cui tutti i tipi di entità partecipanti sono distinti, dal momento che il nome di ciascun tipo di entità può essere usato come nome di ruolo. Però, in alcuni casi lo stesso tipo di entità partecipa più di una volta a un tipo di associazione con *ruoli diversi*. In questi casi il nome di ruolo diviene essenziale per distinguere il significato di ciascuna partecipazione. Associazioni di questo tipo sono dette **associazioni ricorsive**, e la Figura 3.11 ne mostra un esempio. Il tipo di associazione SUPERVISIONE collega un impiegato a un supervisore, dove sia l'entità impiegato sia quella supervisore sono membri dello stesso tipo di entità IMPiegato. Perciò il tipo di entità IMPiegato **partecipa due volte** a SUPERVISIONE: una volta nel ruolo di *supervisore* (o capo), e l'altra volta nel ruolo di *chi è sorvegliato* (o subalterno). Ciascuna istanza di associazione r_i in SUPERVISIONE associa due entità impiegato e_j ed e_k , delle quali una sostiene il ruolo di supervisore e l'altra il ruolo di chi è sorvegliato. In Figura 3.11 le linee contrassegnate con "1" indicano il ruolo di supervisore e quelle contrassegnate con "2" indicano il ruolo di chi è sorvegliato: quindi e_1 sorveglia e_2 ed e_3 ; e_4 sorveglia e_5 ed e_6 ; infine e_5 sorveglia e_1 ed e_4 .

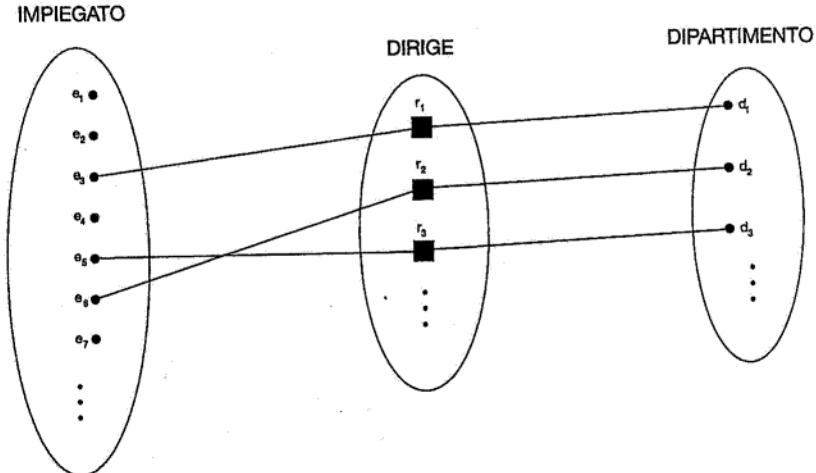


Figura 3.12 L'associazione 1:1 DIRIGE, con partecipazione parziale di IMPiegato e partecipazione totale di DIPARTIMENTO.

3.4.3 Vincoli sui tipi di associazione

I tipi di associazione hanno generalmente alcuni vincoli che limitano le possibili combinazioni di entità che possono far parte del corrispondente insieme di associazioni. Questi vincoli sono determinati dalla situazione del mini-mondo che le associazioni rappresentano. Ad esempio, in Figura 3.9, se l'azienda ha come regola quella che ciascun impiegato deve lavorare per uno e un solo dipartimento, allora si vorrebbe poter descrivere questo vincolo nello schema. È possibile distinguere due tipi principali di vincoli di associazione: *rapporto di cardinalità* e *partecipazione*.

Rapporti di cardinalità per associazioni binarie. Il **rapporto di cardinalità** per un'associazione binaria specifica il numero di istanze di associazione a cui può partecipare un'entità. Ad esempio, nel tipo di associazione binaria LAVORA_PER, DIPARTIMENTO:IMPiegato ha rapporto di cardinalità 1:N, nel senso che ciascun dipartimento può essere correlato con (cioè dà lavoro a) numerosi impiegati,¹⁰ ma un impiegato può essere correlato a (lavorare per) un solo dipartimento. I possibili rapporti di cardinalità per tipi di associazione binari sono 1:1, 1:N, N:1 e M:N.

Un esempio di associazione binaria 1:1 è DIRIGE (Figura 3.12), che correla un'entità dipartimento all'impiegato che dirige quel dipartimento. Ciò rappresenta i vincoli del mini-mon-

¹⁰ N sta per *qualsiasi numero* di entità correlate (zero o più).

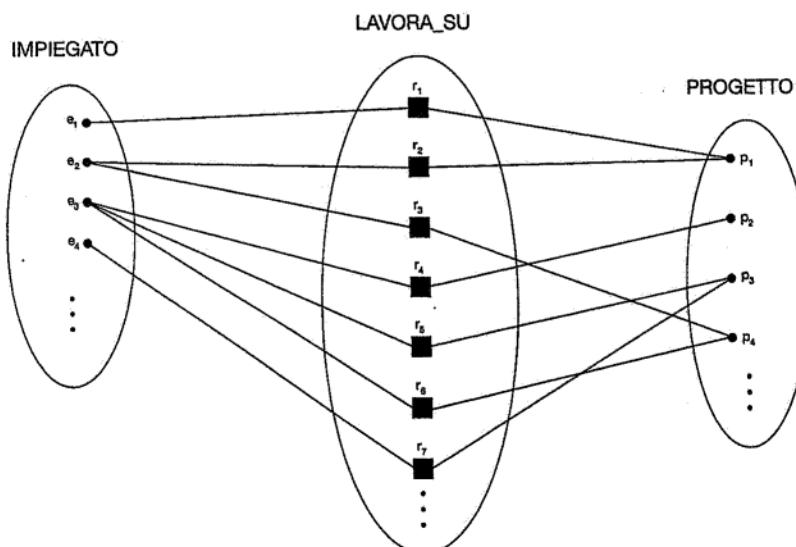


Figura 3.13 L'associazione M:N LAVORA_SU tra IMPIEGATO e PROGETTO.

do secondo i quali un impiegato può dirigere solo un dipartimento e un dipartimento ha un solo direttore. Il tipo di associazione LAVORA_SU (Figura 3.13) ha rapporto di cardinalità M:N, perché la regola del mini-mondo è che un impiegato può lavorare su molti progetti e che un progetto può avere molti impiegati.

I rapporti di cardinalità per associazioni binarie sono rappresentati nei diagrammi ER ponendo 1, M e N sui rombi come mostrato in Figura 3.2.

Vincoli di partecipazione e dipendenze di esistenza. Il vincolo di partecipazione specifica se l'esistenza di un'entità dipende dal suo essere correlata a un'altra entità attraverso un tipo di associazione. Ci sono due tipi di vincoli di partecipazione, totale e parziale. Se la politica di un'azienda stabilisce che *ogni* impiegato deve lavorare per un dipartimento, allora un'entità impiegato può esistere solo se essa partecipa a un'istanza di associazione LAVORA_PER (Figura 3.9). Perciò la partecipazione di IMPIEGATO a LAVORA_PER è detta **partecipazione totale**, intendendo che ogni entità nell'“insieme totale” di entità impiegato deve essere correlata a un'entità dipartimento attraverso LAVORA_PER. La partecipazione totale è detta anche **dipendenza di esistenza**. In Figura 3.12 non ci si aspetta che ogni impiegato diriga un dipartimento, perciò la partecipazione di IMPIEGATO al tipo di associazione DIRIGE è **parziale**, nel senso che *alcune* o “parte dell'insieme delle” entità impiegate sono correlate a un'entità dipartimento attraverso DIRIGE, ma ciò non vale necessariamente per tutte. Si farà qui riferimento al rapporto di cardinalità e ai vincoli di partecipazione, considerati nell'insieme, come ai **vincoli strutturali** di un tipo di associazione.

Nei diagrammi ER la partecipazione totale è rappresentata con una *linea doppia* che collega il tipo di entità partecipante all'associazione, mentre la partecipazione parziale è rappresentata con una *linea singola* (Figura 3.2).

3.4.4 Attributi di tipi di associazione

Anche i tipi di associazione possono avere attributi, analoghi a quelli dei tipi di entità. Ad esempio, per registrare il numero di ore settimanali lavorate da un impiegato su un particolare progetto, è possibile inserire un attributo Ore nel tipo di associazione LAVORA_SU di Figura 3.13. Un altro esempio è quello di inserire la data nella quale un direttore ha cominciato a dirigere un dipartimento attraverso un attributo DataInizio per il tipo di associazione DIRIGE di Figura 3.12.

Si noti che gli attributi di tipi di associazione 1:1 o 1:N possono essere trasferiti a uno dei tipi di entità partecipanti. Ad esempio, l'attributo DataInizio dell'associazione DIRIGE può essere un attributo sia di IMPIEGATO sia di DIPARTIMENTO – per quanto concettualmente esso appartenga a DIRIGE. Ciò avviene perché DIRIGE è un'associazione 1:1, e pertanto ogni entità dipartimento o impiegato partecipa *al più a una* istanza di associazione. Perciò, il valore dell'attributo DataInizio può essere determinato separatamente, o dall'entità partecipante dipartimento o dall'entità partecipante impiegato (direttore).

Per un tipo di associazione 1:N, un attributo dell'associazione può essere trasferito *solo* al tipo di entità al lato N dell'associazione. Ad esempio, in Figura 3.9, se l'associazione LAVORA_PER ha anche un attributo DataInizio che indica quando un impiegato ha cominciato a lavorare per un dipartimento, questo attributo può essere inserito come attributo di IMPIEGATO. Ciò perché ogni entità impiegato partecipa al più a un'istanza di associazione in LAVORA_PER. In entrambi i tipi di associazione 1:1 e 1:N la decisione riguardante dove dovrebbe essere posto un attributo di associazione – come attributo di un tipo di associazione o come attributo di un tipo di entità partecipante – è presa dal progettista dello schema su base soggettiva.

Per i tipi di associazione M:N alcuni attributi possono essere determinati dalla *combinazione di entità partecipanti* in un'istanza di associazione, non da una singola entità. Questi attributi *devono essere specificati come attributi dell'associazione*. Un esempio è l'attributo Ore dell'associazione M:N LAVORA_SU (Figura 3.13); il numero di ore che un impiegato lavora su un progetto è determinato da una combinazione impiegato-progetto e non separatamente da una o dall'altra entità.

3.5 Tipi di entità debole

I tipi di entità che non hanno propri attributi chiave sono detti **tipi di entità debole**. Per contrasto i **tipi di entità regolare** che hanno un attributo chiave sono talvolta detti **tipi di entità forte**. Le entità che appartengono a un tipo di entità debole vengono identificate tramite il loro collegamento con specifiche entità di un altro tipo, in combinazione con i valori di alcuni

dei loro attributi. Chiameremo quest'altro tipo di entità il **tipo di entità identificante o proprietario**,¹¹ e il tipo di associazione che lega un tipo di entità debole al suo proprietario l'**associazione identificante del tipo di entità debole**.¹² Un tipo di entità debole ha sempre un *vincolo di partecipazione totale* (dipendenza di esistenza) relativo alla sua associazione identificante, perché essa non può essere identificata senza un'entità proprietaria. Però non è che ogni dipendenza di esistenza abbia come risultato un tipo di entità debole. Ad esempio, un'entità di PATENTE non può esistere se non è correlata a un'entità di PERSONA, anche se essa ha una propria chiave (NumeroPatente) e di conseguenza non è un'entità debole.

Si consideri il tipo di entità PERSONA_A_CARICO, collegata a IMPiegato, che è usata per tenere traccia delle persone a carico di ciascun impiegato tramite un'associazione 1:N (Figura 3.2). Gli attributi di PERSONA_A_CARICO sono Nome (il nome di battesimo della persona a carico), DataNascita, Sesso e Parentela (con l'impiegato). Due persone a carico di *diversi impiegati* possono, per caso, avere gli stessi valori per gli attributi Nome, DataNascita, Sesso e Parentela, ma essere comunque entità distinte. Esse vengono riconosciute come entità distinte solo dopo aver determinato la *particolare entità impiegato* alla quale ciascuna è collegata. Si dice che ogni entità impiegato possiede le entità persona a carico ad essa collegate.

Normalmente un tipo di entità debole ha una **chiave parziale**, che è l'insieme di attributi che possono identificare univocamente le entità deboli *collegate alla stessa entità proprietaria*.¹³ Nel nostro esempio, se si suppone che due persone a carico dello stesso impiegato non possono mai avere lo stesso nome di battesimo, l'attributo Nome di PERSONA_A_CARICO è la chiave parziale. Nel caso peggiore un attributo composto da *tutti gli attributi dell'entità debole* costituirà la chiave parziale.

Nei diagrammi ER, sia un tipo di entità debole sia la sua associazione identificante vengono distinte circondando i relativi rettangoli e rombi con *linee doppie* (Figura 3.2). L'attributo chiave parziale è sottolineato con una linea tratteggiata o punteggiata.

I tipi di entità debole possono qualche volta essere rappresentati come attributi complessi (composti, multivalore). Nell'esempio precedente si sarebbe potuto specificare, per IMPIEGATO, un attributo multivaleure PersoneACarico, che sarebbe stato un attributo composto con attributi componenti Nome, DataNascita, Sesso e Parentela. La scelta di quale rappresentazione usare è fatta dal progettista della base di dati. Un criterio che può essere usato è quello di scegliere la rappresentazione con tipo di entità debole se ci sono molti attributi. Se l'entità debole partecipa indipendentemente a tipi di associazione diversi dal suo tipo di associazione identificante, allora *non* dovrebbe essere modellata come attributo complesso.

In generale, può essere definito un numero qualsiasi di livelli di tipi di entità debole; un tipo di entità proprietaria può essere esso stesso un tipo di entità debole. Inoltre, un tipo di entità debole può avere più di un tipo di entità identificante e un tipo di associazione identificante di grado maggiore di due, come si vedrà nel Capitolo 4.

3.6 Raffinamento della progettazione ER per la base di dati AZIENDA

È possibile ora raffinare la progettazione della base di dati di Figura 3.8 cambiando gli attributi che rappresentano associazioni con tipi di associazione. Il rapporto di cardinalità e il vincolo di partecipazione di ogni tipo di associazione sono determinati a partire dai requisiti elencati nel Paragrafo 3.2. Se qualche rapporto di cardinalità o qualche dipendenza non possono essere determinati dai requisiti, allora gli utenti devono essere intervistati per determinare tali vincoli strutturali.

Nel nostro esempio sono specificati i seguenti tipi di associazione.

1. DIRIGE, un tipo di associazione 1:1 tra IMPiegato e DIPARTIMENTO. La partecipazione di IMPiegato è parziale. La partecipazione di DIPARTIMENTO non è chiarita dai requisiti. Si intervistano allora gli utenti, che dicono che un dipartimento deve avere sempre un direttore, il che implica una partecipazione totale.¹⁴ A questo tipo di associazione è assegnato l'attributo DataInizio.
2. LAVORA_PER, un tipo di associazione 1:N tra DIPARTIMENTO e IMPiegato. Entrambe le partecipazioni sono totali.
3. CONTROLLA, un tipo di associazione 1:N tra DIPARTIMENTO e PROGETTO. La partecipazione di PROGETTO è totale, mentre, dopo aver consultato gli utenti, si stabilisce che quella di DIPARTIMENTO sia parziale.
4. SUPERVISIONE, un tipo di associazione 1:N tra IMPiegato (nel ruolo di supervisore) e IMPiegato (nel ruolo di chi è sorvegliato). Entrambe le partecipazioni sono considerate parziali, dopo che gli utenti hanno specificato che non ogni impiegato è un supervisore e non ogni impiegato ha un supervisore.
5. LAVORA_SU, considerata come un tipo di associazione M:N con attributo Ore, dopo che gli utenti hanno indicato che un progetto può avere molti impiegati che lavorano su di esso. Si stabilisce che entrambe le partecipazioni siano totali.
6. PERSONA_A_CARICO_DI, un tipo di associazione 1:N tra IMPiegato e PERSONA_A_CARICO, che è anche l'associazione identificante per il tipo di entità debole PERSONA_A_CARICO. La partecipazione di IMPiegato è parziale, mentre quella di PERSONA_A_CARICO è totale.

Dopo aver specificato i sei tipi di associazione visti sopra, si rimuovono dai tipi di entità in Figura 3.8 tutti gli attributi che sono stati trasformati in associazioni. Questi comprendono: Direttore e DataInizioDirettore per DIPARTIMENTO; DipartimentoControllante per PROGETTO; Dipartimento, Supervisore e LavoraSu per IMPiegato; Impiegato per PERSONA_A_CARICO. Quando si progetta lo schema concettuale di una base di dati è importante avere la minor ridondanza possibile. Se si desidera una certa ridondanza a livello di memorizzazione o a livello di vista d'utente, essa può essere introdotta più tardi, come visto nel Sottoparagrafo 1.6.1.

¹¹ Il tipo di entità identificante è talora anche detto tipo di entità genitore o tipo di entità dominante.

¹² Il tipo di entità debole è talora anche detto tipo di entità figlia o tipo di entità subordinata.

¹³ La chiave parziale è detta qualche volta discriminatore.

¹⁴ Le regole nel mini-mondo che determinano i vincoli sono talora dette *regole aziendali (business rules)*, dal momento che esse sono determinate dall'"azienda" od organizzazione che utilizzerà la base di dati.

la notazione del linguaggio universale di modellazione (UML: Universal Modeling Language), che è stata proposta come uno standard per la modellazione concettuale a oggetti.

Qui di seguito verrà invece descritta una notazione ER alternativa, per specificare vincoli strutturali sulle associazioni. Questa notazione comporta di associare una coppia di numeri interi (\min , \max) a ciascuna *partecipazione* di un tipo di entità E a un tipo di associazione R , dove $0 \leq \min \leq \max \geq 1$. I numeri indicano che, per ogni entità e in E , e deve partecipare ad almeno \min e al più a \max istanze di associazione in R , *qualsiasi sia l'istante di tempo considerato*. Usando questo metodo, $\min = 0$ indica partecipazione parziale mentre $\min > 0$ indica partecipazione totale.

In Figura 3.15 è rappresentato lo schema della base di dati AZIENDA usando la notazione (\min , \max).¹⁵ Di solito si usa o la notazione rapporto di cardinalità/linea singola/linea doppia o la notazione min/max. La notazione min/max è più precisa, e può essere comodamente usata per specificare vincoli strutturali per tipi di associazione di *qualsiasi grado*. Però non è sufficiente per specificare alcuni vincoli di chiave su associazioni di grado maggiore, come si vedrà nel Capitolo 4. La Figura mostra anche tutti i nomi di ruolo per lo schema della base di dati AZIENDA.

Sommario

In questo capitolo abbiamo presentato i concetti di modellazione di un modello di dati concettuale di alto livello, il modello Entità-Associazione (ER: Entity-Relationship). Abbiamo cominciato con il discutere il ruolo che un modello di dati di alto livello svolge nel processo di progettazione di una base di dati, e quindi abbiamo presentato un insieme di requisiti per la base di dati AZIENDA, che è uno degli esempi che verranno usati per tutto il libro. Abbiamo quindi definito i concetti di base del modello ER, cioè le entità e i loro attributi. Sono poi stati discussi i valori nulli e sono stati presentati i vari tipi di attributi, che possono essere annidati arbitrariamente per produrre attributi complessi:

- semplici o atomici,
- composti,
- multivale.

È stata anche esaminata brevemente la differenza fra attributi memorizzati e attributi derivati. Abbiamo in seguito studiato i concetti del modello ER a livello di schema o "intensionale":

- tipi di entità e loro corrispondenti insiemi di entità;
- attributi chiave di tipi di entità;

¹⁵ In alcune notazioni, e in particolare in quelle usate nella modellazione a oggetti, i (\min , \max) vengono posizionati sui lati opposti rispetto a quelli qui mostrati. Ad esempio, per l'associazione LAVORA_PER in Figura 3.15, (1,1) sarebbe dalla parte di DIPARTIMENTO e (4,N) sarebbe dalla parte di IMPIEGATO. Qui si è scelto di creare la nota-

- insiemi di valori (domini) di attributi;
- tipi di associazione e loro corrispondenti insiemi di associazioni;
- ruoli di partecipazione di tipi di entità in tipi di associazione.

Abbiamo presentato due metodi per specificare i vincoli strutturali su tipi di associazione. Il primo metodo distingue due tipi di vincoli strutturali:

- rapporti di cardinalità (1:1, 1:N, M:N, per associazioni binarie);
- vincoli di partecipazione (totale, parziale).

Il secondo specifica i numeri minimi e massimi (\min , \max) della partecipazione di ciascun tipo di entità a un tipo di associazione. Si sono studiati i tipi di entità debole e i concetti correlati di tipi di entità proprietaria, tipi di associazione identificante e attributi di chiave parziale.

Gli schemi Entità-Associazione possono essere rappresentati diagrammaticamente come diagrammi ER. Abbiamo mostrato come progettare uno schema ER per la base di dati AZIENDA cominciando col definire i tipi di entità e i loro attributi e quindi raffinando la progettazione per includere tipi di associazione. Abbiamo quindi rappresentato il diagramma ER per lo schema della base di dati AZIENDA.

I concetti di modellazione ER che abbiamo presentato fin qui – tipi di entità, tipi di associazione, attributi, chiavi e vincoli strutturali – possono modellare le tradizionali applicazioni di basi di dati per l'elaborazione di dati aziendali. Però molte applicazioni più nuove e più complesse – come la progettazione ingegneristica, i sistemi informativi a carattere medico o le telecomunicazioni – richiedono concetti aggiuntivi, se desideriamo modellarle con maggiore accuratezza. Noi esamineremo questi concetti di modellazione avanzata nel Capitolo 4, dove descriveremo anche in maggiore dettaglio i tipi di associazione ternari e di grado più elevato, e discuteremo le circostanze nelle quali queste associazioni sono distinte dalle associazioni binarie.

Questionario di verifica

- 3.1. Si discuta il ruolo di un modello di dati di alto livello nel processo di progettazione di una base di dati.
- 3.2. Si elencino i vari casi in cui risulta appropriato l'uso del valore null.
- 3.3. Si definiscano i seguenti termini: *entità*, *attributo*, *valore di attributo*, *istanza di associazione*, *attributo composto*, *attributo multivale*, *attributo derivato*, *attributo complesso*, *attributo chiave*, *insieme di valori (dominio)*.
- 3.4. Cos'è un tipo di entità? Cos'è un insieme di entità? Si spieghino le differenze esistenti tra entità, tipo di entità e insieme di entità.
- 3.5. Si spieghi la differenza esistente tra attributo e insieme di valori.
- 3.6. Cos'è un tipo di associazione? Si spieghino le differenze esistenti tra istanza di associazione, tipo di associazione e insieme di associazioni.
- 3.7. Cos'è un ruolo di partecipazione? Quando è necessario usare nomi di ruolo nella descrizione di tipi di associazione?

- 3.8. Si descrivano le due alternative esistenti per specificare vincoli strutturali su tipi di associazione. Quali sono i vantaggi e gli svantaggi di ciascuna?
- 3.9. Sotto quali condizioni un attributo di un tipo di associazione binaria può essere trasferito per diventare un attributo di uno dei tipi di entità partecipanti?
- 3.10. Quando pensiamo alle associazioni come attributi, quali sono gli insiemi di valori di questi attributi? Che classe di modelli di dati è basata su questo concetto?
- 3.11. Cosa si intende per tipo di associazione ricorsiva? Se ne forniscano alcuni esempi.
- 3.12. Quando viene usato nella modellazione dei dati il concetto di entità debole? Si definiscono i termini: *tipo di entità proprietaria*, *tipo di entità debole*, *tipo di associazione identificante e chiave parziale*.
- 3.13. L'associazione identificante di un tipo di entità debole può avere grado maggiore di due? Si forniscano esempi per illustrare la risposta.
- 3.14. Si discutano le convenzioni usate per rappresentare graficamente uno schema ER tramite un diagramma ER.
- 3.15. Si discutano le convenzioni di denominazione usate per i diagrammi di schema ER.

Esercizi

- 3.16. Si consideri il seguente insieme di requisiti per una base di dati università, usata per tener traccia delle trascrizioni dei libretti universitari degli studenti. Essa è simile ma non identica alla base di dati illustrata in Figura 1.2.
 - a. L'università tiene traccia, per ciascuno studente, del nome, numero di matricola, numero di previdenza sociale, indirizzo e numero di telefono corrente, indirizzo e numero di telefono permanente, data di nascita, sesso, anno di corso (primo, secondo, ...), dipartimento principale di afferenza, dipartimento in cui viene seguita una specializzazione complementare (se c'è), nonché del corso di studi (B.A., B.S., ..., Ph.D.). Alcune applicazioni dell'utente hanno necessità di fare riferimento a città, stato e CAP dell'indirizzo permanente dello studente, e al cognome dello studente. Sia il numero di previdenza sociale sia quello di matricola hanno valori unici per ciascuno studente.
 - b. Ogni dipartimento è descritto da un nome, codice di dipartimento, numero di ufficio, telefono di ufficio e dal relativo college. Sia il nome sia il codice hanno valori unici per ciascun dipartimento.
 - c. Ogni insegnamento ha un nome di insegnamento, una descrizione, un codice di insegnamento, un numero di ore per semestre, un livello e un dipartimento che lo offre. Il valore del codice dell'insegnamento è unico per ciascun insegnamento.
 - d. Ogni modulo ha un docente, un semestre, un anno, un insegnamento e un numero di modulo. Il numero di modulo distingue i moduli dello stesso insegnamento che sono tenuti durante lo stesso semestre/anno; i suoi valori sono 1, 2, 3, ..., fino al numero di moduli insegnati durante ciascun semestre.
 - e. Una votazione è caratterizzata da uno studente, un modulo, un voto in lettere e un voto numerico (0, 1, 2, 3 o 4).

- Si progetti uno schema ER per questa applicazione e si tracci un diagramma ER per questo schema. Si specifichino gli attributi chiave di ciascun tipo di entità e i vincoli strutturali su ciascun tipo di associazione. Si prenda nota di tutti i requisiti non specificati, e si facciano assunzioni appropriate per rendere complete le specifiche.
- 3.17. Gli attributi composti e multivalore possono essere annidati fino a un numero qualsiasi di livelli. Si supponga di voler progettare un attributo per un tipo di entità STUDENTE per tener traccia della precedente educazione universitaria. Un tale attributo avrà un valore per ciascun college precedentemente frequentato, e ciascun valore di questo tipo sarà composto da nome del college, date di inizio e di fine, titoli (titoli conseguiti in quel college, se ve ne sono) e trascrizioni del libretto universitario (insegnamenti completati in quel college, se ve ne sono). Ciascun valore relativo a un titolo contiene il nome del titolo e il mese e l'anno in cui il titolo è stato conseguito, e ciascun valore relativo alla trascrizione del libretto universitario contiene un nome di insegnamento, semestre, anno e voto. Si progetti un attributo per conservare queste informazioni. Si usino le convenzioni di Figura 3.5.
 - 3.18. Si illustri un progetto alternativo per l'attributo descritto nell'Esercizio 3.17, che usi solo tipi di entità (compresi tipi di entità debole, se necessario) e tipi di associazione.
 - 3.19. Si consideri il diagramma ER di Figura 3.16, che illustra uno schema semplificato per un sistema di prenotazioni aeree. Si estraggano dal diagramma ER i requisiti e i vincoli che hanno prodotto questo schema. Si cerchi di essere i più precisi possibile nella specificazione dei requisiti e dei vincoli.
 - 3.20. Nei Capitoli 1 e 2 sono stati discussi l'ambiente e gli utenti di una base di dati. È possibile considerare molti tipi di entità per descrivere un tale ambiente, come il DBMS, la base di dati memorizzata, il DBA e il catalogo/dizionario dei dati. Si cerchi di specificare tutti i tipi di entità che possono descrivere completamente un sistema di basi di dati e il suo ambiente; quindi si specifichino i tipi di associazione tra questi tipi di entità, e si tracci un diagramma ER per descrivere un tale ambiente generale di basi di dati.
 - 3.21. Si progetti uno schema ER per tener traccia delle informazioni sui voti raccolte nella Camera dei Rappresentanti statunitense durante la sessione congressuale biennale corrente. La base di dati deve memorizzare, per ciascuno STATO, il suo Nome (ad es. Texas, New York, California), e comprende la Regione dello stato (il cui dominio è {Northeast, Midwest, Southeast, Southwest, West}). Ciascun MEMBRODELCONGRESSO nella Camera dei Rappresentanti è descritto dal suo Nome, e comprende il Distretto rappresentato, la DataInizio in cui è stato eletto per la prima volta e il Partito politico a cui appartiene (il cui dominio è {Repubblicano, Democratico, Indipendente, Altro}). La base di dati memorizza ogni PROGETTODILEGGE (cioè legge proposta) e comprende il NomeProgettoLegge, la DataDiVotazione sul progetto di legge, informazioni se il progetto di legge è stato ApprovatoORespinto (il cui dominio è {Sì, NO}) e i Sostenitori (i membri del congresso che hanno sostenuto – cioè proposto – il progetto di legge). La base di dati tiene traccia di come ogni membro del congresso ha votato su ciascun progetto di legge (dominio dell'attributo relativo al voto è {Sì, No, Astenuto, Assente}). Si tracci un diagramma dello schema ER per l'applicazione sopra descritta. Si enunci in modo esplicito qualsiasi assunzione venga fatta.
 - 3.22. Si deve costruire una base di dati per memorizzare le squadre e le gare di una lega sportiva. Una squadra ha un certo numero di giocatori, dei quali non tutti partecipano a ogni

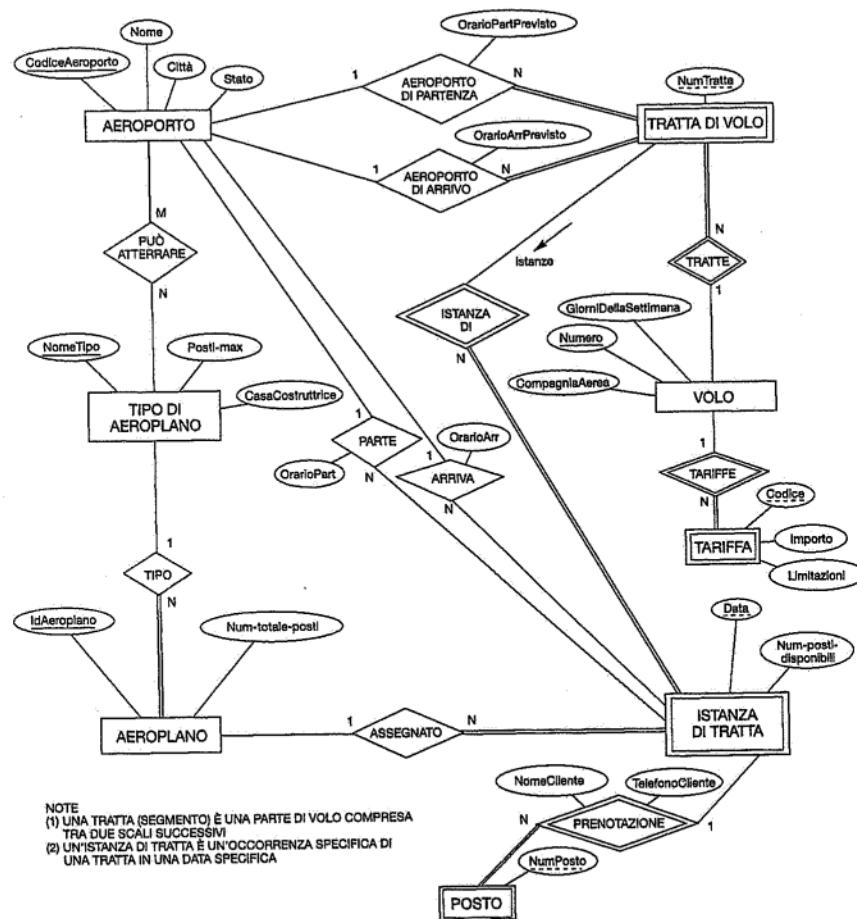


Figura 3.16 Un diagramma ER per una base di dati COMPAGNIA AEREA.

gara. Si desidera tener traccia dei giocatori che partecipano a ciascuna gara per ogni squadra, del ruolo che hanno ricoperto in quella gara e del risultato della gara. Si cerchi di progettare un diagramma dello schema ER per questa applicazione, enunciando esplicitamente ogni assunzione che viene fatta. Si scelga il proprio sport preferito (calcio, baseball, football americano ecc.).

- 3.23. Si consideri il diagramma ER illustrato in Figura 3.17 per una parte della base di dati BANCA. Ciascuna banca può avere più filiali, e ogni filiale può avere numerosi conti e prestiti.

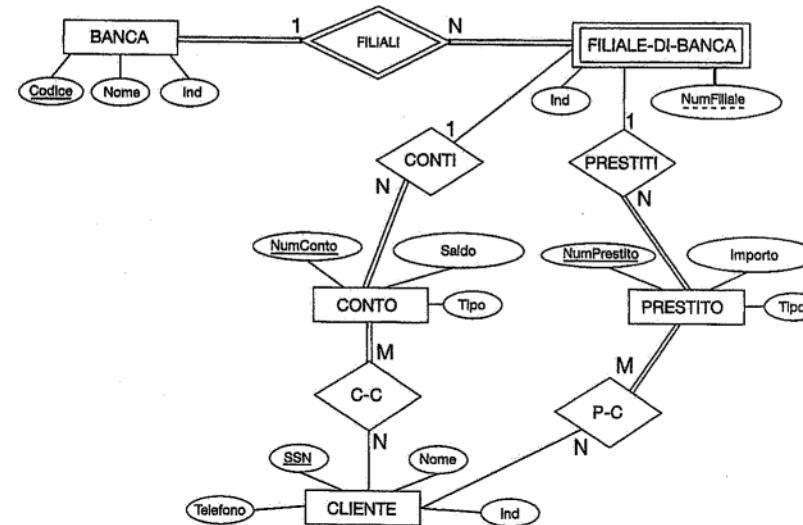


Figura 3.17 Un diagramma ER per una base di dati BANCA.

- Si elenchino i tipi di entità (non debole) nel diagramma ER.
 - C'è un tipo di entità debole? Se sì, si fornisca il suo nome, la sua chiave parziale e la sua associazione identificante.
 - Quali vincoli specificano in questo diagramma la chiave parziale e l'associazione identificante del tipo di entità debole?
 - Si elenchino i nomi di tutti i tipi di associazione e si specifichi il vincolo (min, max) per ciascuna partecipazione di un tipo di entità a un tipo di associazione. Si giustifichino le proprie scelte.
 - Si elenchino concisamente i requisiti d'utente che hanno portato a questo progetto dello schema ER.
 - Si supponga che ogni cliente debba avere almeno un conto ma possa godere al più di due prestiti contemporaneamente, e che una filiale della banca non possa gestire più di 1000 prestiti. Come si può mettere in luce ciò con i vincoli (min, max)?
- 3.24. Si consideri il diagramma ER di Figura 3.18. Si supponga che un impiegato possa lavorare al più in due dipartimenti, ma che possa anche non essere assegnato a nessun dipartimento. Si supponga che ciascun dipartimento debba avere almeno un numero di telefono e possa averne fino a tre. Si forniscono vincoli (min, max) a questo diagramma. Si enunci chiaramente ogni assunzione aggiuntiva che venga fatta. Sotto quali condizioni l'associazione HA_TELEFONO sarebbe ridondante nell'esempio sopra illustrato?
- 3.25. Si consideri il diagramma ER di Figura 3.19. Si supponga che un insegnamento possa usare o meno un libro di testo, ma che un testo per definizione sia un libro che è usato per un certo insegnamento. Un insegnamento non può usare più di cinque libri. I do-

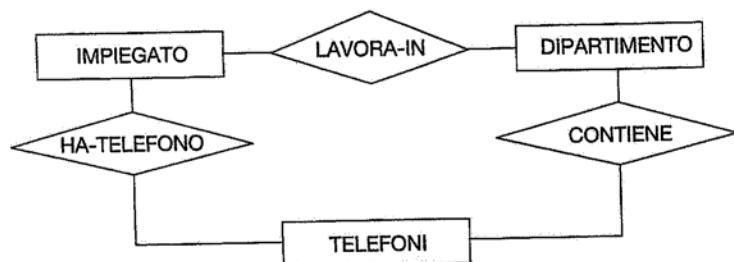


Figura 3.18 Un diagramma ER per una base di dati che tiene traccia dei telefoni di un'azienda e degli impiegati.

centi tengono da due a quattro insegnamenti. Si forniscano i vincoli (min, max) a questo diagramma. Si enunci chiaramente ogni assunzione aggiuntiva che venga fatta. Se si aggiunge l'associazione ADOTTA tra DOCENTE e TESTO, quali vincoli (min, max) si dovrebbero porre in essa? Perché?

- 3.26. In una base di dati UNIVERSITÀ si consideri un tipo di entità MODULO, che descrive le offerte di moduli degli insegnamenti. Gli attributi di MODULO sono: NumeroModulo, Semestre, Anno, CodiceInsegnamento, Docente, NumeroAula (in cui il modulo è insegnato), Edificio (in cui il modulo è insegnato), GiorniDellaSettimana (il dominio è costituito dalle combinazioni possibili di giorni della settimana in cui il modulo può essere offerto {LMV, LM, MG ecc.}) e Ore (il dominio è costituito da tutti i possibili intervalli di tempo durante i quali sono offerti i moduli {9-9.50 A.M., 10-10.50 A.M., 3.30-4.50 P.M., 5.30-6.20 P.M. ecc.}). Si supponga che il NumeroModulo sia unico per ciascun insegnamento all'interno di una particolare combinazione semestre/anno (cioè, se un insegnamento è offerto più volte durante un particolare semestre, le sue offerte di moduli sono numerate 1, 2, 3 ecc.). Ci sono diverse chiavi composte per MODULO, e alcuni attributi sono componenti di più di una chiave. Si individuino tre chiavi composte, e si illustri come esse possono essere rappresentate in un diagramma di schema ER.

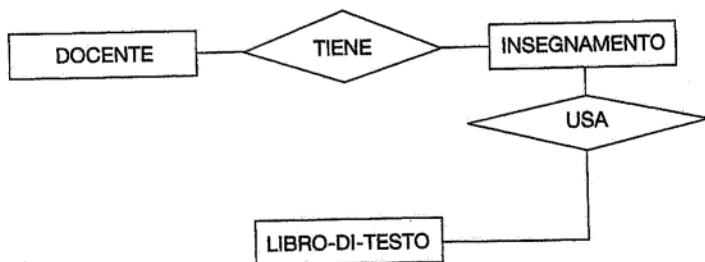


Figura 3.19 Un diagramma ER per una base di dati che memorizza i libri di testo usati per i vari insegnamenti.

Bibliografia selezionata

Il modello Entità-Associazione è stato introdotto da Chen (1976) e lavori collegati si trovano in Schmidt e Swenson (1975), Wiederhold e Elmasri (1979), e Senko (1975). Da allora sono state suggerite numerose modifiche al modello ER, alcune delle quali sono state introdotte nella nostra presentazione. I vincoli strutturali sulle associazioni sono studiati in Abrial (1974), Elmasri e Wiederhold (1980) e Lenzerini e Santucci (1983). Gli attributi multivalore e quelli composti sono inseriti nel modello ER in Elmasri e altri (1985). Anche se in questo capitolo non sono stati esaminati linguaggi per il modello Entità-Associazione e le sue estensioni, ci sono state molte proposte di linguaggi di questo tipo. Elmasri e Wiederhold (1981) propongono il linguaggio di interrogazione GORDAS per il modello ER. Un altro linguaggio di interrogazione ER è proposto da Markowitz e Raz (1983). Senko (1980) presenta un linguaggio di interrogazione per il modello DIAM di Senko. Un insieme formale di operazioni chiamato algebra ER è stato presentato da Parent e Spaccapietra (1985). Gogolla e Hohenstein (1991) presentano un altro linguaggio formale per il modello ER. Campbell e altri (1985) presentano un insieme di operazioni ER e dimostrano che esse sono complete dal punto di vista relazionale. Un congresso per la divulgazione dei risultati di ricerca relativi al modello ER è stato tenuto con regolarità a partire dal 1979. Il congresso, ora noto come Congresso Internazionale per la Modellazione Concettuale, è stato tenuto a Los Angeles (ER 1979, ER 1983, ER 1997), Washington (ER 1981), Chicago (ER 1985), Digione-Francia (ER 1986), New York (ER 1987), Roma (ER 1988), Toronto (ER 1989), Losanna-Svizzera (ER 1990), San Mateo-California (ER 1991), Karlsruhe-Germania (ER 1992), Arlington-Texas (ER 1993), Manchester-Inghilterra (ER 1994), Brisbane-Australia (ER 1995), Cottbus-Germania (ER 1996) e Singapore (ER 1998).



Capitolo 4

Modellazione Entità-Associazione estesa e modellazione a oggetti

I concetti di modellazione ER studiati nel Capitolo 3 sono sufficienti per rappresentare molti schemi di basi di dati per applicazioni “tradizionali”, che comprendono soprattutto applicazioni di elaborazione dati per il commercio e per l’industria. A partire dalla fine degli anni settanta, tuttavia, sono diventate comuni alcune applicazioni innovative – quali basi di dati per la progettazione e la produzione ingegneristica (CAD/CAM),¹ nonché per telecomunicazioni, immagini e grafica, multimedialità,² data mining (esplorazione di dati), data warehousing, sistemi di informazione geografica (GIS), e basi di dati di indici per il World Wide Web – con requisiti più complessi. Per rappresentare il più accuratamente e chiaramente possibile questi requisiti, i progettisti delle applicazioni di basi di dati devono utilizzare concetti aggiuntivi di *modellazione semantica dei dati*. In letteratura sono stati proposti diversi modelli semanticici di dati.

In questo capitolo descriveremo alcune caratteristiche che sono state proposte per modelli semanticici di dati, e mostreremo come il modello ER possa essere ampliato per comprendere questi concetti, arrivando al modello **ER-esteso** o modello **EER** (*Enhanced-ER*).³ Cominceremo nel Paragrafo 4.1 introducendo i concetti di *associazione classe/sottoclasse* ed *ereditarietà di tipo* nel modello ER. Quindi, nel Paragrafo 4.2, aggiungeremo i concetti di *specializzazione* e *generalizzazione*. Nel Paragrafo 4.3 esamineremo i *vincoli* sulla specializzazione/generalizzazione, e nel Paragrafo 4.4 mostreremo come possa essere modellato il costrutto UNIONE tramite l’inserimento del concetto di *categoria* nel modello EER. Nel Paragrafo 4.5 forniremo uno schema esemplificativo di basi di dati UNIVERSITÀ nel modello EER e riasumeremo i concetti del modello EER fornendo definizioni formali.

¹ Queste sigle stanno per computer-aided design/computer-aided manufacturing (progettazione assistita dal calcolatore/produzione assistita dal calcolatore).

² Queste basi di dati memorizzano dati multimediali, come immagini statiche, messaggi vocali e video.

³ EER è stato anche usato per *extended ER*.

Il modello di dati a oggetti comprende molti dei concetti proposti per i modelli di dati semantici. Le metodologie della modellazione a oggetti, come l'OMT (Object Modeling Technique: tecnica di modellazione a oggetti) e l'UML (Universal Modeling Language: linguaggio universale di modellazione), stanno diventando sempre più popolari nella progettazione e nell'ingegnerizzazione del software. Queste metodologie vanno al di là della progettazione di basi di dati, fino a specificare una progettazione dettagliata di moduli software e delle loro interazioni, tramite l'uso di vari tipi di diagrammi. I diagrammi delle classi,⁴ che costituiscono una parte importante di queste metodologie, sono per molti versi simili ai diagrammi EER, però in essi, oltre ad attributi e associazioni, sono pure specificate le *operazioni* sugli oggetti. Le operazioni possono essere usate per specificare i *requisiti funzionali* durante la progettazione di basi di dati, come visto nel Paragrafo 3.1 e illustrato in Figura 3.1. Presenteremo la notazione UML e i concetti per i diagrammi delle classi nel Paragrafo 4.6, e li confronteremo brevemente con la notazione e i concetti EER.

Nel Paragrafo 4.7 presenteremo alcuni dei problemi più complessi relativi alla modellazione di associazioni ternarie e di grado più elevato. Nel Paragrafo 4.8 esamineremo le astrazioni fondamentali usate come basi per molti modelli di dati semantici.

Per un'introduzione dettagliata alla modellazione concettuale, il Capitolo 4 dovrebbe essere considerato una continuazione del Capitolo 3. Tuttavia, se si desidera solo un'introduzione di base alla modellazione ER, questo capitolo può essere tralasciato. Alternativamente il lettore può scegliere di saltarne, in tutto o in parte, i paragrafi finali o (Paragrafi 4.3-4.8).

4.1 Sottoclassi, superclassi ed ereditarietà

Il modello EER comprende tutti i concetti di modellazione propri del modello ER presentati nel Capitolo 3. In più esso comprende i concetti di sottoclasse e di superclasse e i concetti collegati di **specializzazione** e di **generalizzazione** (si vedano i Paragrafi 4.2 e 4.3). Un altro concetto compreso nel modello EER è quello di **categoria** (si veda il Paragrafo 4.4), usato per rappresentare una collezione di oggetti che è l'*unione* di oggetti di diversi tipi di entità. Associato a questi concetti è l'importante meccanismo di **ereditarietà di attributi e associazioni**. Purtroppo non esiste alcuna terminologia standard per questi concetti, per cui si userà qui quella più comune. Una terminologia alternativa sarà fornita nelle note a piè di pagina. Verrà anche descritta una tecnica diagrammatica per rappresentare graficamente questi concetti quando essi si presentano in uno schema EER. I diagrammi di schema risultanti saranno indicati come **diagrammi ER-esteso** o **diagrammi EER**.

Il primo concetto del modello EER da prendere in considerazione è quello di **sottoclasse** di un tipo di entità. Come visto nel Capitolo 3, un tipo di entità è usato per rappresentare sia un *tipo di un'entità* sia l'*insieme di entità* o la *collezione di entità di quel tipo* che sussistono nella base di dati. Ad esempio, il tipo di entità IMPiegato descrive il tipo (cioè gli attributi e le associazioni) di ogni entità impiegato, e fa pure riferimento all'insieme corrente di entità

IMPIEGATO nella base di dati AZIENDA. In molti casi un tipo di entità ha numerose sotto-organizzazioni in gruppi delle sue entità che sono significative e che devono essere rappresentate esplicitamente per la loro importanza per l'applicazione di basi di dati. Ad esempio, le entità che sono membri del tipo di entità IMPiegato possono essere ulteriormente raggruppate in SEGRETERIO, INGEGNERE, DIRETTORE, TECNICO, IMPiegato_STIPENDIATO, IMPiegato_PA-GATO_A_ORE e così via. L'insieme di entità in ciascuna delle ultime organizzazioni in gruppi è un sottoinsieme delle entità che appartengono all'insieme di entità IMPiegato, nel senso che ogni entità che è un membro di una di queste sotto-organizzazioni è anche un impiegato. Ciascuna di queste sotto-organizzazioni rappresenta una **sottoclasse** del tipo di entità IMPiegato, e il tipo di entità IMPiegato è detto la **superclasse** per ognuna di queste sottoclassi.

L'associazione tra una superclasse e una qualsiasi delle sue sottoclassi rappresenta un'**associazione superclasse/sottoclasse** o semplicemente un'**associazione classe/sottoclasse**.⁵ Nell'esempio precedente, IMPiegato/SEGRETERIO e IMPiegato/TECNICO sono due associazioni classe/sottoclasse. Si noti che un'entità membro della sottoclasse rappresenta la *stessa entità del mondo reale* di un certo membro della superclasse; ad esempio, un'entità di SEGRETERIO 'Joan Logano' è pure l'IMPiegato 'Joan Logano'. Quindi il membro della sottoclasse è uguale all'entità della superclasse, ma in un *ruolo specifico* distinto. Quando si implementa un'associazione superclasse/sottoclasse nel sistema di basi di dati, tuttavia, si può rappresentare un membro della sottoclasse come un oggetto distinto della base di dati – ad esempio, un record distinto che è collegato tramite l'attributo chiave alla sua entità di superclasse (nel Paragrafo 9.2 verranno presentate varie opzioni per rappresentare associazioni superclasse/sottoclasse nelle basi di dati relazionali).

In una base di dati non può esistere un'entità che sia solamente membro di una sottoclasse; essa deve essere anche membro della superclasse. Tale entità può essere eventualmente inserita come membro di un numero qualsiasi di sottoclassi. Ad esempio, un impiegato stipendiato che sia anche un ingegnere appartiene alle due sottoclassi (del tipo di entità IMPiegato) INGEGNERE e IMPiegato_STIPENDIATO. Tuttavia non è necessario che ogni entità di una superclasse sia membro di una qualche sottoclasse.

Un importante concetto associato alle sottoclassi è quello di **ereditarietà di tipo**. Si ricorda che il *tipo* di un'entità è definito dagli attributi propri dell'entità e dai tipi di associazione a cui l'entità partecipa. Dato che un'entità nella sottoclasse rappresenta la stessa entità del mondo reale presente nella superclasse, essa dovrebbe disporre di valori per i suoi attributi specifici *così come* di valori per gli attributi posseduti in qualità di membro della superclasse. Un'entità che sia membro di una sottoclasse **eredita** tutti gli attributi dell'entità considerata come membro della superclasse. L'entità eredita anche tutte le associazioni alle quali partecipa la superclasse. Si noti che una sottoclasse, con i suoi attributi e le sue associazioni specifici (o locali), insieme a tutti gli attributi e le associazioni che eredita dalla superclasse, può essere considerata un *tipo di entità* a tutti gli effetti.⁶

⁵ Un'associazione classe/sottoclasse è spesso chiamata associazione È-UN (o È-UNO, È-UNA) per il modo in cui si fa riferimento al concetto. Infatti si dice "un SEGRETERIO È-UN IMPiegato", "un TECNICO È-UN IMPiegato" e così via.

⁶ In alcuni linguaggi di programmazione orientati a oggetti, una restrizione comune è che un'entità (o un oggetto) possa avere *un solo tipo*. Ciò è generalmente troppo restrittivo per la modellazione concettuale di basi di dati.

⁴ Una classe è per molti aspetti simile a un *tipo di entità*.

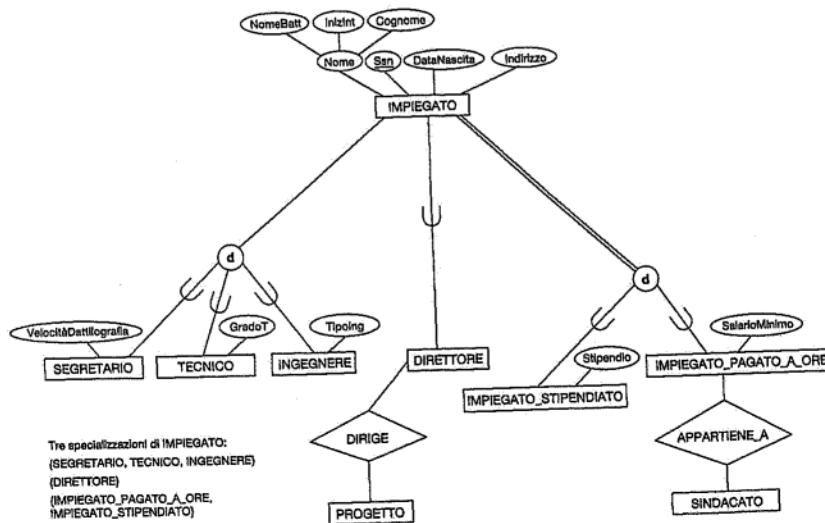


Figura 4.1 Notazione del diagramma EER per rappresentare specializzazione e sottoclassi.

4.2 Specializzazione e generalizzazione

Per **specializzazione** si intende il processo di definizione di un *insieme di sottoclassi* di un tipo di entità; questo tipo di entità è detto **superclasse** della specializzazione. L'insieme di sottoclassi che formano una specializzazione è definito sulla base di una certa caratteristica distintiva delle entità presenti nella superclasse. Ad esempio, l'insieme di sottoclassi {SEGRETARIO, INGENIERE, TECNICO} è una specializzazione della superclasse IMPIEGATO che fa una distinzione fra le entità di IMPIEGATO basata sul *tipo di lavoro* di ciascuna entità. È possibile avere diverse specializzazioni dello stesso tipo di entità, basate su diverse caratteristiche distintive. Ad esempio, un'altra specializzazione del tipo di entità IMPIEGATO può dar luogo all'insieme di sottoclassi {IMPIEGATO_STIPENDIATO, IMPIEGATO_PAGATO_A_ORE}: questa specializzazione fa una distinzione tra gli impiegati basata sulla *modalità di pagamento*.

In Figura 4.1 viene illustrato come si rappresenta diagrammaticamente una specializzazione in un **diagramma EER**. Le sottoclassi che definiscono una specializzazione sono unite tramite linee rette a un cerchio, che è a sua volta collegato alla superclasse. Il simbolo di sottosinsieme su ciascuna linea che collega una sottoclasse al cerchio indica la direzione dell'associazione superclasse/sottoclasse.⁷ Gli attributi che riguardano solo le entità di una parti-

olare sottoclasse – come VelocitàDattilografia di SEGRETARIO – sono uniti al rettangolo che rappresenta quella sottoclasse. Questi sono detti **attributi specifici** (o **attributi locali**) della sottoclasse. Analogamente, una sottoclasse può partecipare a tipi di associazione specifici, come ad esempio la sottoclasse IMPIEGATO_PAGATO_A_ORE che partecipa all'associazione APPARTIENE_A. Il simbolo d presente nei cerchi e la notazione diagrammatica aggiuntiva EER saranno spiegati fra breve.

In Figura 4.2 sono mostrate alcune istanze di entità che appartengono a sottoclassi della specializzazione {SEGRETARIO, INGENIERE, TECNICO}. Si noti ancora che un'entità che appartiene a una sottoclasse rappresenta *la stessa entità del mondo reale* dell'entità collegata ad essa presente nella superclasse IMPIEGATO, anche se la stessa entità è mostrata due volte; ad esempio in Figura 4.2 e_1 è mostrata sia in IMPIEGATO sia in SEGRETARIO. Come questa figura sug-

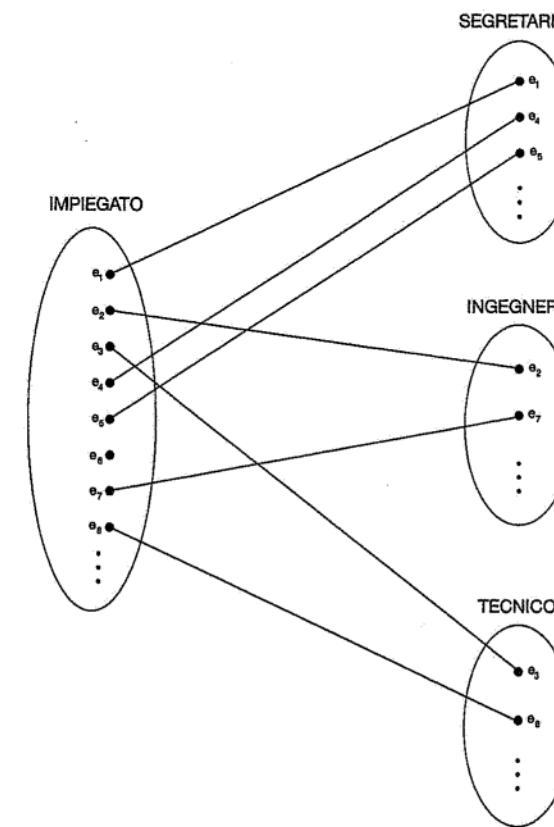


Figura 4.2 Alcune istanze della specializzazione di IMPIEGATO nell'insieme di sottoclassi {SEGRETARIO, INGENIERE, TECNICO}.

⁷ Esistono molte notazioni alternative per la specializzazione; qui si presenteranno la notazione UML nel Paragrafo 6 e zioni, le quali sono illustrate in Appendice A.

gerisce, un'associazione superclasse/sottoclasse come **IMPIEGATO/SEGRETARIO** assomiglia un po' a un'associazione 1:1 a livello di istanza (Figura 3.12). La differenza principale è che in un'associazione 1:1 sono messe in relazione due *entità distinte*, mentre in un'associazione superclasse/sottoclasse l'*entità* nella sottoclasse è la stessa entità del mondo reale dell'*entità* nella superclasse, ma essa svolge un *ruolo specializzato* – ad esempio, un **IMPIEGATO** specializzato nel ruolo di **SEGRETARIO**, o un **IMPIEGATO** specializzato nel ruolo di **TECNICO**.

Ci sono due ragioni fondamentali per inserire associazioni classe/sottoclasse e specializzazioni in un modello di dati. La prima è che certi attributi si possono applicare ad alcune ma non a tutte le entità della superclasse. Una sottoclasse è definita allo scopo di raggruppare le entità alle quali si applicano questi attributi. I membri della sottoclasse possono ancora condividere la maggioranza dei loro attributi con gli altri membri della superclasse. Ad esempio, la sottoclasse **SEGRETARIO** può avere un attributo **VelocitàMax** Dattilografia, mentre la sottoclasse **INGEGNERE** può avere un attributo **TipoIngegnere**, ma **SEGRETARIO** e **INGEGNERE** condividono i loro restanti attributi come membri del tipo di entità **IMPIEGATO**.

La seconda ragione per usare sottoclassi è che ad alcuni tipi di associazione possono partecipare solo entità che siano membri della sottoclasse. Ad esempio, se possono appartenere a un sindacato solo gli impiegati pagati a ore, è possibile rappresentare questo fatto creando la sottoclasse di **IMPIEGATO_IMPIEGATO_PAGATO_A_ORE** e collegando la sottoclasse a un tipo di entità **SINDACATO** attraverso il tipo di associazione **APPARTIENE_A**, come illustrato in Figura 4.1.

Riassumendo, il processo di specializzazione consente di fare quanto segue:

- definire un insieme di sottoclassi di un tipo di entità;
- stabilire attributi specifici aggiuntivi per ciascuna sottoclasse;
- stabilire tipi di associazione specifici aggiuntivi tra ciascuna sottoclasse e altri tipi di entità o altre sottoclassi.

Generalizzazione. Si può pensare a un *processo inverso* di astrazione, nel quale si sopprimono le differenze tra molti tipi di entità, si individuano le loro caratteristiche comuni e le si generalizzano in una singola **superclasse**, di cui i tipi di entità originari sono **sottoclassi** speciali. Ad esempio, si considerino i tipi di entità **AUTOMOBILE** e **CAMION** mostrati in Figura 4.3(a); essi possono essere generalizzati nel tipo di entità **VEICOLO**, come mostrato in Figura 4.3(b). Sia **AUTOMOBILE** sia **CAMION** sono ora sottoclassi della superclasse generalizzata **VEICOLO**. Si usa il termine **generalizzazione** per far riferimento al processo di definizione di un tipo di entità generalizzato a partire dai tipi di entità dati.

Si noti che il processo di generalizzazione può essere considerato come funzionalmente inverso al processo di specializzazione. Perciò in Figura 4.3 è possibile considerare **{AUTOMOBILE, CAMION}** come una specializzazione di **VEICOLO**, piuttosto che considerare **VEICOLO** come una generalizzazione di **AUTOMOBILE** e **CAMION**. Analogamente in Figura 4.1 si può considerare **IMPIEGATO** come una generalizzazione di **SEGRETARIO**, **TECNICO** e **INGEGNERE**. In alcune metodologie di progettazione si usa una notazione diagrammatica per distinguere tra generalizzazione e specializzazione. Una freccia che punta alla superclasse generalizzata rappresenta una generalizzazione, mentre frecce che puntano alle sottoclassi specializzate rappresentano una specializzazione. Qui *non* verrà adottata questa notazione, perché la decisione su quale processo sia il più appropriato in una particolare situazione è spesso soggettiva. L'Appendice A presenta alcune delle notazioni diagrammatiche alternative proposte per diagrammi di schema/diagrammi delle classi.

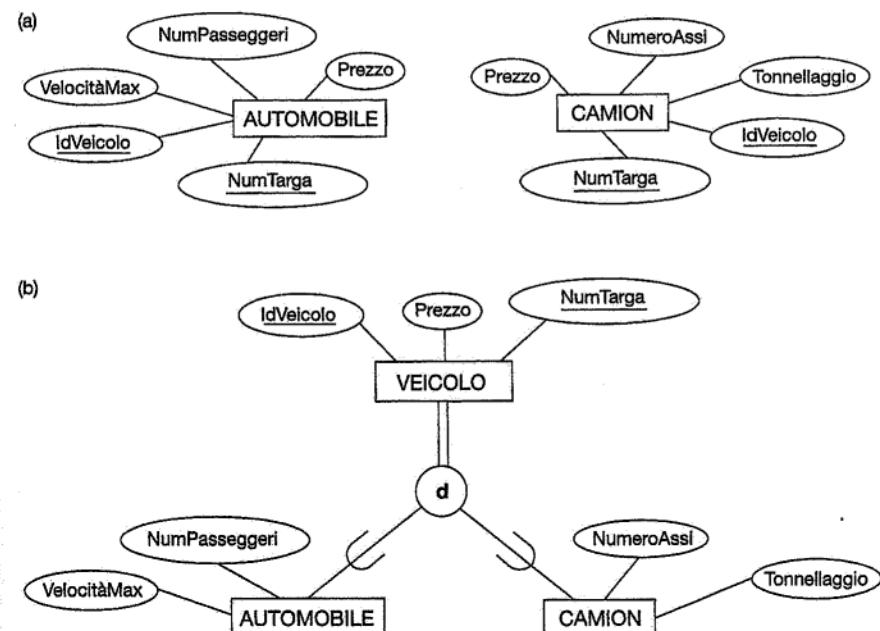


Figura 4.3 Esempi di generalizzazione. (a) Due tipi di entità **AUTOMOBILE** e **CAMION**. (b) Generalizzazione di **AUTOMOBILE** e **CAMION** in **VEICOLO**.

Finora sono stati introdotti i concetti di sottoclasse e di associazione superclasse/sottoclasse, così come i processi di specializzazione e generalizzazione. In generale una superclasse o sottoclasse rappresenta una collezione di entità dello stesso tipo e perciò descrive anche un *tipo di entità*; ecco perché le superclassi e le sottoclassi nei diagrammi EER sono rappresentate graficamente entro rettangoli (come i tipi di entità). Ora verranno esaminati in maggior dettaglio le proprietà delle specializzazioni e generalizzazioni.

4.3 Vincoli e caratteristiche di specializzazione e generalizzazione

In questo paragrafo verranno trattati in primo luogo i vincoli che si applicano a una singola specializzazione o a una singola generalizzazione; tuttavia, per brevità, la trattazione si riferirà solo alla *specializzazione*, anche se è valida sia per la specializzazione sia per la generalizzazione. Quindi saranno prese in esame le differenze tra i *reticolari* (*ereditarietà multipla*) e

le gerarchie (ereditarietà singola) di specializzazioni/generalizzazioni e introdotti particolari sulle differenze tra il processo di specializzazione e quello di generalizzazione durante la progettazione concettuale di uno schema di basi di dati.

Vincoli su specializzazione/generalizzazione. In generale, possono esserci molte specializzazioni definite sullo stesso tipo di entità (o superclasse), come illustrato in Figura 4.1. In casi di questo tipo, le entità possono appartenere a sottoclassi presenti in ciascuna specializzazione. Comunque una specializzazione può anche consistere solo in una *singola* sottoclasse, come la specializzazione {DIRETTORE} in Figura 4.1; in tal caso non si usa la notazione con il cerchio.

In alcune specializzazioni è possibile determinare esattamente le entità che diverranno membri di ciascuna sottoclasse ponendo una condizione sul valore di un certo attributo della superclasse. Tali sottoclassi sono dette **sottoclassi definite tramite un predicato** (o **definite tramite una condizione**). Ad esempio, se il tipo di entità IMPIEGATO ha un attributo TipoLavoro, come mostrato in Figura 4.4, si può specificare la condizione di appartenenza alla sottoclasse SEGRETARIO tramite il predicato (TipoLavoro = 'Segretario'), che chiameremo **predicato di definizione** della sottoclasse. Questa condizione è un **vincolo** che specifica che i membri della sottoclasse SEGRETARIO devono soddisfare il predicato e che tutte le entità del tipo di entità IMPIEGATO il cui valore di attributo per TipoLavoro è 'Segretario' devono appartenere alla sottoclasse. Qui si rappresenterà graficamente una sottoclasse definita tramite un predicato scrivendo la condizione del predicato vicino alla linea che congiunge la sottoclasse al cerchio di specializzazione.

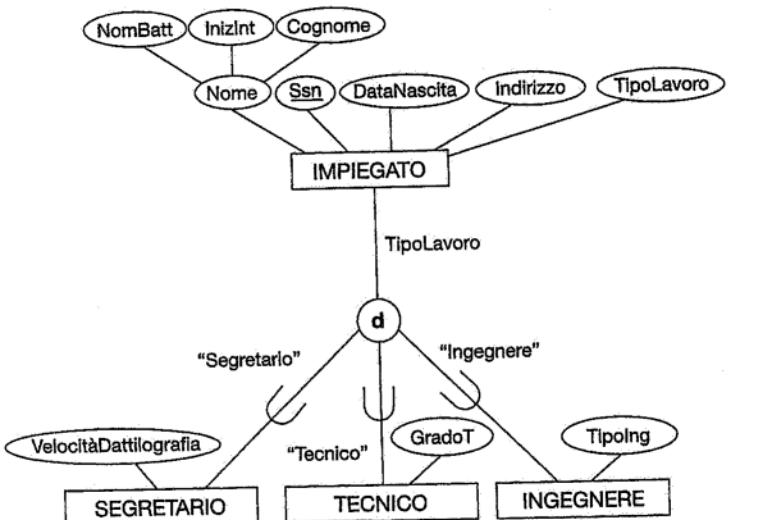


Figura 4.4 Una specializzazione definita tramite un attributo basata sull'attributo Tipolavoro di IMPIEGATO.

Se *tutte* le sottoclassi in una specializzazione hanno la condizione di appartenenza basata sullo *stesso* attributo della superclasse, la specializzazione è detta **specializzazione definita tramite un attributo**, e l'attributo è detto **attributo che definisce la specializzazione**.⁸ Graficamente essa verrà qui rappresentata ponendo il nome dell'attributo che definisce la specializzazione vicino all'arco che va dal cerchio alla superclasse, come mostrato in Figura 4.4.

Quando non si ha una condizione per determinare l'appartenenza a una sottoclasse, la sottoclasse è detta **definita dall'utente**. L'appartenenza a una sottoclasse di questo tipo è determinata dagli utenti della base di dati quando essi eseguono l'operazione di aggiungere un'entità alla sottoclasse; perciò l'appartenenza è *specificata individualmente per ciascuna entità dall'utente*, e non da una condizione che possa essere valutata automaticamente.

A una specializzazione possono essere applicati altri due vincoli. Il primo è il **vincolo di disgiunzione**, che specifica che le sottoclassi della specializzazione devono essere disgiunte. Ciò significa che un'entità può essere membro *al più di una* delle sottoclassi della specializzazione. Una specializzazione che sia definita tramite un attributo comporta il vincolo di disgiunzione se l'attributo usato per definire il predicato di appartenenza è a valore singolo. In Figura 4.4 è illustrato questo caso, dove la **d** nel cerchio sta per disgiunta. Si usa la notazione **d** anche per specificare il vincolo che le sottoclassi di una specializzazione definita dall'utente devono essere disgiunte, come illustrato dalla specializzazione {IMPIEGATO_PAGATO_A_ORE, IMPIEGATO_STIPENDIATO} in Figura 4.1. Se le sottoclassi non sono vincolate ad essere disgiunte, i loro insiemi di entità possono **sovraporsi**; cioè la stessa entità (del mondo reale) può essere membro di più di una sottoclasse della specializzazione. Questo caso, che è quello che si assume implicitamente, è rappresentato graficamente ponendo una **o** (per "overlap") nel cerchio, come mostrato in Figura 4.5.

Il secondo vincolo sulla specializzazione è detto **vincolo di completezza**, che può essere totale o parziale. Un vincolo di **specializzazione totale** stabilisce che *ogni* entità nella superclasse deve essere membro di una qualche sottoclasse della specializzazione. Ad esempio, se ogni IMPIEGATO deve essere o un IMPIEGATO_PAGATO_A_ORE o un IMPIEGATO_STIPENDIATO, allora la specializzazione {IMPIEGATO_PAGATO_A_ORE, IMPIEGATO_STIPENDIATO} di Figura 4.1 è una specializzazione totale di IMPIEGATO; ciò è mostrato nei diagrammi EER usando una linea doppia per collegare la superclasse al cerchio. Una linea singola è usata per rappresentare graficamente una **specializzazione parziale**, che consente a un'entità di non appartenere a nessuna sottoclasse. Ad esempio, se certe entità di IMPIEGATO non appartengono a nessuna delle sottoclassi {SEGRETARIO, INGEGNERE, TECNICO} delle Figure 4.1 e 4.4, allora quella specializzazione è parziale.⁹ Si noti che i vincoli di disgiunzione e di completezza sono *indipendenti*. Perciò si hanno i seguenti quattro possibili vincoli sulla specializzazione:

- disgiunta, totale;
- disgiunta, parziale;
- sovrapposta, totale;
- sovrapposta, parziale.

⁸ Un attributo siffatto è detto *discriminatore* nella terminologia UML.

⁹ La notazione consistente nell'usare linee singole/doppi è simile a quella per la partecipazione parziale/totale di un tipo di entità a un tipo di associazione, come visto nel Capitolo 3.

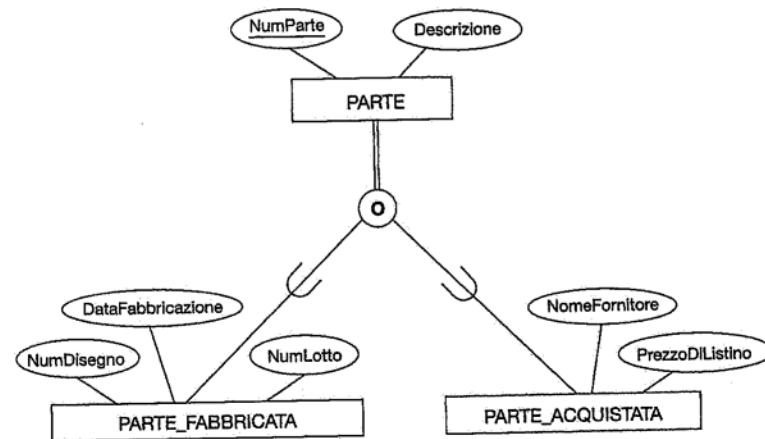


Figura 4.5 Notazione per specializzazione con sottoclassi che si sovrappongono (non disgiunte).

Naturalmente il vincolo corretto è determinato dal significato che assume ciascuna specializzazione nel mondo reale. Comunque, una superclasse che è stata individuata attraverso il processo di *generalizzazione* di solito è **totale**, perché la superclasse è *derivata dalle* sottoclassi e perciò contiene solo le entità che sono nelle sottoclassi.

Alla specializzazione (e alla generalizzazione) si applicano certe regole di inserimento e cancellazione, come conseguenza dei vincoli sopra specificati. Alcune di queste regole sono le seguenti:

- la cancellazione di un'entità da una superclasse implica che essa venga automaticamente cancellata da tutte le sottoclassi a cui appartiene;
- l'inserimento di un'entità in una superclasse implica che l'entità venga obbligatoriamente inserita in tutte le sottoclassi *definite tramite un predicato* (o *definite tramite un attributo*) per le quali l'entità soddisfa il predicato di definizione;
- l'inserimento di un'entità in una superclasse di una *specializzazione totale* implica che l'entità venga obbligatoriamente inserita in almeno una delle sottoclassi della specializzazione.

Il lettore è invitato a stendere un elenco completo di regole di inserimento e cancellazione per i vari tipi di specializzazione.

Gerarchie e reticolini di specializzazioni/generalizzazioni. Una sottoclasse può avere altre sottoclassi specificate su se stessa, a formare una gerarchia o un reticolo di specializzazioni. Ad esempio, in Figura 4.6 INGEGNERE è una sottoclasse di IMPiegato ed è anche una superclasse di DIRETTORE_INGEGNERE; ciò rappresenta il vincolo del mondo reale che ogni direttore del reparto ingegneria deve essere un ingegnere. Una **gerarchia di specializzazioni** presenta il vincolo che ogni sottoclasse partecipa come sottoclasse a una sola associazione classe/sotto-

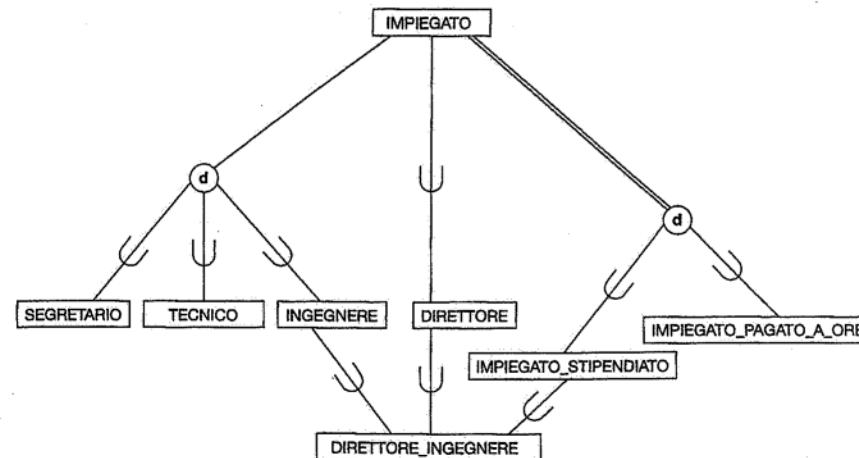


Figura 4.6 Un reticolo di specializzazioni con la sottoclasse condivisa **DIRETTORE_INGEGNERE**.

classe. Al contrario, per un **reticolo di specializzazioni** una sottoclasse può essere sottoclasse in *più di una* associazione classe/sottoclasse. Perciò la Figura 4.6 rappresenta un reticolo.

In Figura 4.7 è mostrato un altro reticolo di specializzazioni con più di un livello. Questo può essere parte di uno schema concettuale per una base di dati UNIVERSITÀ. Si noti che questa disposizione sarebbe una gerarchia se non ci fosse la sottoclasse ASSISTENTE_STUDENTI, che è sottoclasse in due distinte associazioni classe/sottoclasse. In Figura 4.7 tutte le entità persona rappresentate nella base di dati sono membri del tipo di entità PERSONA, che si specializza nelle sottoclassi {IMPIEGATO, EX_ALLIEVO, STUDENTE}. Questa specializzazione è sovrapposta; per esempio un ex allievo può essere anche un impiegato e può essere anche uno studente che prosegue gli studi per ottenere un titolo più avanzato. La sottoclasse STUDENTE è superclasse per la specializzazione {STUDENTE_LAUREATO, STUDENTE_NON_LAUREATO}, mentre IMPIEGATO è superclasse per la specializzazione {ASSISTENTE_STUDENTI, CORPO_DOCENTE, PERSONALE_DI_SUPPORTO}. Si noti che ASSISTENTE_STUDENTI è anche una sottoclasse di STUDENTE. Infine, ASSISTENTE_STUDENTI è superclasse per la specializzazione in {ASSISTENTE_DI_RICERCA, ASSISTENTE_DI_DIDATTICA}.

In un reticolo o gerarchia di specializzazioni una sottoclasse eredita gli attributi non solo della sua diretta superclasse, ma anche di tutte le superclassi che la precedono *fino ad arrivare alla radice* della gerarchia o del reticolo. Ad esempio, un'entità di STUDENTE_LAUREATO eredita tutti gli attributi che quell'entità possiede in quanto STUDENTE e in quanto PERSONA. Si noti che un'entità può trovarsi in più nodi foglia della gerarchia, dove un **nodo foglia** è una classe che non ha nessuna sottoclasse. Ad esempio, un membro di STUDENTE_LAUREATO può anche essere un membro di ASSISTENTE_DI_RICERCA.

Una sottoclasse con *più di una* superclasse è detta **sottoclasse condivisa**. Per esempio, se ogni DIRETTORE_INGEGNERE deve essere un INGEGNERE ma deve anche essere un IMPIEGA-

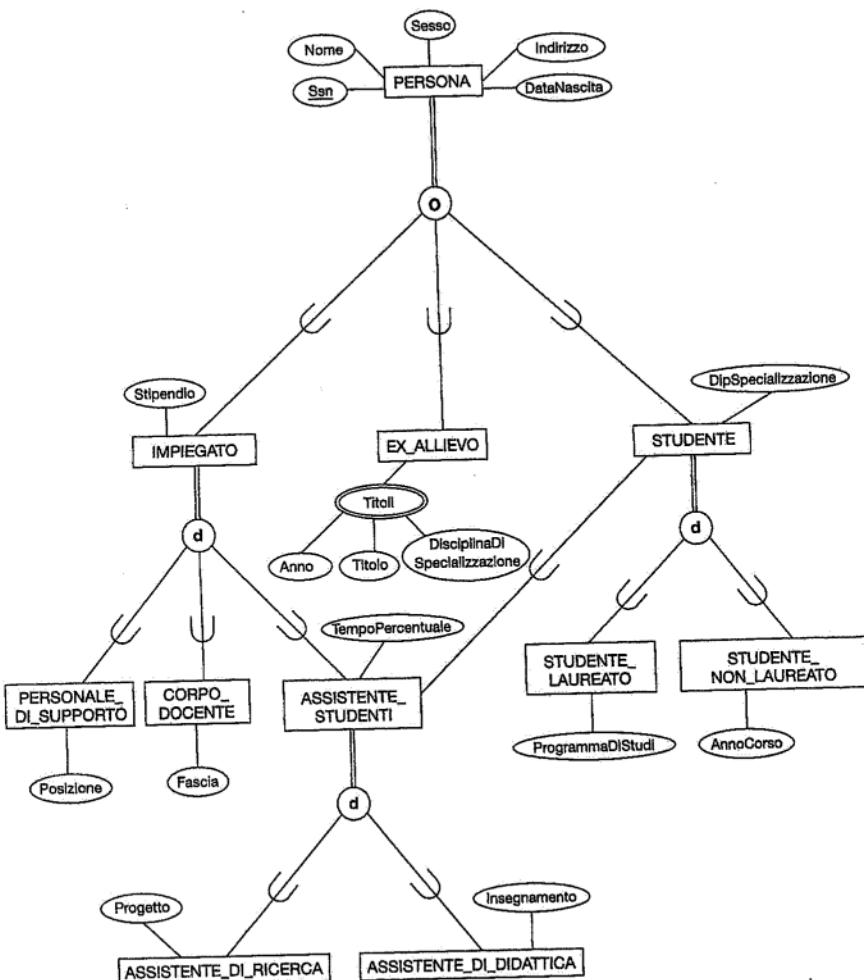


Figura 4.7 Un reticolo di specializzazioni (con ereditarietà multipla) per una base di dati UNIVERSITÀ.

TO_STIPENDIATO e un DIRETTORE, allora DIRETTORE_INGEGNERE dovrebbe essere una sottoclasse condivisa di tutte e tre le superclassi (Figura 4.6). Ciò porta al concetto noto come **ereditarietà multipla**, dal momento che la sottoclasse condivisa DIRETTORE_INGEGNERE eredita direttamente attributi e associazioni da più classi. Si noti che l'esistenza di almeno una sottoclasse condivisa porta a un reticolo (e perciò a *ereditarietà multipla*); se non esistessero sottoclassi condivise, si avrebbe una gerarchia piuttosto che un reticolo. Un'importante regola

la collegata all'ereditarietà multipla può essere illustrata tramite l'esempio della sottoclasse condivisa ASSISTENTE_STUDENTI di Figura 4.7, che eredita attributi sia da IMPIEGATO sia da STUDENTE. Qui sia IMPIEGATO sia STUDENTE ereditano *gli stessi attributi* da PERSONA. La regola stabilisce che se un attributo (o associazione) che ha origine nella stessa *superclasse* (PERSONA) è ereditato più di una volta passando attraverso percorsi diversi nel reticolo (IMPIEGATO e STUDENTE), allora esso deve essere inserito solo una volta nella sottoclasse condivisa (ASSISTENTE_STUDENTI). Perciò gli attributi di PERSONA sono ereditati *solo una volta* nella sottoclasse ASSISTENTE_STUDENTI di Figura 4.7.

A questo proposito è importante notare che altre accezioni di meccanismi di ereditarietà *non consentono* l'ereditarietà multipla (sottoclassi condivise). In un modello che non consente l'ereditarietà multipla è necessario creare sottoclassi aggiuntive per coprire tutte le combinazioni possibili di classi che possono avere una certa entità appartenente ad esse simultaneamente. Quindi ogni specializzazione *sovraposta* richiederebbe sottoclassi multiple aggiuntive. Ad esempio, nella specializzazione sovrapposta di PERSONA in {IMPIEGATO, EX_ALLIEVO, STUDENTE} (o {I, E, S} per brevità), per coprire tutti i tipi possibili di entità sarebbe necessario creare sette sottoclassi di PERSONA: I, E, S, I_E, I_S, E_S, I_E_S. Ovviamente ciò può portare a complessità aggiuntiva.

È anche importante notare che alcuni meccanismi di ereditarietà che consentono l'ereditarietà multipla non consentono a un'entità di avere più tipi, e perciò un'entità può essere membro *solo di una classe*.¹⁰ Anche in un modello di questo tipo è necessario creare sottoclassi condivise aggiuntive come nodi foglia per coprire tutte le combinazioni possibili di classi che possono avere una certa entità appartenente ad esse contemporaneamente. Perciò si avrà bisogno delle stesse sette sottoclassi di PERSONA.

Anche se qui si è parlato soltanto di specializzazione, concetti simili *si applicano egualmente* alla generalizzazione, come detto all'inizio di questo paragrafo. Perciò si può anche parlare di gerarchie di generalizzazioni e reticolli di generalizzazioni.

Uso della specializzazione e della generalizzazione nella modellazione concettuale di dati. Si vedano ora particolari sulle differenze tra il processo di specializzazione e quello di generalizzazione durante la progettazione concettuale di una base di dati. Nel processo di specializzazione si inizia tipicamente con un tipo di entità, per poi definire sottoclassi del tipo di entità attraverso specializzazioni successive; cioè si definiscono ripetutamente sotto-organizzazioni in gruppi sempre più specifiche del tipo di entità. Ad esempio, quando si progetta il reticolo di specializzazioni di Figura 4.7, è possibile dapprima specificare un tipo di entità PERSONA per una base di dati UNIVERSITÀ. Poi ci si rende conto del fatto che nella base di dati saranno rappresentati tre tipi di persone: impiegati dell'università, ex allievi e studenti. A questo scopo si crea la specializzazione {IMPIEGATO, EX_ALLIEVO, STUDENTE} e si sceglie il vincolo di sovrapposizione perché una persona può appartenere a più di una sottoclasse. Quindi si effettua un'ulteriore specializzazione di IMPIEGATO in {PERSONALE_DI_SUPPORTO, CORPO_DOCENTE, ASSISTENTE_STUDENTI} e di STUDENTE in {STUDENTE_LAUREATO, STUDENTE_NON_LAUREATO}. Infine, si opera una specializzazione di ASSISTENTE_STUDENTI in {AS-

¹⁰ In alcuni casi la classe è ulteriormente vincolata a essere un *nodo foglia* nella gerarchia o reticolo.

SISTENTE_DI_RICERCA, ASSISTENTE_DI_DIDATTICA}. Queste specializzazioni successive corrispondono a un **processo di raffinamento concettuale top-down** (dall'alto al basso) durante la progettazione dello schema concettuale. Fin qui si ha una gerarchia; quindi ci si rende conto del fatto che ASSISTENTE_STUDENTI è una sottoclasse condivisa, dal momento che è anche una sottoclasse di STUDENTE, arrivando così al reticolo.

È possibile arrivare alla stessa gerarchia o reticolo dalla direzione opposta. In tal caso il processo prevede generalizzazioni piuttosto che specializzazioni e corrisponde a una **sintesi concettuale bottom-up** (dal basso all'alto). In questo caso i progettisti possono prima scoprire tipi di entità come PERSONALE_DI_SUPPORTO, CORPO_DOCENTE, EX_ALLIEVO, STUDENTE_LAUREATO, STUDENTE_NON_LAUREATO, ASSISTENTE_DI_RICERCA, ASSISTENTE_DI_DIDATTICA e così via; poi generalizzano {STUDENTE_LAUREATO, STUDENTE_NON_LAUREATO} in ASSISTENTE; quindi {ASSISTENTE_DI_RICERCA, ASSISTENTE_DI_DIDATTICA} in ASSISTENTE_STUDENTI; poi {PERSONALE_DI_SUPPORTO, CORPO_DOCENTE, ASSISTENTE_STUDENTI} in IMPIEGATO; infine, {IMPIEGATO, EX_ALLIEVO, STUDENTE} in PERSONA.

In termini strutturali, le gerarchie o i reticolari che risultano da entrambi i processi possono essere identici; la sola differenza riguarda il modo o l'ordine in cui sono state specificate le superclassi e le sottoclassi dello schema. In pratica è probabile che non venga seguito fedelmente né il processo di generalizzazione né il processo di specializzazione, ma che ci si serva di una combinazione dei due processi. In questo caso vengono ripetutamente inserite, in una gerarchia o in una specializzazione, nuove classi, nel momento stesso in cui diventano evidenti a utenti e progettisti. Si noti che l'idea di rappresentare dati e conoscenza usando gerarchie e reticolari di superclassi/sottoclassi è abbastanza comune in sistemi basati sulla conoscenza e sistemi esperti, che combinano la tecnologia delle basi di dati con tecniche di intelligenza artificiale. Ad esempio, gli schemi di rappresentazione della conoscenza basati su frame assomigliano fortemente alle gerarchie di classi. Anche la specializzazione è comune in quelle metodologie di progettazione dell'ingegneria del software che sono basate sul paradigma orientato a oggetti.

4.4 Modellazione di tipi UNIONE attraverso l'uso di categorie

Tutte le associazioni superclasse/sottoclasse viste finora presentano *una sola superclasse*. Una sottoclasse condivisa come DIRETTORE_INGEGNERE nel reticolo di Figura 4.6 è sottoclasse in tre diverse associazioni superclasse/sottoclasse, dove ciascuna delle tre associazioni ha una superclasse *singola*. Non è infrequente, tuttavia, che sorga la necessità di modellare una singola associazione superclasse/sottoclasse con *più di una superclasse*, dove le superclassi rappresentano diversi tipi di entità. In questo caso la sottoclasse rappresenterà una collezione di oggetti che è l'**UNIONE** (o un suo sottoinsieme) di diversi tipi di entità; questa *sottoclasse* è detta **tipo unione o categoria**.¹¹

¹¹ L'uso qui fatto del termine *categoria* è basato sul modello ECR (Entity-Category-Relationship: Entità-Categoria-Associazione) (Elmasri e altri 1985).

Ad esempio, si supponga di avere tre tipi di entità: PERSONA, BANCA e AZIENDA. In una base di dati per l'immatricolazione di veicoli, il proprietario di un veicolo può essere una persona, una banca (che ha un'ipoteca sul veicolo), o un'azienda. Si ha quindi bisogno di creare una classe (collezione di entità) che comprenda entità di tutti e tre i tipi e che sostenga il ruolo di *proprietario del veicolo*. A questo scopo viene creata una categoria PROPRIETARIO che sia una *sottoclasse dell'UNIONE* dei tre insiemi di entità AZIENDA, BANCA e PERSONA. Le categorie in un diagramma EER sono rappresentate graficamente come mostrato in Figura 4.8. Le superclassi AZIENDA, BANCA e PERSONA sono collegate al cerchio contenente il simbolo U, che indica l'*operazione unione di insiemi*. Un arco con il simbolo di sottoinsieme collega il cerchio alla categoria (sottoclasse) PROPRIETARIO. Se si ha bisogno di un predicato di definizione, esso viene rappresentato graficamente vicino alla linea che parte dalla superclasse a cui si applica il predicato. In Figura 4.8 si hanno due categorie: PROPRIETARIO, che è una sottoclasse dell'unione di PERSONA, BANCA e AZIENDA; e VEICOLO_IMMATRICOLATO, che è una sottoclasse dell'unione di AUTOMOBILE e CAMION.

Una categoria ha due o più superclassi che possono rappresentare *tipi diversi di entità*, mentre le altre associazioni superclasse/sottoclasse hanno sempre una sola superclasse. È possibile confrontare una categoria, come PROPRIETARIO in Figura 4.8, con la sottoclasse condivisa DIRETTORE_INGEGNERE di Figura 4.6. Quest'ultima è una sottoclasse di *ciascuna delle* tre superclassi INGEGNERE, DIRETTORE e IMPIEGATO_STIPENDIATO, e pertanto un'entità che sia membro di DIRETTORE_INGEGNERE deve esserci in *tutte e tre*. Ciò rappresenta il vincolo che un direttore del reparto ingegneria deve essere un INGEGNERE, un DIRETTORE e un IMPIEGATO_STIPENDIATO; cioè DIRETTORE_INGEGNERE è un sottoinsieme dell'*intersezione* delle tre superclassi (insiemi di entità). Una categoria, invece, è un sottoinsieme dell'*unione* delle sue superclassi. Perciò un'entità che sia membro di PROPRIETARIO deve esistere in *una sola* delle superclassi. Ciò rappresenta il vincolo di Figura 4.8 che un PROPRIETARIO può essere un'AZIENDA, una BANCA o una PERSONA.

Nel caso di categorie l'ereditarietà di attributi opera in modo più selettivo. Ad esempio in Figura 4.8 una generica entità di PROPRIETARIO eredita gli attributi di AZIENDA, PERSONA o BANCA a seconda della superclasse a cui essa appartiene. Viceversa una sottoclasse condivisa come DIRETTORE_INGEGNERE (Figura 4.6) eredita *tutti* gli attributi delle sue superclassi IMPIEGATO_STIPENDIATO, INGEGNERE e DIRETTORE.

È interessante notare la differenza tra la categoria VEICOLO_IMMATRICOLATO (Figura 4.8) e la superclasse generalizzata VEICOLO (Figura 4.3b). In Figura 4.3(b) ogni automobile e ogni camion è un VEICOLO; d'altra parte in Figura 4.8 la categoria VEICOLO_IMMATRICOLATO comprende alcune automobili e alcuni camion, ma non necessariamente tutti (ad esempio, alcuni camion o automobili possono essere non immatricolati). In generale una specializzazione o generalizzazione come quella in Figura 4.3(b), se fosse *parziale*, non precluderebbe a VEICOLO di contenere altri tipi di entità, come ad esempio motociclette. Invece, una categoria come VEICOLO_IMMATRICOLATO di Figura 4.8 prevede che solo automobili e camion, ma non entità di altri tipi, possano essere membri di VEICOLO_IMMATRICOLATO.

Una categoria può essere **totale** o **parziale**. Ad esempio, TITOLARE_CONTO è una categoria parziale definita tramite un predicato in Figura 4.9(a), dove c_1 e c_2 sono le condizioni del predicato che specificano quali entità di AZIENDA e PERSONA, rispettivamente, sono membri di TITOLARE_CONTO. Viceversa la categoria PROPRIETÀ in Figura 4.9(b) è totale perché ogni edi-

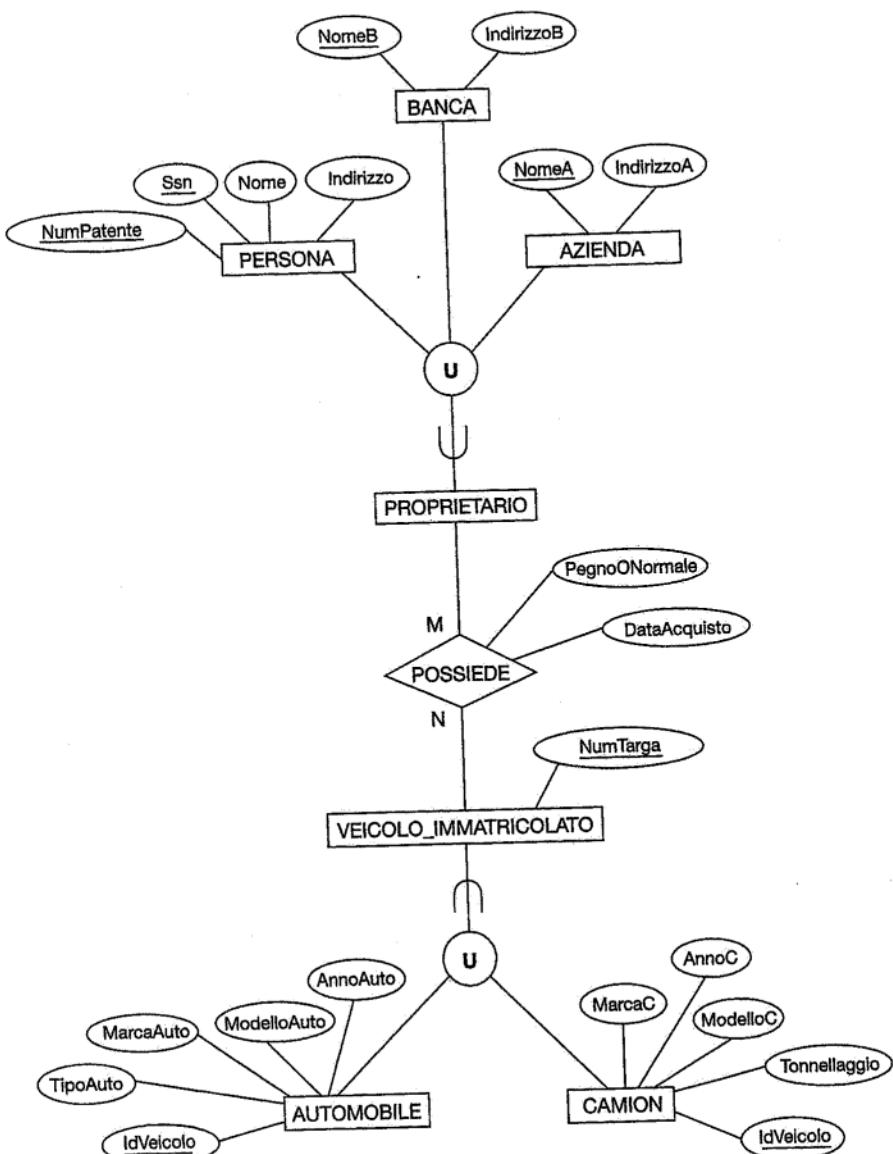


Figura 4.8 Un'illustrazione di come rappresentare l'UNIONE di due o più tipi di entità/classi usando la notazione di categoria. Sono rappresentate graficamente due categorie: PROPRIETARIO e VEICOLO_IMMATRICOLATO.

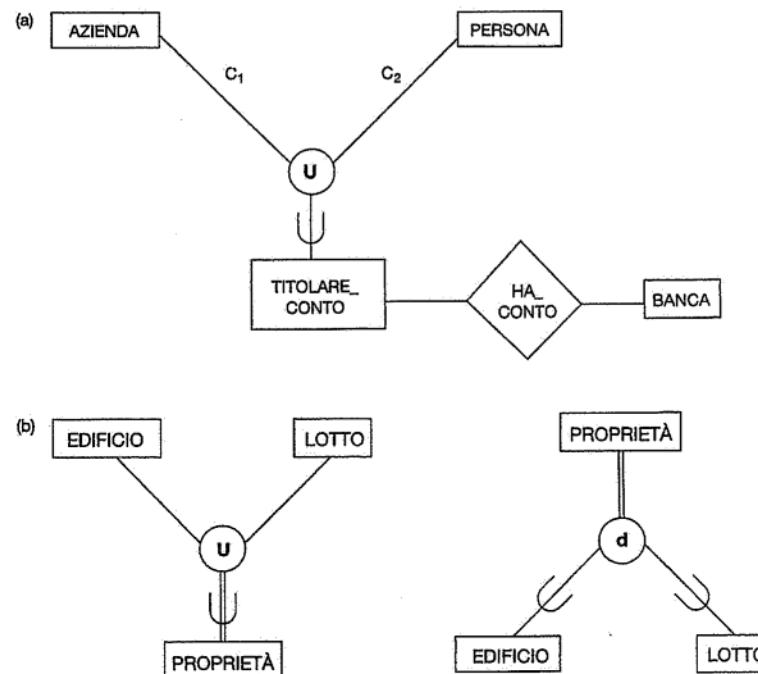


Figura 4.9 Categorie totali e parziali. (a) Categoria parziale TITOLARE_CONTO, sottoinsieme dell'unione di due tipi di entità AZIENDA e PERSONA. (b) La categoria totale PROPRIETÀ è una generalizzazione analoga.

ficio e lotto deve essere membro di PROPRIETÀ; ciò è rappresentato graficamente tramite una linea doppia che collega la categoria al cerchio. Le categorie parziali sono indicate da una linea singola che collega la categoria al cerchio, come in Figura 4.8 e 4.9(a).

Le superclassi di una categoria possono avere attributi chiave diversi, come dimostrato dalla categoria PROPRIETARIO di Figura 4.8, o viceversa avere lo stesso attributo chiave, come dimostrato dalla categoria VEICOLO_IMMATRICOLATO. Si noti che se una categoria è totale (non parziale), essa può essere alternativamente rappresentata come una specializzazione (o una generalizzazione), come illustrato in Figura 4.9(b). In questo caso la scelta su quale rappresentazione usare è soggettiva. Se le due classi rappresentano lo stesso tipo di entità e condividono numerosi attributi, compresi gli stessi attributi chiave, si preferisce la specializzazione/generalizzazione; altrimenti è più appropriata l'introduzione di una categoria (tipo unione).

4.5 Uno schema EER esemplificativo UNIVERSITÀ e definizioni formali per il modello EER

In questo paragrafo sarà dapprima fornito un esempio di schema di basi di dati nel modello EER, per illustrare l'uso dei vari concetti studiati qui e nel Capitolo 3. Successivamente saranno ripresi i concetti del modello EER e definiti formalmente nello stesso modo in cui nel Capitolo 3 erano stati definiti formalmente i concetti del modello ER di base.

La base di dati esemplificativa UNIVERSITÀ. Si consideri una base di dati UNIVERSITÀ che tenga traccia degli studenti e delle loro specializzazioni principali, delle trascrizioni dei libretti universitari e dell'iscrizione, nonché dell'offerta di insegnamenti dell'università. La base di dati memorizza anche i progetti di ricerca finanziati del corpo docente e degli studenti laureati. Questo schema è mostrato in Figura 4.10. Di seguito viene presentato uno studio dei requisiti che hanno portato ad esso.

Per ogni persona la base di dati tiene informazioni su nome [Nome], numero di previdenza sociale [Ssn], indirizzo [Indirizzo], sesso [Sesso] e data di nascita [DataN]. Sono state individuate due sottoclassi del tipo di entità PERSONA: CORPO_DOCENTE e STUDENTE. Attributi specifici di CORPO_DOCENTE sono fascia [Fascia] (assistente, professore associato, professore straordinario, ricercatore, visitatore ecc.), ufficio [UfficioCD], telefono ufficio [TelefonoCD] e stipendio [Stipendio], e tutti i membri del corpo docente sono collegati ai dipartimenti universitari ai quali afferiscono [FA_PARTE_DI] (un membro del corpo docente può afferire a molti dipartimenti, cosicché l'associazione è M:N). Un attributo specifico di STUDENTE è [AnnoCorso] (primo = 1, secondo = 2, ..., studente laureato = 5). Ogni studente è anche collegato al suo dipartimento di specializzazione principale e secondaria, se noti ([SPEC_PRINC] e [SPEC_SEC]), ai moduli di insegnamenti che sta attualmente frequentando ([ISCRITTO_A], e agli insegnamenti già completati [TRASCRIZIONE_LIBRETTO]). Ogni istanza di trascrizione del libretto comprende il voto che lo studente ha ricevuto [Voto] nel modulo dell'insegnamento.

STUDENTE_LAUREATO è una sottoclasse di STUDENTE, con il predicato di definizione AnnoCorso = 5. Per ogni studente laureato viene conservato un elenco dei titoli precedentemente conseguiti in un attributo composto e multivaleore [Titoli]. Inoltre, lo studente laureato viene collegato a un tutor facente parte del corpo docente [TUTOR] e a una commissione di tesi [COMMISSIONE], se ne esiste una.

Un dipartimento universitario ha gli attributi nome [NomeD], telefono [TelefonoD], e numero di ufficio [Ufficio] ed è collegato al membro del corpo docente che ne è il direttore [DIRIGE] e al college di cui fa parte [CD]. Ogni college ha come attributi nome del college [NomeC], numero di ufficio [UfficioC], e il nome del suo preside [Preside].

Un insegnamento ha come attributi codice insegnamento [I#], nome insegnamento [NomeI] e descrizione insegnamento [DescrI]. Sono offerti molti moduli per ciascun insegnamento, e ogni modulo ha per attributi il codice del modulo [Mod#] e l'anno e il trimestre nei quali il modulo è stato offerto ([Anno] e [Trim]).¹² I codici del modulo identificano univocamente ciascun modulo. I moduli che vengono offerti durante il trimestre in corso sono in una sottoclasse MODULO_CORRENTE di MODULO, con il predicato di definizione Trim = TrimCorrente e Anno = AnnoCorrente. Ciascun modulo è collegato al docente che lo ha insegnato o che lo sta insegnando ([INSEGNA], se quel docente è nella base di dati).

¹² Si assume che in questa università venga usato il sistema dei *trimestri* anziché quello dei *semestri*.

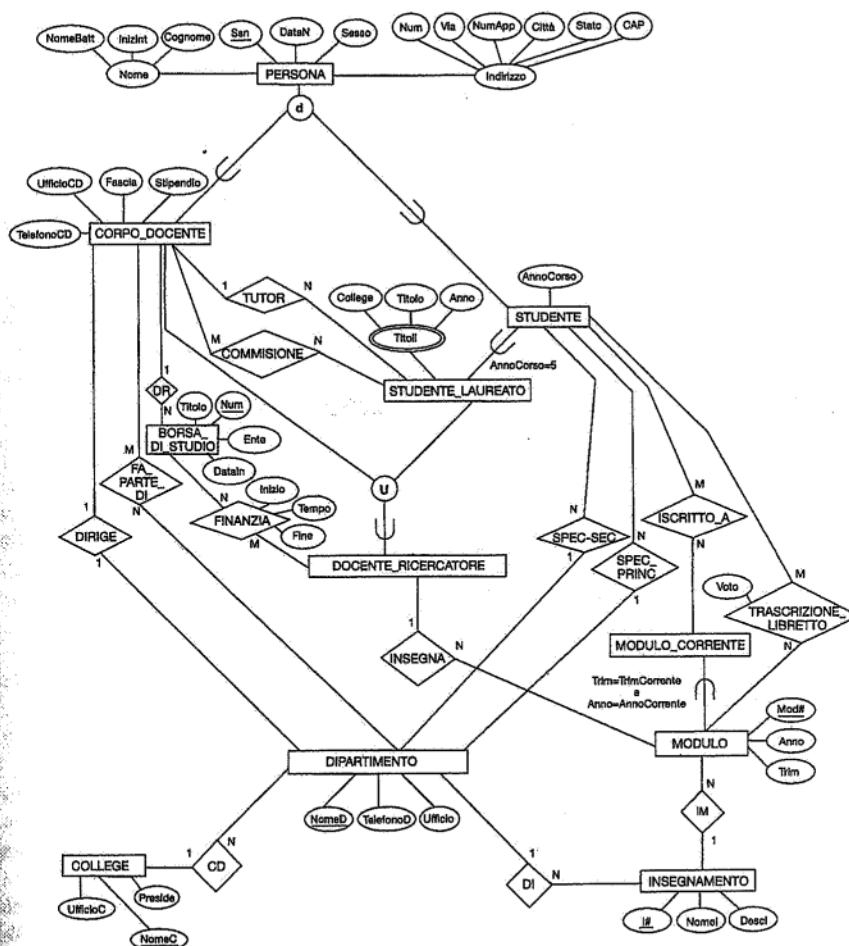


Figura 4.10 Uno schema concettuale EER per una base di dati UNIVERSITÀ.

te ciascun modulo. I moduli che vengono offerti durante il trimestre in corso sono in una sottoclasse MODULO_CORRENTE di MODULO, con il predicato di definizione Trim = TrimCorrente e Anno = AnnoCorrente. Ciascun modulo è collegato al docente che lo ha insegnato o che lo sta insegnando ([INSEGNA], se quel docente è nella base di dati).

La categoria DOCENTE_RICERCATORE è un sottoinsieme dell'unione di CORPO_DOCENTE e STUDENTE_LAUREATO e comprende tutto il corpo docente, così come gli studenti laureati che si mantengono con l'insegnamento o la ricerca. Infine, il tipo di entità BORSA_DI_STUDIO tie-

ne traccia delle borse di studio e dei contratti di ricerca assegnati all'università. Ogni borsa di studio ha come attributi il titolo della borsa di studio [Titolo], il numero della borsa di studio [Num], l'ente che la assegna [Ente], e la data d'inizio [DataIn]. Una borsa di studio è collegata a un direttore di ricerca [DR] e a tutti i ricercatori che finanzia [FINANZIA]. Ciascuna istanza di FINANZIA ha come attributi la data di inizio del finanziamento [Inizio], la data di fine del finanziamento (se nota) [Fine], e la percentuale di tempo che viene dedicata al progetto [Tempo] dal ricercatore finanziato.

Definizioni formali per i concetti del modello EER. Una classe¹³ è un insieme o collezione di entità; con ciò si include qualsiasi costrutto dello schema EER che raggruppi entità, come ad esempio tipo di entità, sottoclasse, superclasse e categoria. Una sottoclasse S è una classe le cui entità devono sempre essere un sottoinsieme delle entità presenti in un'altra classe, detta la superclasse C dell'associazione superclasse/sottoclasse (o È-UN). Indicheremo un'associazione di questo tipo con C/S . Per tale associazione superclasse/sottoclasse occorre sempre avere

$$S \subseteq C$$

Una specializzazione $Z = \{S_1, S_2, \dots, S_n\}$ è un insieme di sottoclassi che hanno la stessa superclasse G ; cioè, G/S_i è un'associazione superclasse/sottoclasse per $i = 1, 2, \dots, n$. G è detto tipo di entità generalizzato (o superclasse della specializzazione, o generalizzazione delle sottoclassi $\{S_1, S_2, \dots, S_n\}$). Si dice che Z è totale se si ha sempre (in ogni istante di tempo):

$$\bigcup_{i=1}^n S_i = G$$

altrimenti Z è detta parziale. Si dice che Z è disgiunta se si ha sempre:

$$S_i \cap S_j = \emptyset \text{ (insieme vuoto) per } i \neq j$$

altrimenti si dice che Z è sovrapposta.

Una sottoclasse S di C si dice definita tramite un predicato se si usa un predicato p sugli attributi di C per specificare quali entità di C sono anche membri di S ; cioè $S = C[p]$, dove $C[p]$ è l'insieme di entità di C che soddisfano p . Una sottoclasse non definita tramite un predicato è detta definita dall'utente.

Una specializzazione Z (o una generalizzazione G) è detta definita tramite un attributo se, per specificare l'insieme dei membri di ciascuna sottoclasse S_i di Z , viene usato un predicato ($A = c_i$), dove A è un attributo di G e c_i è un valore costante del dominio di A . Si noti che, se $c_i \neq c_j$ per $i \neq j$, e A è un attributo a valore singolo, allora la specializzazione sarà disgiunta.

Una categoria T è una classe sottoinsieme dell'unione di n superclassi che la definiscono D_1, D_2, \dots, D_n , $n > 1$, ed è formalmente specificata come segue:

$$T \subseteq (D_1 \cup D_2 \cup \dots \cup D_n)$$

¹³ Qui l'uso della parola classe differisce dall'uso più comune che ne è fatto nei linguaggi di programmazione orientati a oggetti, come ad esempio C++. In C++, infatti, una classe consiste nella definizione di un tipo strutturato insieme alle funzioni ad esso applicabili (operazioni).

Un predicato p_i sugli attributi di D_i può essere usato per specificare i membri di ciascun D_i che sono anche membri di T . Se un predicato è specificato su ciascun D_i , si ha

$$T = (D_1[p_1] \cup D_2[p_2] \cup \dots \cup D_n[p_n])$$

Ora si dovrà estendere la definizione di tipo di associazione data nel Capitolo 3 consentendo a ogni classe – non solo a ogni tipo di entità – di partecipare a un'associazione. Perciò si dovrà sostituire in quella definizione le parole tipo di entità con classe. La notazione grafica del modello EER è coerente con quella dell'ER perché tutte le classi sono rappresentate tramite rettangoli.

4.6 Uso dei diagrammi delle classi UML per la modellazione concettuale a oggetti

Le metodologie di modellazione a oggetti, come ad esempio l'UML (Universal Modeling Language) e l'OMT (Object Modeling Technique), stanno diventando sempre più popolari. Anche se esse sono state sviluppate principalmente per la progettazione del software, una parte importante di questa riguarda la progettazione delle basi di dati a cui si accederà tramite i moduli software. Perciò una parte importante di tali metodologie – costituita dai diagrammi delle classi¹⁴ – è per molti aspetti simile ai diagrammi EER. Purtroppo è la terminologia che spesso differisce. In questo paragrafo verrà passata brevemente in rassegna parte della notazione, terminologia e concetti usati nei diagrammi delle classi UML, e confrontata con la terminologia e la notazione EER. In Figura 3.15 viene mostrato come lo schema ER della base di dati AZIENDA di Figura 3.15 possa essere rappresentato usando la notazione UML. I tipi di entità in Figura 3.15 sono modellati come classi in Figura 4.11. Un'entità nel modello ER corrisponde a un oggetto in UML.

Nei diagrammi delle classi UML una classe è rappresentata graficamente con un riquadro che comprende tre sezioni: la sezione in alto fornisce il nome della classe; la sezione in mezzo comprende gli attributi per i singoli oggetti della classe; l'ultima sezione comprende operazioni che possono essere eseguite su questi oggetti. A differenza che nei diagrammi UML, nei diagrammi EER non sono specificate le operazioni. Si consideri la classe IMPIEGATO in Figura 4.11. I suoi attributi sono Nome, Ssn, DataN, Sesso, Indirizzo e Stipendio. Se lo desidera, il progettista ha facoltà di specificare il dominio di un attributo, ponendo un carattere: seguito da nome o descrizione del dominio (gli attributi Nome, Sesso e DataN di IMPIEGATO). Un attributo composto è modellato con un dominio strutturato, come illustrato dall'attributo Nome di IMPIEGATO. Un attributo multivalelo sarà generalmente modellato con una classe separata, come illustrato dalla classe SEDE.

I tipi di associazione sono detti legami (associations) nella terminologia UML, e le istanze di associazione sono dette collegamenti (links). Un legame binario (tipo di associazione binaria) è rappresentato con una linea che collega le classi partecipanti (tipi di entità), e può ave-

¹⁴ Una classe è simile a un tipo di entità, ma, a differenza di questo, può disporre di operazioni.

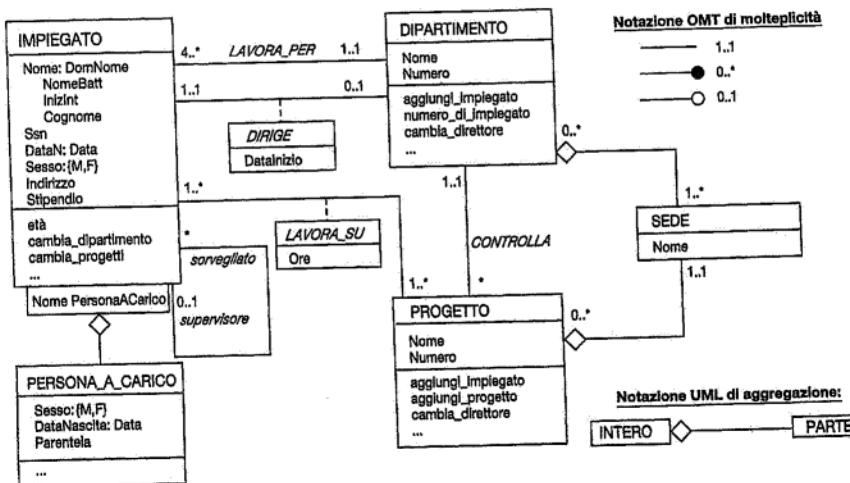


Figura 4.11 Lo schema concettuale UML per la base di dati AZIENDA di Figura 3.15.

re un nome (opzionale). Un attributo di associazione, detto *attributo di collegamento*, è posto in un riquadro che è collegato alla linea del legame tramite una linea tratteggiata. La notazione (min, max), descritta nel Sottoparagrafo 3.7.4, è usata per specificare vincoli di associazione, che sono detti *molteplicità* nella terminologia UML. Le molteplicità sono specificate nella forma *min..max*, e un asterisco (*) indica che non esiste limite massimo alla partecipazione. Ad ogni modo, le molteplicità sono poste *su parti opposte dell'associazione* se confrontate con la notazione esaminata nel Sottoparagrafo 3.7.4 (si confrontino le Figure 4.11 e 3.15). In UML, un asterisco da solo indica una molteplicità di 0..*, e un 1 da solo indica una molteplicità di 1..1. Un'associazione ricorsiva (si veda Sottoparagrafo 3.4.2) è detta *legame riflessivo* in UML, e i nomi di ruolo – come le molteplicità – vengono posizionati su lati opposti di un legame, se confrontati con il posizionamento dei nomi di ruolo in Figura 3.15.

In UML esistono due tipi di associazioni: il legame e l'aggregazione. L'*aggregazione* è pensata per rappresentare un'associazione tra un oggetto completo e le sue parti componenti, e ha una notazione diagrammatica distinta. In Figura 4.11 le sedi di un dipartimento e la singola sede di un progetto sono state modellate con aggregazioni. Comunque aggregazioni e legami non hanno proprietà strutturali diverse, e la scelta su quale tipo di associazione usare è piuttosto soggettiva. Nel modello EER sia i legami sia le aggregazioni sono rappresentate da associazioni. L'UML distingue anche tra legami/aggregazioni *unidirezionali* – che sono rappresentate con una freccia a indicare che c'è bisogno di una sola direzione per accedere a oggetti collegati – e legami/aggregazioni *bidirezionali* – che sono quelle assunte implicitamente (default). Inoltre è possibile specificare le istanze di associazione per poterle *ordinare*. In UML i nomi di associazione (legame) sono *opzionali*, e gli attributi di associazione sono rappresentati dentro un riquadro unito con una linea tratteggiata alla linea che rappresenta il legame/aggregazione (DataInizio e Ore in Figura 4.11).

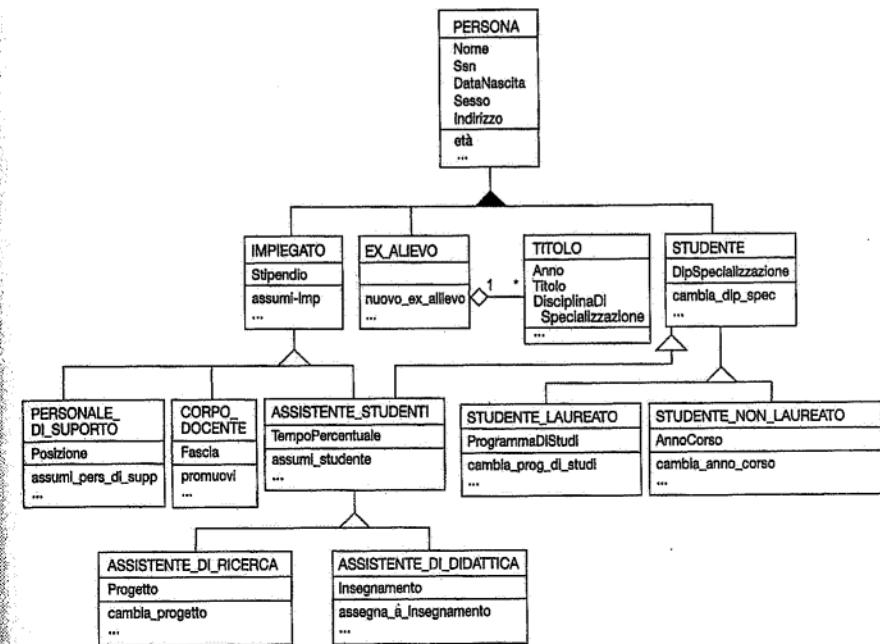


Figura 4.12 Notazione UML di specializzazione/generalizzazione illustrata tramite un diagramma delle classi corrispondente al diagramma EER di Figura 4.7.

Le operazioni fornite per ogni classe sono derivate dai requisiti funzionali dell'applicazione, come visto nel Paragrafo 3.1. All'inizio è in genere sufficiente specificare i nomi delle operazioni per indicare le operazioni logiche che si prevede di eseguire sui singoli oggetti di una classe, come mostrato in Figura 4.11. Man mano che il progetto si raffina, vengono aggiunti maggiori dettagli, come i tipi esatti degli argomenti (parametri) per ogni operazione, nonché una descrizione funzionale di ciascuna operazione. L'UML dispone di *descrizioni di funzioni* e di *diagrammi di sequenza* per specificare alcuni dettagli delle operazioni, ma ciò va al di là della portata della presente discussione, ed è generalmente descritto nei testi di ingegneria del software.

In UML le entità deboli possono essere modellate usando il costrutto detto *legame qualificato* (o *aggregazione qualificata*); esso può rappresentare sia l'associazione identificante sia la chiave parziale, posta in un riquadro attaccato alla classe proprietaria. Ciò è illustrato dalla classe PERSONA_A_CARICO e dalla sua aggregazione qualificata a IMPIEGATO in Figura 4.11.¹⁵

¹⁵ Le associazioni qualificate non si limitano alla modellazione di entità deboli, ma possono essere utilizzate per modellare anche altre situazioni.

In Figura 4.12 è illustrata la notazione UML per la generalizzazione/specializzazione, fornendo un possibile diagramma delle classi UML corrispondente al diagramma EER in Figura 4.7. Un triangolo vuoto indica una specializzazione/generalizzazione *disgiunta*, e un triangolo pieno indica *sovraposizione*.

La discussione e gli esempi sopra riportati forniscono una rapida visione d'insieme sui diagrammi delle classi e sulla terminologia UML. La bibliografia a fine capitolo fornisce alcune indicazioni di libri che descrivono completamente i dettagli dell'UML.

4.7 Tipi di associazione di grado maggiore di due

Nel Sottoparagrafo 3.4.2 è stato definito il **grado** di un tipo di associazione come il numero di tipi di entità che partecipano al tipo di associazione, e chiamato *binario* un tipo di associazione di grado due, e *ternario* un tipo di associazione di grado tre. In questo paragrafo verranno analizzate le differenze tra associazioni binarie e associazioni di grado maggiore, quando scegliere un'associazione binaria o un'associazione di grado più elevato, e i vincoli su associazioni di grado maggiore di due.

Scelta tra associazioni binarie e associazioni ternarie (o di grado più elevato). La notazione diagrammatica ER per un tipo di associazione ternario è mostrata in Figura 4.13(a), la quale rappresenta graficamente lo schema per il tipo di associazione FORNITURA che è stato rappresentato a livello di istanza in Figura 3.10. In generale, un tipo di associazione R di grado n avrà n spigoli in un diagramma ER, ognuno dei quali collega R a un diverso tipo di entità partecipante.

La Figura 4.13(b) presenta un diagramma ER per i tre tipi di associazione binari PUÒ_FORNIRE, USA e FORNISCE. In generale, un tipo di associazione ternario fornisce più informazioni di quanto non facciano tre tipi di associazione binari. Si considerino i tre tipi di associazioni binari PUÒ_FORNIRE, USA e FORNISCE. Si supponga che PUÒ_FORNIRE, tra FORNITORE e PARTE, comprenda un'istanza (s, p) ogni volta che il fornitore s può fornire parte p (a un progetto qualsiasi); USA, tra PROGETTO e PARTE, comprende un'istanza (j, p) ogni volta che il progetto j usa la parte p ; FORNISCE, tra FORNITORE e PROGETTO, comprende un'istanza (s, j) ogni volta che il fornitore s fornisce una qualche parte al progetto j . L'esistenza delle tre istanze di associazione (s, p) , (j, p) e (s, j) in PUÒ_FORNIRE, USA e FORNISCE, rispettivamente, non implica necessariamente che esista un'istanza (s, j, p) nell'associazione ternaria FORNITURA, perché le due cose hanno *significato diverso!* È spesso difficile decidere se una particolare associazione debba essere rappresentata come un tipo di associazione di grado n o se invece debba essere suddivisa in più tipi di associazione di grado inferiore. Il progettista deve basare questa decisione sulla semantica o significato della particolare situazione che viene rappresentata. Soluzione tipica è quella di inserire l'associazione ternaria *e anche* una o più delle associazioni binarie, a seconda delle necessità.

Alcuni strumenti per la progettazione di basi di dati sono basati su variazioni del modello che controllano le associazioni binarie. In questo caso un'associazione ternaria come

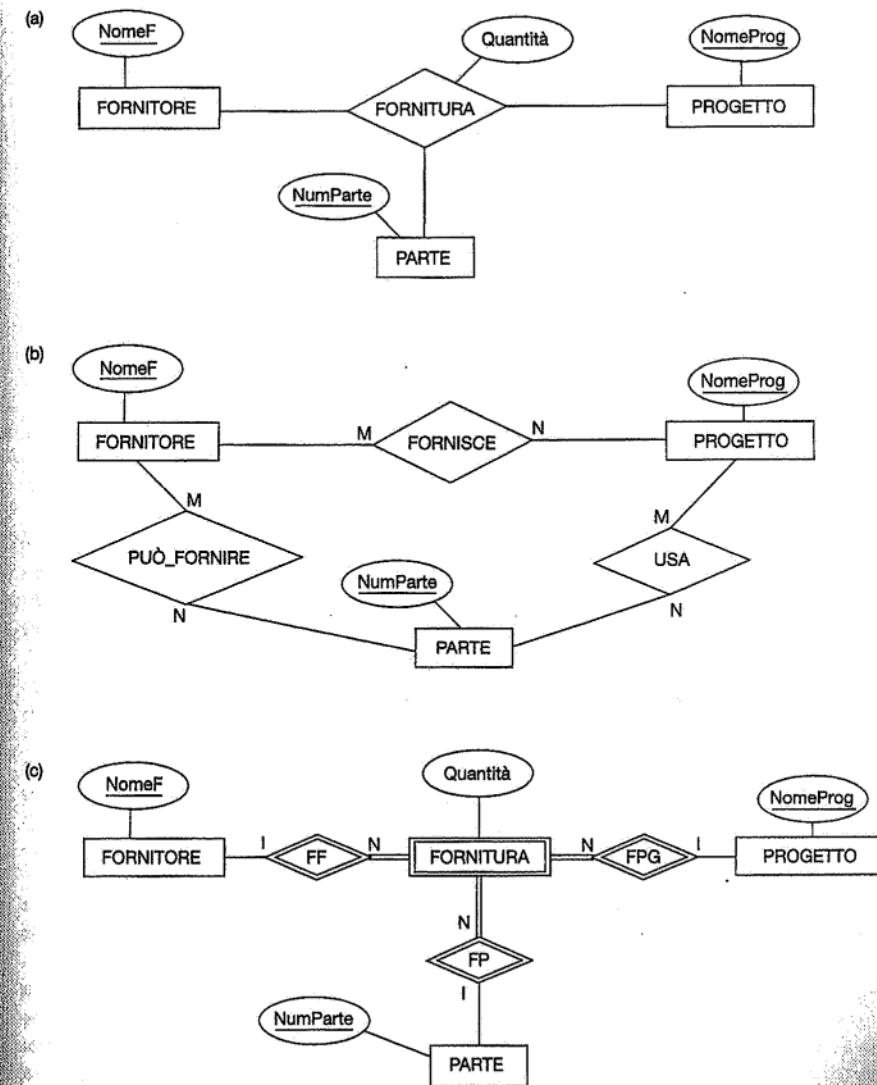


Figura 4.13 Illustrazione di tipi di associazione ternari. (a) Il tipo di associazione ternario FORNITURA. (b) Tre tipi di associazione binari, non equivalenti al tipo di associazione ternario FORNITURA. (c) FORNITURA rappresentato come un tipo di entità debole.

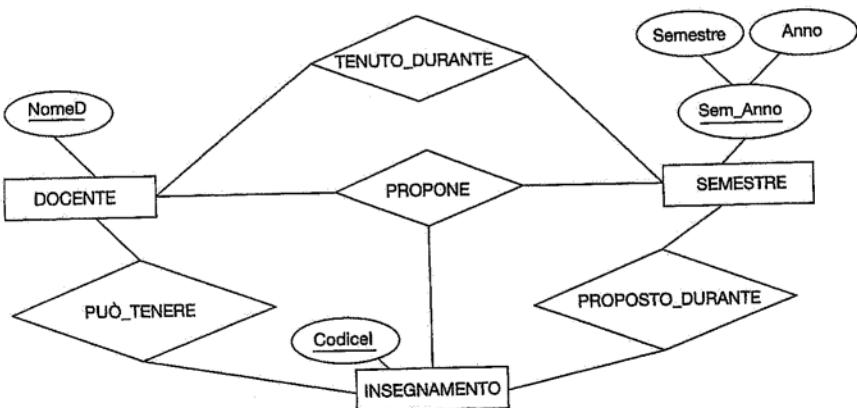


Figura 4.14 Un altro esempio di confronto fra tipi di associazione ternari e tipi di associazione binari.

FORNITURA deve essere rappresentata come un tipo di entità debole, senza chiave parziale e con tre associazioni identificanti. I tre tipi di entità partecipanti FORNITORE, PARTE e PROGETTO costituiscono nell'insieme i tipi di entità proprietari (Figura 4.13c). Perciò un'entità del tipo di entità debole FORNITURA di Figura 4.13(c) è identificata dalla combinazione delle sue tre entità proprietarie prese da FORNITORE, PARTE e PROGETTO.

Un altro esempio è mostrato in Figura 4.14. Il tipo di associazione ternario PROPONE rappresenta la situazione di docenti che propongono insegnamenti durante semestri specifici; perciò esso comprende un'istanza di associazione (i, s, c) ogni volta che il docente i ("i" come *instructor*) propone l'insegnamento c ("c" come *course*) durante il semestre s . I tre tipi di associazione binari mostrati in Figura 4.14 hanno il seguente significato: PUÒ_TENERE collega un insegnamento al docente che può tenere quell'insegnamento; TENUTO_DURANTE collega un semestre al docente che ha tenuto un certo insegnamento durante quel semestre; PROPOSTO_DURANTE collega un semestre agli insegnamenti proposti durante quel semestre *da qualsiasi docente*. In generale queste associazioni ternarie e binarie rappresentano situazioni diverse, ma comunque dovrebbero sussistere certi vincoli tra le associazioni. Ad esempio, in PROPONE non dovrebbe esistere un'istanza di associazione (i, s, c) se non esiste un'istanza (i, s) in TENUTO_DURANTE, un'istanza (s, c) in PROPOSTO_DURANTE e un'istanza (i, c) in PUÒ_TENERE. Però il contrario non è sempre vero; si possono avere cioè istanze $(i, s), (s, c)$ e (i, c) nei tre tipi di associazione binari senza per questo avere l'istanza corrispondente (i, s, c) in PROPONE. In presenza di certi *vincoli aggiuntivi*, però, può valere anche quest'ultima proprietà – ad esempio, se l'associazione PUÒ_TENERE è 1:1 (un docente può tenere un solo insegnamento, e un insegnamento può essere tenuto da un solo docente). Il progettista dello schema deve analizzare ogni situazione specifica per decidere quali tipi di associazioni binarie e ternarie sono necessari.

Si noti che è possibile avere un tipo di entità debole con un tipo di associazione identificante ternario (o n -ario). In questo caso il tipo di entità debole può avere più tipi di entità proprietari. Un esempio è mostrato in Figura 4.15.

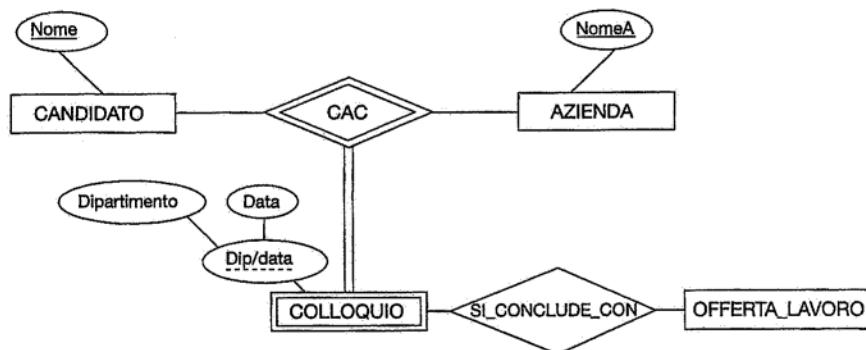


Figura 4.15 Un tipo di entità debole COLLOQUIO, con un tipo di associazione identificante ternario.

Vincoli su associazioni ternarie (o di grado maggiore). Ci sono due possibili notazioni per specificare vincoli strutturali su associazioni n -arie, ed esse specificano vincoli diversi. Dovrebbero perciò essere usate entrambe se è importante specificare pienamente i vincoli strutturali su un'associazione ternaria o di grado più elevato. La prima notazione è basata sulla notazione del rapporto di cardinalità per associazioni binarie, rappresentata in Figura 3.2. In questo caso 1, M o N sono specificati su ogni arco partecipante. Si illustra qui tale vincolo usando l'associazione FORNITURA di Figura 4.13.

Si ricordi che l'insieme di associazioni di FORNITURA è un insieme di istanze di associazione (s, j, p) , dove s è un FORNITORE, j è un PROGETTO e p è una PARTE. Si supponga che esista il vincolo che per una specifica combinazione progetto-parte possa essere usato solo un fornitore (solo un fornitore fornisce una parte specifica a uno specifico progetto). In questo caso, in Figura 4.13, si pone 1 sulla partecipazione di FORNITORE, e M, N sulle partecipazioni di PROGETTO, PARTE. Ciò stabilisce il vincolo che una specifica combinazione (j, p) può apparire al più una volta nell'insieme di associazioni. Perciò ogni istanza di associazione (s, j, p) è univocamente identificata nell'insieme di associazioni dalla sua combinazione (j, p) , il che rende (j, p) una chiave per l'insieme di associazioni. In generale, alle partecipazioni che hanno su di sé esplicitato un 1 non è richiesto di far parte della chiave per l'insieme di associazioni.¹⁶

La seconda notazione è basata sulla notazione (min, max) rappresentata in Figura 3.15 per associazioni binarie. Qui un (min, max) su una partecipazione precisa che ogni entità è collegata ad almeno min e al più a max istanze di associazione nell'insieme di associazioni. Questi vincoli non hanno alcun rapporto con la determinazione della chiave di un'associazione n -aria, se $n > 2$,¹⁷ ma stabiliscono un tipo diverso di vincolo che pone restrizioni sul numero di istanze di associazione a cui può partecipare ogni entità.

¹⁶ Ciò è vero anche per i rapporti di cardinalità delle associazioni binarie.

¹⁷ Tuttavia il vincolo (min, max) può determinare le chiavi per associazioni binarie.

4.8 Concetti di astrazione dei dati e di rappresentazione della conoscenza

In questo paragrafo verranno esaminati in termini astratti alcuni dei concetti di modellazione descritti in modo piuttosto dettagliato nella presentazione dei modelli ER e EER nei Capitoli 3 e 4. La terminologia adottata è usata sia nella letteratura sulla modellazione concettuale dei dati sia in quella di intelligenza artificiale quando si tratta la **rappresentazione della conoscenza (knowledge representation, KR)**. Lo scopo delle tecniche KR è quello di sviluppare concetti per modellare accuratamente un certo **dominio del discorso** tramite la creazione di un'**ontologia**¹⁸ che descrive i concetti del dominio. Questa ontologia è poi usata per immagazzinare e gestire conoscenza per trarre conclusioni, prendere decisioni o semplicemente rispondere a domande. Gli scopi della rappresentazione della conoscenza sono simili a quelli dei *modelli di dati semanticici*, ma è possibile elencare alcune importanti somiglianze e differenze tra le due discipline.

- Entrambe usano un processo di astrazione per individuare proprietà comuni e aspetti importanti di oggetti nel mini-mondo (dominio del discorso) e nel contempo per eliminare differenze insignificanti e dettagli ininfluenti.
- Entrambe forniscono concetti, vincoli, operazioni e linguaggi per definire dati e rappresentare conoscenza.
- La rappresentazione della conoscenza ha generalmente portata più ampia dei modelli di dati semanticici. Negli schemi KR (*KR schemes*) sono rappresentate forme diverse di conoscenza, come regole (usate in inferenza, deduzione e ricerca), conoscenza incompleta e per difetto e conoscenza temporale e spaziale. I modelli di basi di dati stanno via via espandendosi per includere alcuni di questi concetti.
- Gli schemi KR comprendono **meccanismi di ragionamento** che deducono fatti aggiuntivi dai fatti memorizzati in una base di dati. Perciò, mentre la maggior parte degli attuali sistemi di basi di dati si limitano a rispondere a interrogazioni dirette, i sistemi basati sulla conoscenza che usano schemi KR possono rispondere a interrogazioni che comportano **inferenze sui dati memorizzati**. Attualmente la tecnologia delle basi di dati sta per essere estesa con meccanismi di inferenza.
- Mentre la maggior parte dei modelli di dati si concentrano sulla rappresentazione di schemi di basi di dati, o meta-conoscenza, gli schemi KR spesso mescolano schemi con istanze per fornire flessibilità nel rappresentare eccezioni. Ciò spesso comporta inefficienze quando gli schemi KR sono implementati, specialmente quando vengono confrontati con basi di dati e quando è necessario immagazzinare una gran quantità di dati (o fatti).

Quattro sono i **concetti di astrazione** usati sia nei modelli di dati semanticici, come il modello EER, sia negli schemi KR: (1) classificazione e istanziazione, (2) identificazione, (3) specializzazione e generalizzazione, e (4) aggregazione e associazione. I concetti accoppiati

¹⁸ Un'ontologia è sotto molti aspetti simile a uno schema concettuale, ma con più conoscenza, regole ed eccezioni.

di classificazione e istanziazione sono uno l'inverso dell'altro, così come la generalizzazione e la specializzazione. Anche i concetti di aggregazione e associazione sono collegati. Qui si esamineranno tali concetti astratti e le loro relazioni con le rappresentazioni concrete usate nel modello EER per chiarire il processo di astrazione dei dati e per migliorare la comprensione del processo collegato di progettazione dello schema concettuale.

4.8.1 Classificazione e istanziazione

Il processo di **classificazione** comporta un'assegnazione sistematica di oggetti/entità simili a classi di oggetti/tipi di entità, rispettivamente. È possibile descrivere (nelle basi di dati) o ragionare su (nella rappresentazione della conoscenza) classi piuttosto che oggetti singoli. Collezioni di oggetti condividono gli stessi tipi di attributi, associazioni e vincoli, e tramite la classificazione di oggetti si semplifica il processo di scoperta delle loro proprietà. L'**istanziazione** è l'opposto della classificazione e si riferisce alla generazione e all'esame specifico di oggetti distinti di una classe. Perciò un'istanza di oggetto è collegata alla sua classe di oggetti tramite l'**associazione È-UN'ISTANZA-DI**.¹⁹

In generale gli oggetti di una classe dovrebbero avere una struttura di tipo simile. Però alcuni oggetti possono mostrare proprietà che differiscono per certi aspetti da quelle di altri oggetti della classe; anche questi **oggetti d'eccezione** devono essere modellati, e gli schemi KR consentono eccezioni più variate rispetto ai modelli di basi di dati. Inoltre certe proprietà si applicano alla classe come a un tutto e non ai singoli oggetti; gli schemi KR consentono queste **proprietà di classe**.²⁰

Nel modello EER, le entità sono classificate in tipi di entità secondo le loro proprietà e la loro struttura di base. Sono poi ulteriormente classificate in sottoclassi e categorie basate su ulteriori somiglianze e differenze (eccezioni) esistenti fra loro. Le istanze di associazione sono classificate in tipi di associazione. Perciò tipi di entità, sottoclassi, categorie e tipi di associazione costituiscono i diversi tipi di classe nel modello EER. Il modello EER non contempla esplicitamente proprietà di classe, ma può essere esteso per fare ciò. In UML gli oggetti sono classificati in classi, ed è possibile rappresentare sia proprietà di classe sia oggetti singoli.

I modelli di rappresentazione della conoscenza consentono più schemi di classificazione nei quali una classe è un'*istanza* di un'altra classe (detta **meta-classe**). Si noti che ciò *non può essere* rappresentato direttamente nel modello EER, perché si hanno solo due livelli – classi e istanze. La sola associazione tra classi nel modello EER è l'**associazione superclasse/sottoclasse**, mentre in alcuni schemi KR un'**associazione aggiuntiva classe/istanza** può essere rappresentata direttamente in una gerarchia di classi. Un'istanza può essere essa stessa un'altra classe, consentendo così schemi di classificazione a livello multiplo.

¹⁹ I diagrammi UML consentono una forma di istanziazione permettendo la rappresentazione grafica di singoli oggetti. Questa caratteristica non è però stata descritta nel Paragrafo 4.6.

²⁰ Anche i diagrammi UML consentono la specificazione di proprietà di classe.

4.8.2 Identificazione

L'**identificazione** è il processo di astrazione per mezzo del quale classi e oggetti sono resi univocamente identificabili grazie a un certo **identificatore**. Ad esempio, un nome di classe identifica univocamente un'intera classe. È necessario un ulteriore meccanismo per distinguere istanze di oggetti distinti tramite identificatori di oggetti. Inoltre è necessario individuare manifestazioni multiple, nella base di dati, dello stesso oggetto del mondo reale. Ad esempio, può capitare di avere una tupla <Matthew Clarke, 610618, 376-9821> in una relazione PERSONA e un'altra tupla <301-54-0836, CS, 3,8> in una relazione STUDENTE che rappresentano la stessa entità del mondo reale. Non c'è alcun modo di identificare il fatto che questi due oggetti (tuple) della base di dati rappresentano la stessa entità del mondo reale se non si provvede al momento della progettazione a un riferimento incrociato appropriato per consentire questa identificazione. Perciò l'identificazione è necessaria a due livelli:

- per distinguere gli oggetti e le classi della base di dati;
- per individuare gli oggetti della base di dati e per collegarli ai loro equivalenti del mondo reale.

Nel modello EER l'identificazione dei costrutti dello schema è basata su un sistema di nomi univoci per i costrutti. Ad esempio ogni classe – sia essa un tipo di entità, una sottoclasse, una categoria o un tipo di associazione – deve avere un nome distinto. I nomi degli attributi di una data classe devono pure essere distinti. Sono anche necessarie delle regole per identificare senza ambiguità i riferimenti a un nome di attributo in un reticolo o gerarchia di specializzazioni o generalizzazioni.

A livello di oggetto, i valori degli attributi chiave sono usati per fare una distinzione tra entità di un particolare tipo di entità. Per tipi di entità deboli, le entità sono identificate da una combinazione dei valori delle loro chiavi parziali e delle entità, dei tipi di entità proprietari, a cui sono correlate. Le istanze di associazione sono identificate da una certa combinazione delle entità che esse collegano, a seconda del rapporto di cardinalità specificato.

4.8.3 Specializzazione e generalizzazione

Per specializzazione si intende quel processo che consiste nella classificazione di una classe di oggetti in sottoclassi più specializzate. La generalizzazione è il processo inverso, che consiste nel generalizzare numerose classi in una classe astratta di più alto livello, che comprende gli oggetti presenti in tutte queste classi. La specializzazione è un raffinamento concettuale, mentre la generalizzazione è una sintesi concettuale. Nel modello EER sono usate le sottoclassi per rappresentare la specializzazione e la generalizzazione. L'associazione tra una sottoclasse e la sua superclasse associazione è nota come **È-UNA-SOTTOCLASSE-DI** o semplicemente associazione **È-UN**.

4.8.4 Aggregazione e associazione

L'**aggregazione** è un concetto di astrazione che consente di costruire oggetti composti a partire dagli oggetti che li compongono. Ci sono tre casi in cui questo concetto può essere messo in relazione con il modello EER. Il primo caso è quello in cui si aggregano i valori di attributi di un oggetto per formare l'oggetto intero. Il secondo caso si presenta quando si rappresenta un'associazione di aggregazione come un'associazione ordinaria. Il terzo caso, non contemplato esplicitamente dal modello EER, prevede la possibilità di combinare oggetti che siano collegati da una particolare istanza di associazione in un *oggetto aggregato di più alto livello*. Ciò è qualche volta utile quando l'oggetto aggregato di più alto livello deve esso stesso essere collegato a un altro oggetto. L'associazione tra gli oggetti primitivi e il loro oggetto aggregato sarà nota come **È-UNA-PARTE-DI**; l'associazione inversa è detta **È-UN-COMPONENTE-DI**. L'UML contempla tutti e tre i tipi di aggregazione.

L'astrazione di **associazione** è usata per associare oggetti provenienti da più *classi indipendenti*, per cui è piuttosto simile al secondo uso dell'aggregazione. È rappresentata nel modello EER tramite tipi di associazione e nel modello UML tramite legami. Questa associazione astratta è detta **È-ASSOCIAZIONE-CON**.

Al fine di una migliore comprensione dei diversi usi possibili dell'aggregazione, si consideri lo schema ER mostrato in Figura 4.16(a), che contiene informazioni su colloqui di lavoro sostenuti da candidati presso varie aziende. La classe AZIENDA è un'aggregazione degli attributi (od oggetti componenti) NomeA (nome azienda) e IndirizzoA (indirizzo azienda), mentre CANDIDATO è un aggregato di Ssn, Nome, Indirizzo e Telefono. Gli attributi di associazione NomeContatto e TelefonoContatto rappresentano il nome e il numero di telefono della persona dell'azienda responsabile del colloquio di lavoro. Si supponga che alcuni colloqui si concludano con offerte di lavoro e altri no. Si vorrebbe qui trattare COLLOQUIO come una classe per associarla a OFFERTA_LAVORO. Lo schema mostrato in Figura 4.16(b) è *sbagliato* perché esso richiede che ogni istanza dell'associazione colloquio comporti un'offerta di lavoro. Lo schema mostrato in Figura 4.16(c) non è consentito, perché il modello ER non consente associazioni fra associazioni (il modello UML, invece, le consente).

Un modo per rappresentare questa situazione è quello di creare una classe aggregata di più alto livello composta da AZIENDA, CANDIDATO e COLLOQUIO e di collegare questa classe a OFFERTA_LAVORO, come mostrato in Figura 4.16(d). Sebbene il modello EER, per come è stato descritto in questo libro, non consenta questa possibilità, alcuni modelli semanticici di dati la consentono, e chiamano l'oggetto risultante **oggetto composto o molecolare**. Altri modelli trattano in modo uniforme i tipi di entità e i tipi di associazione e perciò permettono associazioni fra associazioni (Figura 4.16c).

Per rappresentare correttamente questa situazione nel modello ER, così come esso è stato descritto qui, è necessario creare un nuovo tipo di entità debole COLLOQUIO, come mostrato in Figura 4.16(e), e correlarlo a OFFERTA_LAVORO. Perciò si può sempre rappresentare correttamente queste situazioni nel modello ER tramite la creazione di tipi di entità aggiuntivi, anche se può essere concettualmente più desiderabile consentire una rappresentazione diretta dell'aggregazione come in Figura 4.16(d), o consentire associazioni fra associazioni come in Figura 4.16(c).

La principale differenza strutturale tra aggregazione e associazione è che, quando un'i-

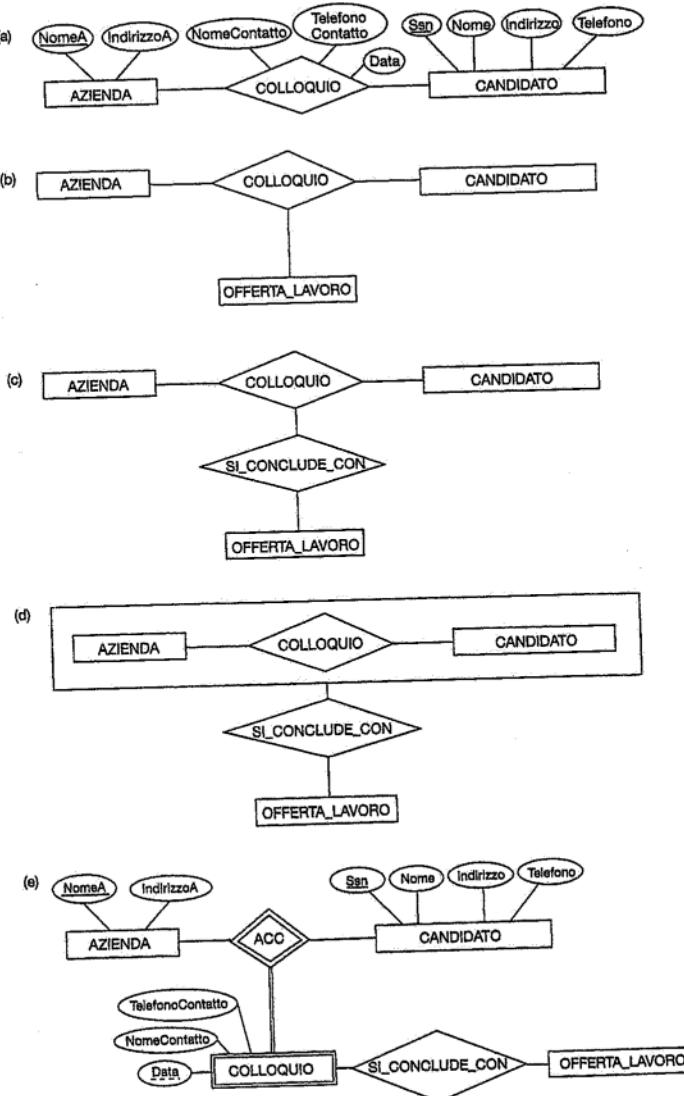


Figura 4.16 Illustrazione dell'aggregazione. (a) Il tipo di associazione **COLLOQUIO**. (b) Inserimento di **OFFERTA_LAVORO** in un tipo di associazione ternario (errato). (c) Inserimento di **OFFERTA_LAVORO** in un tipo di associazione alla quale partecipa un'altra associazione (in genere ciò non è consentito nel modello ER). (d) Uso dell'aggregazione e di un oggetto composto (molecolare) (in genere ciò non è consentito nel modello ER). (e) Rappresentazione corretta nel modello.

stanza di associazione viene cancellata, gli oggetti partecipanti possono continuare a esistere. Però, se si supporta la nozione di oggetto aggregato – ad esempio, un'AUTOMOBILE che è costituita dagli oggetti MOTORE, TELAIO e PNEUMATICI – allora la cancellazione dell'oggetto aggregato AUTOMOBILE equivale alla cancellazione di tutti i suoi oggetti componenti.

Sommario

In questo capitolo abbiamo prima di tutto studiato alcune estensioni del modello ER che aumentano le sue possibilità di rappresentazione. Abbiamo chiamato il modello risultante modello ER-esteso o modello EER (enhanced-ER). Sono stati presentati il concetto di sottoclasse e relativa superclasse e il correlato meccanismo di ereditarietà di attributi/associazioni. Abbiamo visto come sia talora necessario creare classi aggiuntive di entità, o per la presenza di attributi aggiuntivi specifici, o per la presenza di tipi di associazione specifici. Abbiamo esaminato due processi principali di definizione di gerarchie e reticolati di superclassi/sottoclassi – la specializzazione e la generalizzazione.

Abbiamo poi mostrato come rappresentare graficamente questi nuovi costrutti in un diagramma EER. Abbiamo anche esaminato i vari tipi di vincoli che si possono applicare alla specializzazione o alla generalizzazione. I due vincoli principali sono totale/parziale e disgiunta/sovraposta. Inoltre può essere specificato un predicato di definizione per una sottoclasse o un attributo di definizione per una specializzazione. Si sono quindi discusse le differenze tra sottoclassi definite dall'utente e sottoclassi definite tramite un predicato, nonché tra specializzazioni definite dall'utente e specializzazioni definite tramite un attributo. Infine abbiamo studiato il concetto di categoria, che è un sottoinsieme dell'unione di due o più classi, e abbiamo dato definizioni formali di tutti i concetti presentati.

Si è poi introdotta la notazione e la terminologia del linguaggio universale di modellazione (UML: Universal Modeling Language), che è sempre più usato nell'ingegneria del software. Abbiamo quindi esaminato brevemente le somiglianze e le differenze tra i concetti, la notazione e la terminologia UML e EER. Abbiamo anche discusso alcuni dei problemi riguardanti le differenze tra associazioni binarie e associazioni di grado più elevato, in quali circostanze ognuna di esse dovrebbe essere usata quando si progetta uno schema concettuale, e come possano essere specificati tipi diversi di vincoli su associazioni n -arie. Nel Paragrafo 4.8 abbiamo brevemente studiato la disciplina di rappresentazione della conoscenza e come essa è collegata alla modellazione semantica di dati. Abbiamo anche fornito una visione d'insieme e un sommario sui vari tipi di concetti di rappresentazione astratta dei dati: classificazione e istanziazione, identificazione, specializzazione e generalizzazione, aggregazione e associazione, e abbiamo visto come i concetti EER e UML sono collegati a ciascuno di questi.

Questionario di verifica

- 4.1. Cos'è una sottoclasse? Quando è necessaria una sottoclasse nella modellazione dei dati?

- 4.2. Si definiscano i seguenti termini: *superclasse di una sottoclasse*, *associazione superclasse/sottoclasse*, *associazione È-UN*, *specializzazione*, *generalizzazione*, *categoria*, *attributi specifici (locali)*, *associazioni specifiche*.
- 4.3. Si esamini il meccanismo di ereditarietà di attributi/associazioni. Perché è utile?
- 4.4. Si trattino le sottoclassi definite dall'utente e quelle definite tramite un predicato, e si identifichino le differenze tra questi due tipi di sottoclassi.
- 4.5. Si trattino le specializzazioni definite dall'utente e quelle definite tramite un attributo, e si identifichino le differenze tra questi due tipi di specializzazioni.
- 4.6. Si esaminino i due tipi principali di vincoli su specializzazione e generalizzazione.
- 4.7. Qual è la differenza tra una gerarchia di specializzazioni e un reticolo di specializzazioni?
- 4.8. Qual è la differenza tra specializzazione e generalizzazione? Perché noi non evidenziamo questa differenza nei diagrammi di schema?
- 4.9. In cosa differisce una categoria da una normale sottoclasse condivisa? Per cosa è usata una categoria? Si illustri la risposta con opportuni esempi.
- 4.10. Per ciascuno dei seguenti termini UML si esamini il termine corrispondente nel modello EER, se esiste: *oggetto*, *classe*, *legame*, *aggregazione*, *generalizzazione*, *moltiplicità*, *attributi*, *discriminatore*, *collegamento*, *attributo di collegamento*, *legame riflessivo*, *legame qualificato*.
- 4.11. Si esaminino le differenze principali tra la notazione per i diagrammi di schema EER e quella per i diagrammi delle classi UML, confrontando come sono rappresentati nei due casi i concetti comuni.
- 4.12. Si esaminino le due notazioni per specificare vincoli su associazioni *n*-arie, e si illustri il possibile uso di ciascuna.
- 4.13. Si elenchino i vari concetti di astrazione dei dati e i corrispondenti concetti di modellazione del modello EER.
- 4.14. Quale caratteristica di aggregazione manca nel modello EER? Come può essere ulteriormente esteso il modello EER per supportarla?
- 4.15. Quali sono le principali differenze e caratteristiche comuni tra le tecniche di modellazione concettuale di una base di dati e le tecniche di rappresentazione della conoscenza?

Esercizi

- 4.16. Si progetti uno schema EER per un'applicazione di basi di dati a cui si è interessati. Si specifichino tutti i vincoli che dovrebbero sussistere nella base di dati. Ci si accerti che lo schema abbia almeno cinque tipi di entità, quattro tipi di associazione, un tipo di entità debole, un'associazione superclasse/sottoclasse, una categoria e un tipo di associazione *n*-ario ($n > 2$).
- 4.17. Si consideri lo schema ER BANCA di Figura 3.17, e si supponga che sia necessario tener traccia di diversi tipi di CONTI (DEPOSITI_RISPARMIO, CONTI_CORRENTI, ...) e PRESTITI (PRESTITI_AUTO, PRESTITI_CASA, ...). Si supponga che si desideri anche tener traccia (tramite TRANSAZIONE) delle transazioni di ogni conto (depositi, prelevamenti,

assegni, ...) e (tramite PAGAMENTO) dei pagamenti di ciascun prestito; sia le transazioni sia i pagamenti comprendono l'importo, la data e l'ora. Si modifichi lo schema BANCA usando i concetti ER e EER di specializzazione e generalizzazione. Si enunci esplicitamente ogni assunzione fatta riguardante i requisiti aggiuntivi.

- 4.18. Il seguente resoconto descrive una versione semplificata dell'organizzazione delle strutture olimpiche progettate per le Olimpiadi di Atlanta del 1996. Si disegni un diagramma EER che mostri i tipi di entità, gli attributi, le associazioni e le specializzazioni per questa applicazione. Si enunci esplicitamente ogni assunzione fatta. Le strutture olimpiche sono distribuite in complessi sportivi. I complessi sportivi si suddividono in complessi *per un solo sport* e complessi *polisportivi*. I complessi polisportivi hanno aree interne destinate a ciascuno sport con un indicatore di collocazione (ad es., centro, angolo-NE ecc.). Un complesso ha una collocazione, un organizzatore capo, un'area totale occupata e così via. In ogni complesso è tenuta una serie di eventi (ad es. nello stadio di atletica possono tenersi molte gare diverse). Per ogni evento c'è una data prevista, una durata prevista, un numero previsto di partecipanti, un numero previsto di giudici eccetera. Sarà tenuto un elenco di tutti i giudici insieme con la lista di eventi a cui parteciperà ciascun giudice. È necessaria un'attrezzatura diversa per gli eventi (ad es. pali delle porte, aste, parallele), come pure per la manutenzione. I due tipi di struttura (*per un solo sport* e *polisportiva*) avranno tipi diversi di informazioni. Per ciascun tipo di struttura si memorizza il numero di servizi necessari, insieme a un budget approssimato.

- 4.19. Si individuino tutti i concetti importanti rappresentati nel caso di studio della base di dati biblioteca sotto descritto. In particolare, si individui l'astrazione di classificazione (tipi di entità e tipi di associazione), aggregazione, identificazione e specializzazione/generalizzazione. Si specifichino i vincoli di cardinalità (min, max), quando possibile. Si elenchino i dettagli che incideranno su un futuro progetto, ma che non hanno alcun rapporto con la progettazione concettuale. Si elenchino separatamente i vincoli semantici. Si disegni un diagramma EER della base di dati biblioteca.

Caso di Studio: La Biblioteca Tecnica della Georgia (GTL: Georgia Tech Library) ha approssimativamente 16.000 soci, 100.000 titoli e 250.000 volumi (ossia una media di 2,5 copie per libro). In un generico istante circa il 10 per cento dei volumi è fuori in prestito. I bibliotecari cercano di garantire che i libri siano disponibili quando i soci vogliono prenderli a prestito. Inoltre i bibliotecari devono sapere in ogni istante quante copie di ciascun libro ci sono nella biblioteca o sono fuori in prestito. È disponibile in linea un catalogo di libri che elenca i libri per autore, titolo e soggetto. Per ciascun titolo presente nella biblioteca si tiene nel catalogo una descrizione del libro, di dimensioni che vanno da una frase a molte pagine. I bibliotecari addetti alla consultazione vogliono poter accedere a questa descrizione quando i soci richiedono informazioni su un libro. Il personale della biblioteca è diviso in bibliotecario capo, bibliotecari associati dipartimentali, bibliotecari addetti alla consultazione, personale di controllo e assistenti di biblioteca. I libri possono essere tenuti in prestito per 21 giorni. Ai soci è consentito avere in prestito solo cinque libri alla volta. I soci di solito restituiscono i libri nel giro di tre o quattro settimane. Molti soci sanno di avere una settimana di proroga prima che venga loro inviato un avviso, perciò cercano di restituire il libro prima che finisca il periodo di proroga. Al 5 per cento circa dei soci devono essere inviate lettere di

sollecito perché restituiscano il libro. La maggior parte dei libri con prestito scaduto viene restituita entro un mese dalla data dovuta. Approssimativamente il 5 per cento dei libri con prestito scaduto viene ulteriormente trattenuto o addirittura non è mai più restituito. Soci più attivi della biblioteca sono definiti quelli che ricorrono al prestito almeno dieci volte l'anno. L'1 per cento più attivo dei soci effettua il 15 per cento dei prestiti, mentre il 10 per cento più attivo dell'insieme dei soci effettua il 40 per cento dei prestiti totali. Circa il 20 per cento dei soci è totalmente inattivo, nel senso che sono soci ma non ricorrono mai al prestito. Per diventare socio della biblioteca i candidati riempiono un modulo che richiede il loro SSN, indirizzo postale al campus e di casa, nonché i loro numeri di telefono. Dopo di ciò i bibliotecari rilasciano una tessera numerata, leggibile da una macchina, con la fotografia del socio. Questa tessera è valida per quattro anni. Un mese prima della scadenza della tessera viene inviato al socio un avviso per il rinnovo. I professori dell'istituto sono automaticamente considerati soci. Quando un nuovo membro del corpo docente entra a far parte dell'istituto, le informazioni a lui relative sono prelevate dai record degli impiegati e una tessera della biblioteca viene inviata al suo indirizzo nel campus. Ai professori è concesso di tenere i libri in prestito per periodi di tre mesi, con un periodo di proroga di due settimane. Gli avvisi di rinnovo rivolti ai professori sono inviati all'indirizzo del campus. La biblioteca non presta alcuni libri, come ad esempio libri di consultazione, libri rari, e carte geografiche. I bibliotecari devono distinguere fra libri che possono essere prestati e libri che non possono essere prestati. Inoltre, i bibliotecari hanno un elenco di libri al cui acquisto sono interessati ma che non possono ottenere, come ad esempio libri rari o esauriti, e un elenco di libri che sono stati persi o distrutti ma non sono stati sostituiti. I bibliotecari devono avere un sistema che tenga traccia dei libri che non possono essere prestati, nonché dei libri al cui acquisto sono interessati. Alcuni libri possono avere lo stesso titolo, e pertanto il titolo non può essere usato come uno strumento di identificazione. Ogni libro è identificato dal suo codice ISBN (International Standard Book Number: numero internazionale standard per libri), un codice univoco internazionale assegnato a tutti i libri. Due libri con lo stesso titolo possono avere diversi ISBN se sono scritti in lingue diverse o hanno diverse rilegature (copertina rigida o pieghevole). Le varie edizioni dello stesso libro hanno diversi ISBN. Il sistema di basi di dati proposto deve essere progettato per tener traccia di soci, libri, catalogo e attività di prestito.

4.20. Si progetti una base di dati che memorizzi informazioni per un museo d'arte. Si supponga che siano stati raccolti i seguenti requisiti.

- Il museo ha una collezione di oggetti d'arte. Ogni OGGETTO_D'ARTE ha un NumId univoco, un Artista (se noto), un Anno (in cui è stato creato, se noto), un Titolo e una Descrizione.
 - Gli oggetti d'arte sono classificati in base al loro tipo. Ci sono tre tipi principali: DIPINTO, SCULTURA e STATUA, più un altro tipo detto ALTRO per accogliere oggetti che non ricadono in uno dei tre tipi principali.
 - Un DIPINTO ha un TipoPittura (olio, acquerello ecc.), un materiale sul quale è stato Steso (carta, tela di canapa, legno ecc.), e uno Stile (moderno, astratto ecc.).
 - Una SCULTURA ha un Materiale con cui è stata creata (legno, pietra ecc.), un'Altezza, un Peso e uno Stile.
 - Un oggetto d'arte nella categoria ALTRO ha un Tipo (stampa, foto ecc.) e uno Stile.
- Gli oggetti d'arte sono anche classificati come di COLLEZIONE_PERMANENTE, che sono quelli posseduti dal museo (che ha informazioni sulla DataAcquisto, sul fatto che sia InMostra o in magazzino, e sul Costo) o PRESTATO, che ha informazioni sulla Collezione (da cui è stato preso a prestito), DataPrestito e DataRestituzione.
- Gli oggetti d'arte hanno anche informazioni che descrivono il loro paese/cultura usando informazioni su paese/cultura d'Origine (italiano, egiziano, americano, indiano ecc.), Epoca (Rinascimentale, Moderna, Antica ecc.).
- Il museo tiene traccia di informazioni sull'ARTISTA, se note: Nome, DataNascita, DataMorte (se non è vivente), PaeseOrigine, Epoca, StilePrincipale, Descrizione. Si suppone che il Nome sia univoco.
- È organizzata più di una ESPOSIZIONE, ognuna delle quali ha un Nome, una DataInizio, una DataFine, ed è collegata a tutti gli oggetti d'arte che sono stati esposti durante l'esposizione.
- Si tengono informazioni (tramite COLLEZIONE) su altre collezioni con cui il museo interagisce; queste informazioni comprendono il Nome (univoco), il Tipo (museo, personale ecc.), la Descrizione, l'Indirizzo, il Telefono e la PersonaContatto attuale.

Si disegni un diagramma di schema EER per questa applicazione. Si discuta ogni assunzione fatta e che giustifichi le scelte di progettazione EER.

- 4.21. In Figura 4.17 è mostrato un esempio di un diagramma EER per la base di dati di un piccolo aeroporto privato, usata per memorizzare le informazioni relative agli aeroplani, ai loro proprietari, agli impiegati dell'aeroporto e ai piloti. A partire dai requisiti per questa base di dati sono state raccolte le seguenti informazioni. Ogni aeroplano ha un numero di matricola [Mat#], fa parte di uno specifico tipo di aeroplano [DI_TIPO] e viene posto in un hangar specifico [POSTO_IN]. Ogni tipo di aeroplano ha un numero di modello [Modello], una capacità [Capacità] e un peso [Peso]. Ogni hangar ha un numero [Numero], una capacità [Capacità], e un'ubicazione [Ubicazione]. La base di dati tiene anche traccia dei proprietari di ogni aeroplano [POSSIEDE] e degli impiegati che hanno effettuato la manutenzione dell'aeroplano [EFFETTUA_MANUTENZIONE]. Ogni istanza di associazione in POSSIEDE collega un aeroplano a un proprietario e comprende la data di acquisto [DataA]. Ogni istanza di associazione in EFFETTUA_MANUTENZIONE collega un impiegato a una nota di servizio [SERVIZIO]. Ogni aeroplano è sottoposto molte volte al servizio di manutenzione; perciò è collegato tramite [SERVIZIO_AEROPLANO] a un certo numero di note di servizio. Una nota di servizio comprende come attributi la data di manutenzione [Data], il numero di ore dedicate al lavoro [Ore] e il tipo di lavoro svolto [CodiceLavoro]. Usiamo un tipo di entità debole [SERVIZIO_AEROPLANO] per rappresentare il servizio su un aeroplano, perché il numero di matricola dell'aeroplano è usato per identificare una nota di servizio. Un proprietario può essere o una persona o un'azienda. Perciò usiamo una categoria unione [PROPRIETARIO] che è un sottoinsieme dell'unione dei tipi di entità azienda [AZIENDA] e persona [PERSONA]. Sia i piloti [PILOTA] sia gli impiegati [IMPIEGATO] sono sottoclassi di PERSONA. Ogni pilota ha come attributi specifici numero di licenza [Num-Lic] e limitazioni [Limit]; ogni impiegato ha come attributi specifici stipendio [Stipendio] e turno lavorato [Turno]. Per tutte le entità persona nella base di dati vengono memorizzati i dati sul rispettivo numero di previdenza sociale [Ssn], nome [Nome], indirizzo [Indirizzo] e num-

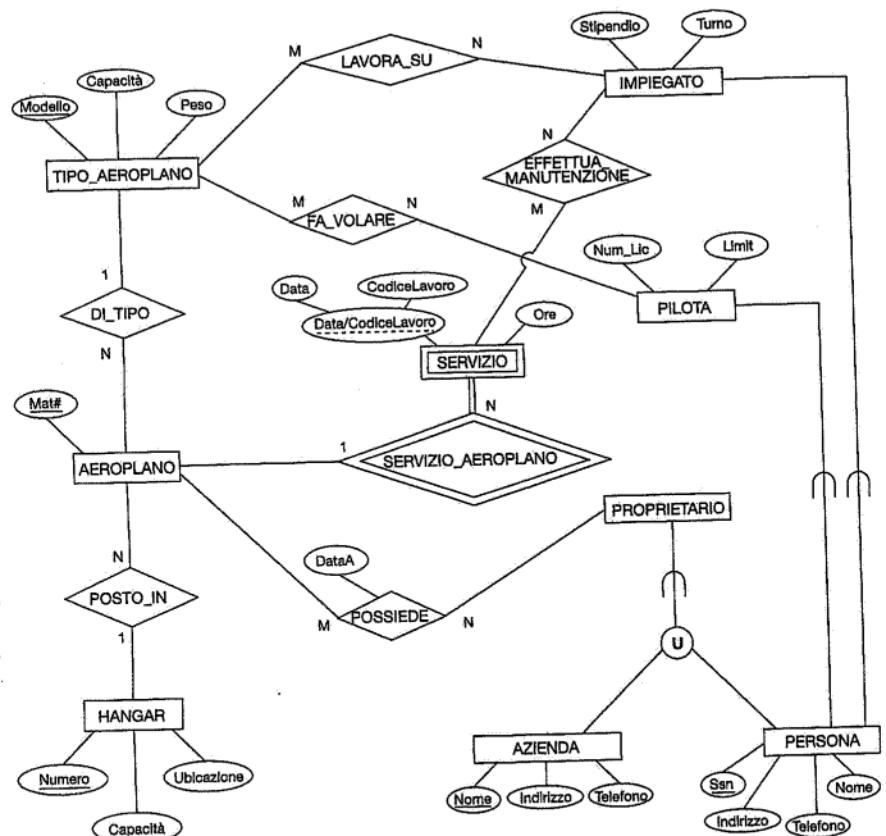


Figura 4.17 Schema EER per una base di dati PICCOLO AEROPORTO.

telefonico [Telefono]. Per le entità azienda i dati memorizzati comprendono nome [Nome], indirizzo [Indirizzo] e numero telefonico [Telefono]. La base di dati tiene anche traccia dei tipi di aeroplani che ciascun pilota è autorizzato a far volare [FA_VOLARE] e dei tipi di aeroplani su cui ciascun impiegato può fare un lavoro di manutenzione [LAVORA_SU]. Si illustri come lo schema EER PICCOLO AEROPORTO di Figura 4.17 possa essere rappresentato con notazione UML. (Nota: Non essendo stato qui esaminato come rappresentare le categorie [tipi unione] in UML, in questa domanda e nella successiva non è richiesta la trasformazione delle categorie).

- 4.22 Si mostri come lo schema EER UNIVERSITÀ di Figura 4.10 possa essere rappresentato con la notazione UML.

Bibliografia selezionata

Molti articoli hanno proposto modelli di dati concettuali o semantici. Qui se ne fornisce un elenco rappresentativo. Un gruppo di articoli, fra cui Abrial (1974), il modello DIAM di Senko (1975), il metodo NIAM (Verheijen e VanBekkum 1982), e Bracchi e altri (1976), presenta modelli semantici basati sul concetto di associazione binaria. Un altro gruppo di articoli, fra i primi scritti sull'argomento, esamina metodi per estendere il modello relazionale in modo da aumentare le sue potenzialità di modellazione. Questo gruppo comprende gli articoli di Schmid e Swenson (1975), Navathe e Schkolnick (1978), il modello RM/T di Codd (1979), Furtado (1978), e il modello strutturale di Wiederhold e Elmasri (1979).

Il modello ER è stato proposto originariamente da Chen (1976) ed è stato formalizzato in Ng (1981). Da allora sono state proposte numerose estensioni delle sue potenzialità di modellazione, come in Scheuermann e altri (1979), Dos Santos e altri (1979), Teorey e altri (1986), Gogolla e Hohenstein (1991), e il modello Entità-Categoria-Associazione (ECR) di Elmasri e altri (1985). Smith e Smith (1977) presentano i concetti di generalizzazione e aggregazione. Il modello semantico di dati di Hammer e McLeod (1981) ha introdotto il concetto di reticolo di classi/sottoclassi, così come altri concetti di modellazione avanzata.

Uno sguardo generale sulla modellazione semantica di dati è presente in Hull e King (1987). Eick (1991) esamina la progettazione e le trasformazioni di schemi concettuali. In Soutou (1998) è fornita un'analisi dei vincoli per associazioni *n*-arie. L'UML è descritto in dettaglio in Booch, Rumbaugh e Jacobson (1999).

Capitolo 5

Memorizzazione dei record e organizzazioni primarie dei file

Le basi di dati sono fisicamente memorizzate come file di record, che a loro volta sono tipicamente memorizzati su dischi magnetici. In questo capitolo e nel successivo tratteremo l'organizzazione fisica delle basi di dati e le tecniche per accedere efficientemente ad esse usando vari algoritmi, alcuni dei quali richiedono strutture dati ausiliarie dette indici. Cominceremo nel Paragrafo 5.1 introducendo i concetti di gerarchie di memorizzazione nel computer e vedendo come essi vengono usati nei sistemi di basi di dati. Nel Paragrafo 5.2 descriveremo i dispositivi di memorizzazione a dischi magnetici, con le loro caratteristiche, e i dispositivi di memorizzazione a nastri magnetici, mentre nel Paragrafo 5.3 un sistema alternativo di memorizzazione dati più recente detto RAID (Redundant Arrays of Inexpensive [or Independent] Disks: vettori ridondanti di dischi economici [o indipendenti]), che fornisce una maggiore affidabilità e migliori prestazioni. Volgeremo poi l'attenzione ai metodi per organizzare i dati su dischi: nel Paragrafo 5.4 tratteremo la tecnica della doppia bufferizzazione, che è usata per accelerare il recupero di più blocchi di disco, nel Paragrafo 5.5 modi diversi di formattazione e memorizzazione dei record di un file su disco e nel Paragrafo 5.6 i vari tipi di operazioni che vengono tipicamente eseguite sui record di un file. Presenteremo poi tre metodi primari di organizzazione dei record di un file su disco: record non ordinati nel Paragrafo 5.7, record ordinati nel Paragrafo 5.8 e record hash nel Paragrafo 5.9.

Nel Paragrafo 5.10 esamineremo molto concisamente i file di record misti e altri metodi primari per organizzare i record, come gli alberi B. Questi sono particolarmente importanti per la memorizzazione di basi di dati a oggetti. Nel Capitolo 6 esamineremo le tecniche per creare strutture dati ausiliarie, dette indici, che velocizzano la ricerca e il recupero dei record e che prevedono la memorizzazione di dati ausiliari, detti file di indici, in aggiunta ai record del file stessi.

I lettori che hanno già studiato le organizzazioni dei file possono limitarsi a dare una rapida scorsa ai Capitoli 5 e 6, o addirittura passare oltre. La loro lettura può essere anche rimandata a dopo aver studiato il modello relazionale.

5.1 Introduzione

La collezione di dati che costituisce una base di dati computerizzata deve essere fisicamente memorizzata su un qualche **supporto di memoria** del calcolatore. Il software del DBMS potrà poi recuperare, aggiornare ed elaborare questi dati quando necessario. I supporti di memoria del calcolatore formano una *gerarchia di memoria* che comprende due categorie fondamentali.

- **Memoria principale.** Questa categoria comprende supporti di memoria su cui può operare direttamente l'*unità centrale di elaborazione* (CPU: *central processing unit*) del calcolatore, come la memoria centrale del calcolatore e le meno capienti ma più veloci memorie cache. La memoria principale di solito consente un rapido accesso ai dati ma ha una limitata capacità di memorizzazione.
- **Memoria secondaria.** Questa categoria comprende dischi magnetici, dischi ottici e nastri. Questi dispositivi hanno solitamente una maggiore capacità, costano meno ma forniscono un accesso più lento ai dati rispetto ai dispositivi di memorizzazione principale. I dati nella memoria secondaria non possono essere elaborati direttamente dalla CPU, ma devono essere prima copiati nella memoria principale.

Si fornirà qui innanzitutto una visione d'insieme dei vari dispositivi di memorizzazione usati per la memoria principale e secondaria, per poi passare a esaminare come sono tipicamente gestite le basi di dati nella gerarchia di memoria.

5.1.1 Gerarchie di memoria e dispositivi di memorizzazione

In un moderno sistema di elaborazione i dati risiedono in e vengono trasferiti fra supporti di memorizzazione organizzati in gerarchia. La memoria più veloce è anche la più costosa ed è pertanto disponibile con minore capacità; la memoria meno veloce è costituita dai nastri magnetici ed è sostanzialmente disponibile con capacità indefinita.

A *livello di memoria principale* la **memoria cache**, costituita da una RAM statica (RAM: Random Access Memory: memoria ad accesso casuale) è la più costosa. La memoria cache viene tipicamente usata dalla CPU per accelerare l'esecuzione dei programmi. Il livello successivo di memoria principale è costituita dalla DRAM (Dynamic RAM: RAM dinamica), che fornisce l'area di lavoro principale della CPU per memorizzare programmi e dati ed è comunemente detta **memoria centrale (main memory)**. Il vantaggio della DRAM consiste nel suo basso costo, che continua a diminuire; lo svantaggio è la sua volatilità¹ e la minore velocità se confrontata con la RAM statica. A *livello di memorizzazione secondaria* la gerarchia comprende dischi magnetici come pure **memoria di massa** nella forma di dispositivi CD-ROM

¹ La memoria volatile perde il suo contenuto nel caso di un'interruzione dell'alimentazione, al contrario della memoria non volatile.

(Compact Disk – Read-Only Memory: compact disk – memoria a sola lettura), e infine nastri, i meno costosi. La **capacità di memoria** è misurata in kilobyte (Kbyte o 2^{10} byte), megabyte (Mbyte o 2^{20} byte), gigabyte (Gbyte o 2^{30} byte) e anche terabyte (2^{40} byte).

I programmi risiedono e vengono eseguiti nella DRAM. In genere basi di dati permanenti di grandi dimensioni risiedono in memoria secondaria, e quando necessario porzioni della base di dati sono poste in, e trascritte da, buffer di memoria centrale. Ora che i personal computer e le workstation hanno decine di megabyte di dati in DRAM sta diventando possibile caricare un'ampia frazione della base di dati in memoria centrale. In alcuni casi intere basi di dati possono essere memorizzate in memoria centrale (con una copia di backup su disco magnetico), portando a **basi di dati in memoria centrale**; esse sono particolarmente utili in applicazioni in tempo reale che richiedono tempi di risposta estremamente brevi. Un esempio è costituito dalle applicazioni di commutazione telefonica, che tengono in memoria centrale basi di dati che contengono informazioni di instradamento (routing) e di linea.

Tra la memoria DRAM e il disco magnetico, un'altra forma di memoria, la **memoria flash**, sta diventando comune, in particolare perché non è volatile. Le memorie flash sono memorie ad alta densità e ad alte prestazioni che usano la tecnologia EEPROM (Electrically Erasable Programmable Read-Only Memory: memoria a sola lettura programmabile cancellabile elettricamente). Il vantaggio della memoria flash risiede nell'elevata velocità di accesso; lo svantaggio nel fatto che deve essere cancellato e riscritto un intero blocco alla volta.²

I CD-ROM memorizzano i dati otticamente e sono letti da un laser; contengono dati pre-registrati che non possono essere sovrascritti. I dischi WORM (Write-Once-Read-Many: scrivi una volta e leggi più volte) costituiscono una forma di memorizzazione ottica usata per archiviare dati: essi consentono che i dati vengano scritti una volta e letti un numero qualsiasi di volte senza avere la possibilità di cancellarli. Memorizzano circa mezzo gigabyte di dati per disco e durano molto più a lungo dei dischi magnetici. I **juke-box di memorie ottiche** usano una schiera di CD-ROM, che vengono caricati nei drive (unità di lettura) su richiesta. Anche se i juke-box ottici hanno capacità dell'ordine di centinaia di gigabyte, i loro tempi di recupero sono dell'ordine di centinaia di millisecondi, e sono quindi sensibilmente più lenti dei dischi magnetici.³ Questo tipo di memoria non è diventato così popolare come ci si aspettava a causa della rapida diminuzione di costo e del rapido aumento di capacità dei dischi magnetici. Il DVD (Digital Video Disk: video disco digitale) è uno standard recente per i dischi ottici che consente da quattro a quindici gigabyte di memoria per disco.

Infine, i **nastri magnetici** sono usati per archiviare e salvare grandi quantità di dati. I **juke-box di nastri** – contenenti una banca di nastri che sono catalogati e che possono essere caricati automaticamente nei drive per i nastri – stanno diventando popolari come **memoria terziaria** per conservare terabyte di dati. Ad esempio il sistema EOS (Earth Observation Satellite: satellite di osservazione terrestre) della NASA memorizza in questo modo basi di dati d'archivio.

Si prevede che nel giro di pochi anni molte organizzazioni di grandi dimensioni considereranno normale avere basi di dati delle dimensioni di terabyte. La locuzione **base di dati**

² Ad esempio l'INTEL DD28F032SA è una memoria flash con una capacità di 32 megabit, con una velocità di accesso di 70 nanosecondi e tasso di trasferimento in scrittura di 430 KB/secondo.

³ Le loro velocità di rotazione sono minori (circa 400 rpm [revolutions per minute: giri al minuto]) e ciò dà luogo a maggiori ritardi di latenza e bassi tassi di trasferimento (tra 100 e 200 KB al secondo).

molto ampia (*very large database*) non può più essere definita precisamente, perché le capacità della memorizzazione su disco stanno aumentando e i costi stanno diminuendo; potrebbe perciò essere ben presto riservata alle basi di dati che contengono decine di terabyte.

5.1.2 Memorizzazione di basi di dati

Le basi di dati tipicamente memorizzano grandi quantità di dati, i quali devono *persistere* per lunghi periodi di tempo. Durante questi periodi si accede ai dati e li si elabora ripetutamente. Ciò è in contrasto con la nozione di strutture dati *transitorie* che persistono solo per un tempo limitato durante l'esecuzione del programma. La maggior parte delle basi di dati è memorizzata permanentemente (o *persistentemente*) su memoria secondaria a disco magnetico, per le seguenti ragioni:

- di solito le basi di dati sono troppo ampie per poter essere interamente contenute in memoria centrale;
- le circostanze che causano una perdita permanente di dati memorizzati si verificano meno frequentemente per la memorizzazione secondaria su disco che per la memorizzazione primaria; per questo motivo ci si riferisce al disco – e ad altri dispositivi di memoria secondaria – come a **memoria non volatile**, mentre la memoria centrale è spesso chiamata **memoria volatile**;
- il costo di memorizzazione per unità di dato è di un ordine di grandezza inferiore per il disco che per la memoria principale.

È verosimile che alcune delle più recenti tecnologie – come i dischi ottici, i DVD e i jukebox di nastri – forniranno valide alternative all'uso dei dischi magnetici. Inoltre in futuro le basi di dati potranno risiedere, nella gerarchia di memoria, a livelli diversi rispetto a quelli descritti nel Sottoparagrafo 5.1.1. Per il momento, tuttavia, è importante studiare e capire le proprietà e le caratteristiche dei dischi magnetici e il modo in cui i file di dati possono essere organizzati su disco, per progettare basi di dati reali con prestazioni accettabili.

I nastri magnetici sono frequentemente usati come supporto di memoria per fare una copia di backup della base di dati, dato che la memorizzazione su nastro costa ancor meno della memorizzazione su disco. Però l'accesso ai dati su nastro è piuttosto lento. I dati memorizzati su nastro sono **off-line** (non in linea); cioè è necessario l'intervento di un operatore – o un dispositivo automatico di caricamento – per caricare un nastro prima che questi dati si rendano effettivamente disponibili. Al contrario, i dischi sono dispositivi **on-line** (in linea) a cui si può accedere direttamente in ogni momento.

Le tecniche usate per memorizzare grandi quantità di dati strutturati su disco sono importanti per i progettisti di una base di dati, il DBA e gli implementatori di un DBMS. I progettisti della base di dati e il DBA devono conoscere i vantaggi e gli svantaggi di ciascuna tecnica di memorizzazione quando progettano, implementano e gestiscono una base di dati su uno specifico DBMS. Di solito il DBMS fornisce molte opzioni per organizzare i dati, e il processo di **progettazione fisica di una base di dati** prevede la scelta, fra le opzioni, delle particolari tecniche di organizzazione dei dati che meglio si adattano ai requisiti propri dell'applicazione. Gli implementatori di sistemi DBMS devono studiare le tecniche di organizzazione

ne dei dati per poterle implementare efficientemente e perciò fornire a DBA e utenti un numero sufficiente di opzioni.

Le applicazioni tipiche di basi di dati hanno bisogno per l'elaborazione solo di una piccola porzione alla volta della base di dati. Tutte le volte che c'è bisogno di una certa porzione dei dati, essa deve essere localizzata su disco, copiata in memoria centrale per l'elaborazione, e quindi riscritta su disco se i dati sono cambiati. I dati memorizzati su disco sono organizzati come **file di record**. Ogni record è costituito da una collezione di valori di dati che possono essere interpretati come fatti relativi alle entità, ai loro attributi e alle loro associazioni. I record dovrebbero essere memorizzati su disco in modo da rendere possibile individuarne efficientemente la collocazione ogni volta che se ne ha bisogno.

Ci sono molte **organizzazioni primarie dei file** (si vedano i Paragrafi 5.7-5.10) che determinano come sono *collocati fisicamente* sul disco i record di un file, e perciò come si può accedere ad essi. Un **file heap** (heap: ammasso, cumulo) (o **file non ordinato**) colloca i record su disco senza un ordine particolare, semplicemente aggiungendo nuovi record alla fine del file, mentre un **file ordinato** (o **file sequenziale**) tiene i record in ordine sulla base del valore di un campo particolare (detto chiave di ordinamento). Un **file hash** (to hash: tritare, sminuzzare) usa una funzione hash applicata a un campo particolare (detto chiave hash) per determinare la collocazione di un record su disco. Altre organizzazioni primarie dei file, come gli **alberi B**, usano per l'appunto strutture ad albero. Un'**organizzazione secondaria**, o **struttura di accesso ausiliaria**, consente un accesso efficiente ai record di un file basata su **campi alternativi** rispetto a quelli che sono stati usati per l'organizzazione primaria dei file. La maggior parte di questi esistono come indici e saranno studiati nel Capitolo 6.

5.2 Dispositivi di memoria secondaria

In questo paragrafo verranno descritte alcune caratteristiche dei dischi e dei nastri magnetici. I lettori che hanno già studiato questi dispositivi possono limitarsi a una rapida lettura.

5.2.1 Descrizione dell'hardware di dispositivi a disco

I dischi magnetici sono usati per memorizzare grandi quantità di dati. La più elementare unità dati su disco è il singolo **bit** di informazione. Magnetizzando un'area su disco in certi modi, si può far sì che essa rappresenti un valore di bit di 0 (zero) o 1 (uno). Per codificare le informazioni, i bit sono raggruppati in **byte** (o **caratteri**). Le dimensioni dei byte vanno da 4 a 8 bit, a seconda del computer e del dispositivo. Noi supporremo che un carattere sia memorizzato in un singolo byte, e useremo i termini **byte** e **carattere** indifferentemente. Per **capacità** di un disco si intende il numero di byte che esso può memorizzare, che di solito è molto grande. I piccoli floppy disk usati nei microcomputer tipicamente contengono da 400 Kbyte a 1,5 Mbyte; i dischi rigidi per questi elaboratori tipicamente contengono da molte centinaia di Mbyte a qualche Gbyte; e le grosse pile di dischi (*disk packs*) usate nei minicomputer e nei

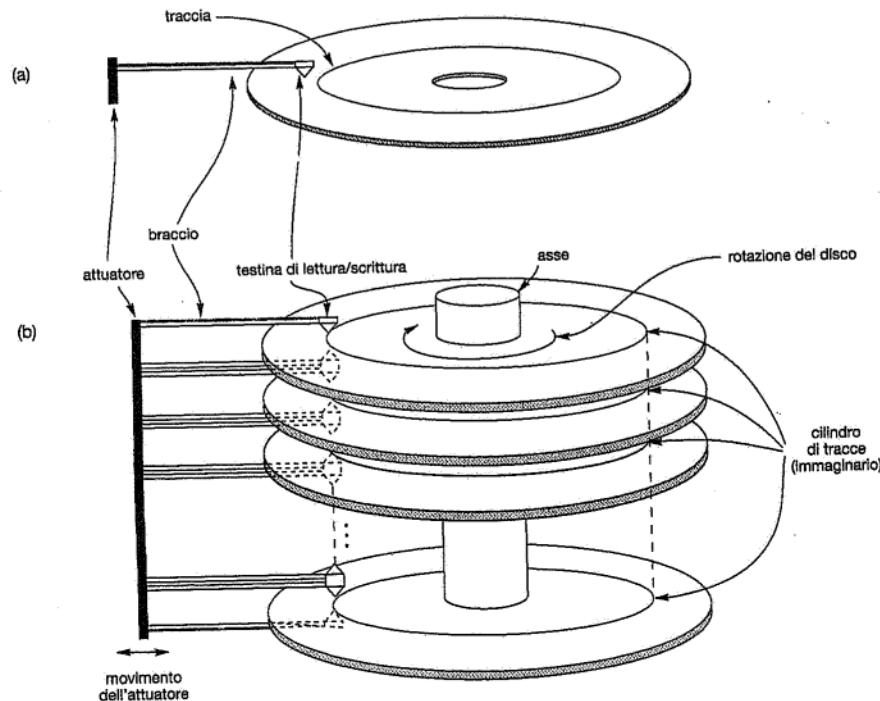


Figura 5.1 (a) Un disco a singola faccia con l'hardware di lettura/scrittura. (b) Una pila di dischi con l'hardware di lettura/scrittura.

mainframe hanno capacità che vanno da qualche decina a centinaia di Gbyte. Le capacità dei dischi continuano ad aumentare man mano che la tecnologia migliora.

Qualunque sia la loro capacità, i dischi sono tutti fatti di materiale magnetico della forma di sottile disco circolare (Figura 5.1a) e protetto tramite una copertura di plastica o di acrilico. Un disco è a singola faccia (single-sided) se memorizza informazioni su uno solo dei suoi lati e a doppia faccia (double-sided) se sono usati entrambi i lati. Per aumentare la capacità di memorizzazione i dischi sono assemblati in una pila di dischi (Figura 5.1b), che può comprendere molti dischi e perciò molti lati. L'informazione è memorizzata sulla faccia di un disco su circonferenze concentriche di piccola larghezza,⁴ ciascuna delle quali ha un diverso diametro. Ogni circonferenza è detta traccia (track). Per le pile di dischi le tracce con lo stesso diametro sulle varie facce sono dette cilindro per la struttura che formerebbero se connesso.

⁴ In alcuni dischi le circonferenze sono ora collegate, in modo da formare una specie di spirale continua.

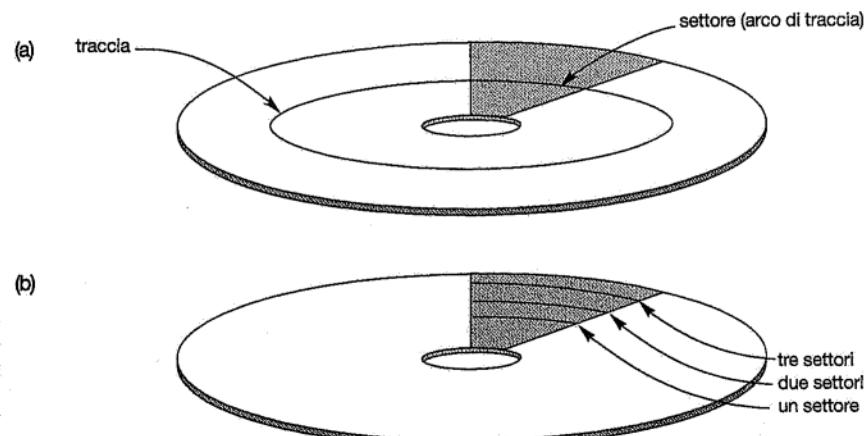


Figura 5.2 Diverse organizzazioni a settori di disco. (a) Settori che sottendono un angolo fisso. (b) Settori che mantengono una densità di registrazione uniforme.

se nello spazio. Il concetto di cilindro è importante perché i dati memorizzati in un cilindro possono essere recuperati molto più velocemente che se fossero distribuiti tra cilindri diversi.

Il numero di tracce per un disco va da qualche centinaio a qualche migliaio, e la capacità di ciascuna traccia va tipicamente dalle decine di Kbyte fino a 150 Kbyte. Dato che una traccia di solito contiene una gran quantità di informazione, essa viene suddivisa in blocchi o settori più piccoli. La suddivisione di una traccia in settori è codificata nell'hardware sulla faccia del disco e non può essere cambiata. Un tipo di organizzazione a settori chiama settore una porzione di traccia che sottende un fissato angolo al centro (Figura 5.2a). Sono possibili molte altre organizzazioni a settori, una delle quali consiste nell'avere settori che sottendono angoli al centro più piccoli man mano che ci si allontana dal centro stesso, mantenendo così una densità di memorizzazione costante (Figura 5.2b). Non tutti i dischi hanno le tracce suddivise in settori.

La divisione di una traccia in blocchi di disco (o pagine) di uguale dimensione è fissata dal sistema operativo durante la formattazione (o inizializzazione) del disco. La dimensione di un blocco è fissata durante l'inizializzazione e non può essere cambiata dinamicamente. Tipiche dimensioni di un blocco di disco vanno da 512 a 4096 byte. Un disco con settori codificati nell'hardware ha spesso i settori che vengono suddivisi in blocchi durante l'inizializzazione. I blocchi sono separati da spazi tra blocchi (interblock gaps) di dimensioni fisse, che comprendono informazioni di controllo codificate appositamente, scritte durante l'inizializzazione del disco. Queste informazioni sono usate per determinare quale blocco della traccia segue ciascuno spazio tra blocchi. In Tabella 5.1 sono rappresentate le specifiche di un disco tipico.

C'è un continuo miglioramento nella capacità di memorizzazione e nei tassi di trasferimento associati ai dischi; la memorizzazione su disco sta anche diventando via via più eco-

Tabella 5.1 Specifiche dei tipici dischi di fascia alta Cheetah, prodotti dalla Seagate.

Descrizione	ST136403LC	ST318203LC
Codice modello	Cheetah 36	Cheetah 18LP
Nome modello	3,5 pollici	3,5 pollici
Fattore di forma (larghezza)	1,04 kg	0,59 kg
Peso		
Capacità/Interfaccia		
Capacità, formattato	36,4 Gbyte, formattato	18,2 Gbyte, formattato
Tipo di interfaccia	80-pin Ultra-2 SCSI	80-pin Ultra-2 SCSI
Configurazione		
Numeri di dischi (fisici)	12	6
Numeri di testine (fisiche)	24	12
Cilindri totali (solo SCSI)	9772	9801
Tracce totali (solo SCSI)	N/D	117612
Byte per settore	512	512
Densità delle tracce (TPI)	N/D tracce/pollice	12580 tracce/pollice
Densità di registrazione (BPI, max)	N/D bit/pollice	258048 bit/pollice
Prestazioni		
Tassi di trasferimento		
Tasso di trasferimento interno (min)	193 Mbit/sec	193 Mbit/sec
Tasso di trasferimento interno (max)	308 Mbit/sec	308 Mbit/sec
Tasso di trasferimento int., formattato (min)	18 Mbit/sec	18 Mbit/sec
Tasso di trasferimento int., formattato (max)	28 Mbit/sec	28 Mbit/sec
Tasso di trasferimento esterno (I/O) (max)	80 Mbit/sec	80 Mbit/sec
Tempi di posizionamento		
Tempo di posizionamento medio, lettura	5,7 msec, tipico	5,2 msec, tipico
Tempo di posizionamento medio, scrittura	6,5 msec, tipico	6 msec, tipico
Posizionamento da traccia a traccia, lettura	0,6 msec, tipico	0,6 msec, tipico
Posizionamento da traccia a traccia, scrittura	0,9 msec, tipico	0,9 msec, tipico
Posizionamento su traccia esterna, lettura	12 msec, tipico	12 msec, tipico
Posizionamento su traccia esterna, scrittura	13 msec, tipico	13 msec, tipico
Latenza media	2,99 msec	2,99 msec
Altre		
Dimensione di default del buffer (cache)	1024 Kbyte	1024 Kbyte
Velocità di rotazione dell'asse	10000 RPM	10016 RPM
Tasso di errore non riparabile	1 ogni 10^{15} bit letti	1 ogni 10^{15} bit letti
Errori di posizionamento (SCSI)	1 ogni 10^8 bit letti	1 ogni 10^8 bit letti

Per cortesia della Seagate Technology © 1999.

nomica – costando attualmente solo una frazione di dollaro per megabyte. I costi stanno diminuendo così rapidamente che si prevedono costi dell'ordine di un centesimo di dollaro per megabyte ossia 10.000 dollari per terabyte entro la fine del 2001.

Un disco è un dispositivo indirizzabile *ad accesso casuale*. Il trasferimento dati tra la memoria centrale e il disco avviene in unità di blocchi di disco. L'**indirizzo hardware** di un blocco – una combinazione di numero di faccia, numero di traccia (nella faccia) e numero di blocco (nella traccia) – è fornito all'hardware dell'input/output (I/O) del disco. Viene anche fornito l'**indirizzo di un buffer** – un'area riservata di locazioni contigue in memoria centrale che può contenere un blocco. Con un comando **read** il blocco dal disco è copiato nel buffer, mentre con un comando **write** il contenuto del buffer è copiato nel blocco del disco. Talora possono essere trasferiti come una cosa sola molti blocchi contigui, detti **cluster** (grappolo). In questo caso la dimensione del buffer è adattata per corrispondere al numero di byte presenti nel

L'effettivo meccanismo hardware che legge o scrive un blocco è la **testina di lettura/scrittura**, che fa parte di un sistema detto **unità disco (disk drive)**. Un disco o una pila di dischi sono inseriti nell'unità disco, che ha un motore che fa ruotare i dischi. Una testina di lettura/scrittura è costituita da un componente elettronico fissato a un **braccio meccanico**. Le pile di dischi con più facce sono controllate da molte testine di lettura/scrittura – una per ogni faccia (Figura 5.1b). Tutti i bracci sono collegati a un **attuatore** unito a un altro motore elettrico, che muove le testine di lettura/scrittura all'unisono e le posiziona esattamente sopra il cilindro di tracce specificato nell'indirizzo di un blocco.

Le unità disco per i dischi rigidi fanno ruotare la pila di dischi ininterrottamente a una velocità costante (compresa tipicamente tra 3600 e 7200 rpm). Per un floppy disk l'unità disco inizia a far ruotare il disco ogni volta che viene dato inizio a una particolare richiesta di lettura o scrittura e cessa la rotazione subito dopo che è stato completato il trasferimento dati. Una volta che la testina di lettura/scrittura è stata posizionata sulla traccia corretta e il blocco specificato nell'indirizzo di blocco si muove sotto di essa, il componente elettronico della testina di lettura/scrittura viene attivato per trasferire i dati. Alcune unità a disco hanno testine di lettura/scrittura fisse, con tante testine quante sono le tracce. Queste unità sono dette **dischi a testina fissa**, mentre le unità a disco con un attuatore sono dette **dischi a testina mobile**. Per i dischi a testina fissa, una traccia o cilindro è selezionata tramite la commutazione elettronica all'appropriata testina di lettura/scrittura piuttosto che tramite un effettivo movimento meccanico; di conseguenza questa unità è molto più veloce. Tuttavia, il costo delle testine di lettura/scrittura aggiuntive è piuttosto alto, e pertanto i dischi a testina fissa non sono usati comunque.

Un **disk controller** (controllore di disco), tipicamente inserito nell'unità disco, controlla l'unità disco e la interfaccia al sistema di elaborazione. Una delle interfacce standard usate oggi per le unità disco nei PC e nelle workstation è detta **SCSI** (Small Computer Storage Interface: interfaccia di memoria per piccoli computer). Il controller accetta comandi di I/O di alto livello e intraprende azioni appropriate per posizionare il braccio, facendo sì che abbia luogo l'azione di lettura/scrittura. Per trasferire un blocco di disco, dato il suo indirizzo, il controller deve prima di tutto posizionare meccanicamente la testina di lettura/scrittura sulla traccia corretta. Il tempo necessario per fare ciò è detto **tempo di posizionamento (seek time)**. Tempi di posizionamento tipici sono di 12-14 msec per i personal computer e di 8-9 msec per i server. Segue un ritardo – detto **ritardo di rotazione o latenza (latency)** – per attendere che l'inizio del blocco desiderato ruoti fino alla posizione sotto la testina di lettura/scrittura. Infine, c'è bisogno di un tempo aggiuntivo per trasferire i dati, detto **tempo di trasferimento di blocco**. Perciò il tempo totale necessario per localizzare e trasferire un blocco arbitrario, dato il suo indirizzo, è costituito dalla somma del tempo di posizionamento, ritardo di rotazione e tempo di trasferimento di blocco. Il tempo di posizionamento e il ritardo di rotazione sono solitamente molto più grandi del tempo di trasferimento di blocco. Per rendere più efficiente il trasferimento di più blocchi, è comune trasferire molti blocchi consecutivi sulla stessa traccia o cilindro. Ciò elimina il tempo di posizionamento e il ritardo di rotazione per tutti i blocchi tranne il primo e può avere come risultato un sostanziale risparmio di tempo quando vengono trasferiti numerosi blocchi contigui. Di solito i costruttori di dischi forniscono un **tasso di trasferimento di una mole di dati (bulk transfer rate)** per calcolare il tempo richiesto per trasferire blocchi consecutivi (si veda l'Appendice B).

Il tempo necessario per localizzare e trasferire un blocco di disco è dell'ordine dei millisecondi, andando di solito dai 12 ai 60 msec. Per blocchi contigui la localizzazione del primo blocco richiede da 12 a 60 msec, ma il trasferimento dei blocchi successivi può richiedere solo da 1 a 2 msec ciascuno. Molte tecniche di ricerca traggono profitto dal recupero di blocchi consecutivi quando cercano dati su disco. In ogni caso un tempo di trasferimento dell'ordine dei millisecondi è considerato piuttosto alto se confrontato con il tempo richiesto per elaborare dati in memoria centrale dalle CPU attuali. Perciò la localizzazione di dati su disco è un *collo di bottiglia importante* nelle applicazioni di basi di dati. Le strutture di file che esamineremo qui e nel Capitolo 6 tentano di *ridurre al minimo il numero di trasferimenti di blocchi* necessari per localizzare e trasferire i dati richiesti da disco a memoria centrale.

5.2.2 Dispositivi di memorizzazione a nastro magnetico

I dischi sono dispositivi di memoria secondaria ad accesso casuale, perché si può accedere "a caso" a un arbitrario blocco di disco, una volta che si è specificato il suo indirizzo. I nastri magnetici sono dispositivi ad accesso sequenziale: per accedere all' n -esimo blocco sul nastro, occorre prima scandire gli $n - 1$ blocchi precedenti. I dati sono memorizzati su bobine di nastro magnetico di elevata capacità, molto simili ai nastri audio o video. Si richiede a un'unità a nastro (tape drive) di leggere i dati da, o di scrivere i dati in, una bobina di nastro. Di solito ogni gruppo di bit che forma un byte è memorizzato sul nastro, e i byte sono memorizzati consecutivamente.

Per leggere o scrivere dati su nastro si usa una testina di lettura/scrittura. Anche i record di dati su nastro sono memorizzati in blocchi – sebbene i blocchi possano essere notevolmente più grandi di quelli per i dischi, e gli spazi tra blocchi siano pure piuttosto grandi. Con le densità tipiche per un nastro che vanno da 1600 a 6250 byte per pollice, un tipico spazio tra blocchi⁵ di 0,6 pollici corrisponde a uno spreco di spazio di memoria che va da 960 a 3750 byte. Per una migliore utilizzazione dello spazio è consuetudine raggruppare insieme in un solo blocco molti record.

La caratteristica principale di un nastro sta nel suo requisito di accedere ai blocchi di dati in ordine sequenziale. Per riuscire a raggiungere un blocco nel mezzo di una bobina di nastro, quest'ultimo viene inserito e quindi analizzato, fino a quando il blocco richiesto si trova sotto la testina di lettura/scrittura. Per questa ragione l'accesso al nastro può essere lento e i nastri non sono usati per memorizzare dati in linea, se non per alcune applicazioni specializzate. Però i nastri forniscono una funzione molto importante, quella di **backup** (salvataggio) della base di dati, che permette di tenere copie dei file presenti su disco, utili nel caso in cui i dati vengano persi a causa di una rottura di disco, che può verificarsi se la testina di lettura/scrittura tocca la superficie del disco per un malfunzionamento meccanico. Per questa ragione i file su disco sono copiati periodicamente su nastro. I nastri possono anche essere usati per memorizzare file troppo grandi della base di dati. Infine, file della base di dati usati ra-

ramente o superati, ma richiesti per mantenere una registrazione storica, possono essere archiviati su nastro. Recentemente stanno diventando popolari per il salvataggio di file di dati per le workstation e i personal computer nastri magnetici ridotti, di 8 mm (simili a quelli usati nelle videocamere), che possono memorizzare fino a 50 Gbyte, nonché le cartucce di dati a scansione elicoidale di 4 mm e i CD-ROM. Essi sono anche usati per memorizzare immagini e librerie di sistema.

5.3 Rendere parallelo l'accesso al disco attraverso l'uso della tecnologia RAID

Con la crescita esponenziale delle prestazioni e delle capacità dei dispositivi e delle memorie a semiconduttore, si stanno diffondendo microprocessori più veloci con memorie principali sempre più grandi. Per far fronte a questa crescita, è naturale aspettarsi che anche la tecnologia della memoria secondaria tenda ad adeguarsi, per prestazioni e affidabilità, alla tecnologia dei processori.

Un miglioramento importante nella tecnologia della memoria secondaria è rappresentato dallo sviluppo della tecnologia **RAID**, che originariamente stava per **Redundant Arrays of Inexpensive Disks** (vettori ridondanti di dischi economici). Più tardi la "I" di RAID è stata interpretata come **Independent** (indipendenti). L'idea RAID ha ricevuto un'adesione molto positiva dall'industria ed è stata sviluppata in un insieme variegato di architetture RAID alternative (livelli RAID da 0 a 6).

Lo scopo principale della tecnologia RAID è quello di uguagliare i tassi molto diversi di miglioramento delle prestazioni dei dischi rispetto a quelli della memoria principale e dei microprocessori.⁶ Mentre le capacità della RAM sono quadruplicate ogni due o tre anni, i tempi

Tabella 5.2 Tendenze nella tecnologia dei dischi.

Valori dei parametri nel 1993*	Tasso storico di miglioramento annuo (%)*	Valori previsti per il 1999**
Densità areale	50-150 Mbit/pollice quadrato	27
Densità lineare	40000-60000 bit/pollice	13
Densità inter-traccia	1500-3000 tracce/pollice	10
Capacità (fattore di forma: 3,5")	100-2000 MB	27
Tasso di trasferimento	3-4 MB/s	22
Tempo di posizionamento	7-20 ms	8
		2-3 GB/pollice quadrato 238 Kbit/pollice 11550 tracce/pollice 36 GB 17-28 MB/sec 5-7 msec

* Fonte: Chen, Lee, Gibson, Katz e Patterson (1994). Riproduzione autorizzata.

** Fonte: Unità a disco rigido IBM Ultrastar 36XP e 18ZX.

⁶ Gordon Bell ha previsto che questi fossero circa del 40 per cento ogni anno tra il 1974 e il 1984, e ora si suppone che superino il 50 per cento annuo.

⁵ Nella terminologia propria dei nastri si parla di *spazi tra record* (interrecord gaps).

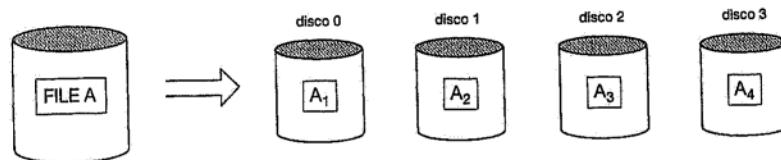


Figura 5.3 Data striping. Il file A è suddiviso (*striped*) su quattro dischi.

di accesso al disco stanno migliorando meno del 10 per cento all'anno, e i *tassi di trasferimento* stanno aumentando approssimativamente del 20 per cento annuo. In verità le *capacità* dei dischi stanno aumentando più del 50 per cento all'anno, ma i miglioramenti nella velocità e nel tempo di accesso sono di entità molto inferiore. In Tabella 5.2 sono illustrate le tendenze nella tecnologia dei dischi in termini dei valori dei parametri del 1993 e dei tassi di miglioramento.

Esiste poi una seconda differenza qualitativa tra la capacità di microprocessori speciali che si rivolgono a nuove applicazioni che riguardano l'elaborazione di dati video, audio, di immagini e spaziali e la corrispondente mancanza di accesso rapido a grandi insiemi di dati condivisi.

La soluzione naturale consiste in un grande vettore di piccoli dischi indipendenti che si comporta come un singolo disco logico di maggiori prestazioni. Viene usato un concetto detto **data striping** (suddivisione dei dati), che utilizza il *parallelismo* per incrementare le prestazioni del disco. Il data striping distribuisce i dati in modo trasparente fra più dischi, così che essi si comportino come un unico disco, grande e veloce. In Figura 5.3 è rappresentato un file distribuito o *suddiviso (striped)* fra quattro dischi. La suddivisione migliora le prestazioni di I/O complessive consentendo che I/O multipli vengano serviti in parallelo, fornendo così alti tassi di trasferimento complessivi. La suddivisione dei dati realizza anche un bilanciamento del carico fra i dischi. Inoltre, memorizzando informazioni ridondanti sui dischi con l'uso della parità o di qualche altro codice a correzione d'errore, può essere migliorata l'affidabilità.

5.3.1 Miglioramento dell'affidabilità

Per un vettore di n dischi la frequenza di un guasto è n volte quella che si ha per un solo disco. Perciò, se l'MTTF (Mean Time To Failure: tempo medio perché si verifichi un guasto) di un'unità disco è considerato di 200.000 ore, ossia circa 22,8 anni (tempi tipici vanno fino a circa 1 milione di ore), quello di un banco di 100 unità disco diventa solo di 2000 ore, ossia 83,3 giorni. Mantenere una sola copia di dati in un tale vettore di dischi causerà una significativa perdita di affidabilità. Un'ovvia soluzione è quella di servirsi di una certa ridondanza dei dati in modo tale che i guasti ai dischi possano essere tollerati. Gli svantaggi sono molti: operazioni di I/O aggiuntive per la scrittura, calcoli supplementari per mantenere la ridondanza e per effettuare il ripristino dagli errori, e capacità di disco aggiuntiva per memorizzare i dati.

Una tecnica per introdurre ridondanza è detta **mirroring** (*to mirror*: riflettere) o **shadowing** (*to shadow*: seguire come un'ombra). I dati sono scritti con ridondanza in due dischi fisici identici, che sono trattati come un solo disco logico. Quando i dati vengono letti, essi possono essere recuperati dal disco con minori ritardi di accodamento, di posizionamento e di rotazione. Se un disco si guasta, l'altro disco viene usato finché il primo non è stato riparato. Si supponga che il tempo medio per la riparazione sia di 24 ore; allora il tempo medio per la perdita di dati in un sistema a disco riflesso (*mirrored disk system*) che usa 100 dischi con MTTF di 200.000 ore ciascuno è di $(200.000)^2/(2 \cdot 24) = 8,33 \cdot 10^8$ ore, cioè di 95.028 anni.⁷ La riflessione di disco raddoppia anche il tasso con il quale sono gestite le richieste di lettura, dal momento che una lettura può essere indirizzata all'uno o all'altro disco. Il tasso di trasferimento di ciascuna lettura, però, rimane uguale a quello che si ha per un singolo disco.

Un'altra soluzione al problema dell'affidabilità è quella di memorizzare informazioni supplementari che non sono normalmente necessarie ma che possono essere usate per ricostruire le informazioni perse in caso di guasto del disco. L'introduzione di ridondanza deve considerare due problemi: la scelta di una tecnica per calcolare l'informazione ridondante e la scelta di un metodo per distribuire l'informazione ridondante sul vettore di dischi. Si affronta il primo problema tramite l'uso di codici a correzione d'errore che utilizzano bit di parità, o codici specializzati come i codici Hamming. Con lo schema di parità si può considerare che un disco ridondante contenga la somma di tutti i dati presenti negli altri dischi. Quando un disco si guasta le informazioni mancanti possono essere ricostruite tramite un processo simile a una sottrazione.

Per il secondo problema, i due approcci più importanti consistono nel memorizzare le informazioni ridondanti su un piccolo numero di dischi o nel distribuirle uniformemente su tutti i dischi. Quest'ultimo ha come risultato un miglior bilanciamento del carico. I diversi livelli di RAID scelgono una combinazione di queste opzioni per implementare la ridondanza, e perciò per migliorare l'affidabilità.

5.3.2 Miglioramento delle prestazioni

I vettori di dischi si servono della tecnica del data striping per ottenere tassi di trasferimento migliori. Si noti che i dati possono essere letti o scritti solo un blocco alla volta, cosicché un tipico trasferimento comprende 512 byte. La suddivisione su disco può essere eseguita a un livello di granularità più fine, scomponendo un byte di dati in bit e distribuendo i bit su dischi diversi. Perciò il **data striping a livello di bit** (*bit-level data striping*) consiste nello spezzare un byte di dati e nello scrivere il bit j nel j -esimo disco. Con byte di 8 bit, otto dischi fisici possono essere considerati come un solo disco logico con un aumento di otto volte del tasso di trasferimento dati. Ogni disco partecipa a ciascuna richiesta di I/O e l'ammontare totale di dati letti per ogni richiesta è di otto volte i dati letti dal singolo disco. Lo striping a livello di

⁷ Le formule per i calcoli dell'MTTF sono presenti in Chen e altri (1994).

5.4 Bufferizzazione di blocchi

Quando devono essere trasferiti da disco a memoria centrale numerosi blocchi, e sono noti tutti i loro indirizzi, per accelerare il trasferimento possono essere riservati in memoria centrale molti buffer. Mentre viene letto o scritto un buffer, la CPU può elaborare i dati nell'altro buffer. Ciò è possibile perché esiste un processore indipendente per l'I/O di disco (controller) che, una volta avviato, può procedere nel trasferire un blocco di dati tra la memoria e il disco indipendentemente dall'elaborazione della CPU, e in parallelo con essa.

In Figura 5.5 è illustrato come due processi possano avanzare in parallelo. I processi A e B sono eseguiti **concorrentemente** e in modo **interleaved**, mentre i processi C e D sono eseguiti **concorrentemente** e in **parallelo**. Quando una sola CPU controlla più processi, l'esecuzione parallela non è possibile. Però i processi possono ancora essere eseguiti concorrentemente in modo interleaved. Là bufferizzazione è più utile quando i processi possono essere eseguiti concorrentemente in parallelo, o perché è disponibile un processore di I/O di disco separato, o perché esistono più processori di CPU.

In Figura 5.6 è illustrato come la lettura e l'elaborazione possano procedere in parallelo quando il tempo richiesto per elaborare un blocco di disco in memoria è minore del tempo richiesto per leggere il blocco successivo e riempire un buffer. La CPU può cominciare a elaborare un blocco una volta che il suo trasferimento in memoria centrale è completato; allo stesso tempo il processore dell'I/O di disco può essere impegnato nella lettura e nel trasferimento del blocco successivo in un buffer diverso. Questa tecnica è detta **doppia bufferizzazione** e può essere usata anche per scrivere un flusso continuo di blocchi da memoria a disco. Essa consente una lettura o scrittura continua di dati su blocchi di disco consecutivi, che elimina il tempo di posizionamento e il ritardo di rotazione per i trasferimenti di tutti i blocchi escluso il primo. Inoltre i dati sono tenuti pronti per essere elaborati, riducendo così il tempo di attesa nei programmi.

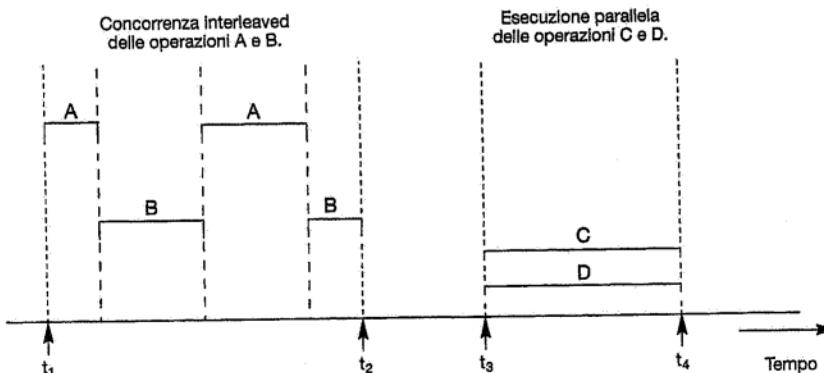


Figura 5.5 Confronto tra la concorrenza interleaved e l'esecuzione in parallelo.

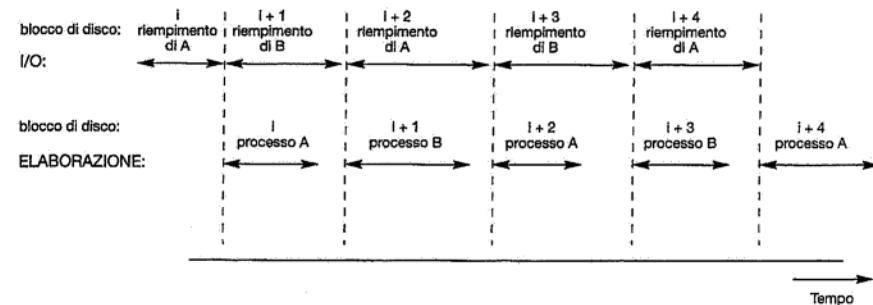


Figura 5.6 Uso di due buffer, A e B, per la lettura da disco.

5.5 Collocazione su disco dei record di un file

In questo paragrafo saranno definiti i concetti di record, tipo di record e file. Verranno poi analizzate alcune tecniche per la collocazione su disco dei record di un file.

5.5.1 Record e tipi di record

I dati sono normalmente memorizzati sotto forma di **record**. Ogni record consiste di una collezione di **valori** o **voci** di dati collegati, dove ogni valore è formato da uno o più byte e corrisponde a un particolare **campo** del record. I record di solito descrivono le entità e i loro attributi. Per esempio, un record **IMPiegato** rappresenta un'entità impiegato, e il valore di ciascun campo del record specifica alcuni attributi di quell'impiegato, come **NOME**, **DATA_NASCITA**, **STIPENDIO** o **SUPERVISORE**. Una collezione di nomi di campi e dei tipi di dati corrispondenti costituisce una definizione di **tipo di record** o **formato di record**. Un **tipo di dati**, associato a ciascun campo, specifica il tipo di valori che possono essere assunti da quel campo.

Il tipo di dati di un campo è di solito uno dei tipi di dati standard usati in programmazione. Questi comprendono i tipi di dati numerici (integer, long integer o floating point), stringhe di caratteri (a lunghezza fissa o variabile), booleani (che assumono solo i valori 0 e 1 o VERO e FALSO), e talora i tipi di dati codificati in modo speciale **date** e **time**. Per ciascun sistema di elaborazione il numero di byte richiesti per ogni tipo di dati è fissato. Un integer può richiedere 4 byte, un long integer 8 byte, un numero reale 4 byte, un booleano 1 byte, una data 10 byte (assumendo un formato del tipo YYYY-MM-DD) e una stringa a lunghezza fissa di k caratteri k byte. Le stringhe a lunghezza variabile possono richiedere tanti byte quanti sono i caratteri in ogni valore del campo. Ad esempio un tipo di record **IMPiegato** può es-

sere definito – usando la notazione del linguaggio di programmazione C – con la seguente struttura:

```
struct impiegato {
    char nome[30];
    char ssn[9];
    int stipendio;
    int codice_lavoro;
    char dipartimento[20];
};
```

In applicazioni di basi di dati recenti può sorgere la necessità di memorizzare voci di dati che consistono di grandi oggetti non strutturati, rappresentanti immagini, flussi digitalizzati audio o video, o testo libero. Queste sono indicate come **BLOB** (Binary Large Object: grande oggetto binario). Una voce di dati BLOB è tipicamente memorizzata separatamente dal suo record in un pool di blocchi di disco, e ci si limita a inserire nel record un puntatore al BLOB.

5.5.2 File, record a lunghezza fissa e record a lunghezza variabile

Un **file** è una *sequenza* di record. In molti casi tutti i record presenti in un file fanno parte dello stesso tipo di record. Se ogni record nel file ha esattamente la stessa dimensione (in byte), si dice che il file è costituito da **record a lunghezza fissa**. Se record diversi nel file hanno dimensioni diverse, si dice che il file è costituito da **record a lunghezza variabile**. Un file può avere record a lunghezza variabile per molte ragioni:

- i record del file sono dello stesso tipo di record, ma uno o più campi sono di dimensione variabile (**campi di lunghezza variabile**), ad esempio, il campo NOME di IMPIEGATO può essere un campo di lunghezza variabile;
- i record del file sono dello stesso tipo di record, ma uno o più dei campi può avere valori multipli per singoli record; un tale campo è detto **campo che si ripete** (repeating field) e un gruppo di valori per quel campo è spesso detto **gruppo che si ripete** (repeating group);
- i record del file sono dello stesso tipo di record, ma uno o più campi sono **opzionali**, cioè possono assumere valori per alcuni ma non per tutti i record del file (**campi opzionali**);
- il file contiene record di *diversi tipi di record* e perciò di dimensione variabile (**file misto**). Ciò potrebbe verificarsi se record collegati di tipi diversi fossero *raggruppati* (*clustered*: collocati insieme) su blocchi di disco; ad esempio i record VOTAZIONE di uno specifico studente possono essere collocati di seguito al record di quello STUDENTE.

I record a lunghezza fissa IMPIEGATO in Figura 5.7(a) presentano una dimensione di record di 71 byte. Ogni record ha gli stessi campi, e le lunghezze dei campi sono fisse, cosicché il sistema può individuare la posizione del byte di inizio di ciascun campo relativamente alla posizione iniziale del record. Ciò facilita la localizzazione dei valori del campo da parte dei programmi che accedono a questi file. Si noti che è possibile rappresentare un file che da un

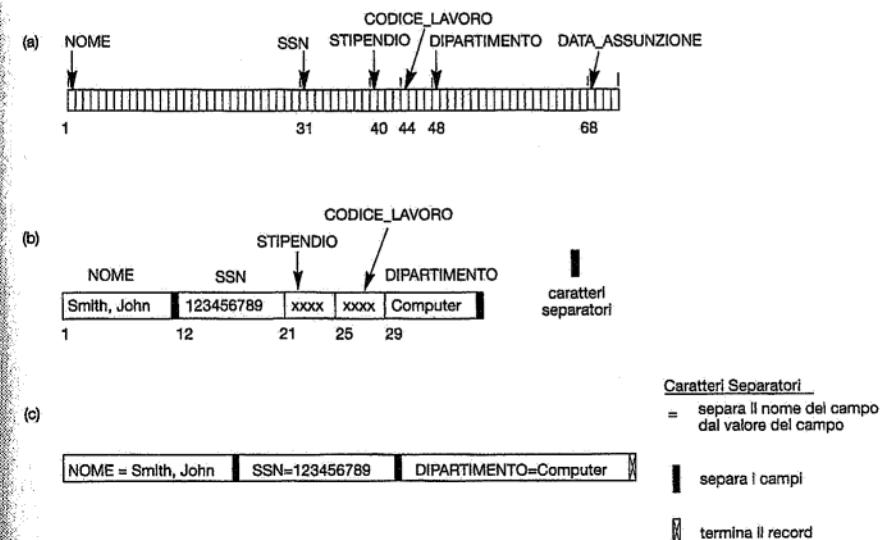


Figura 5.7 Tre formati di memorizzazione di record. (a) Un record a lunghezza fissa con sei campi e dimensione di 71 byte. (b) Un record con due campi di lunghezza variabile e tre campi di lunghezza fissa. (c) Un record a campi variabili con tre tipi di caratteri separatori.

punto di vista logico dovrebbe avere record a lunghezza variabile come un file di record a lunghezza fissa. Ad esempio, nel caso di campi opzionali sarebbe possibile aver inserito *tutti i campi in ogni record del file*, ma poi memorizzare uno speciale valore nullo se non esiste alcun valore per quel campo. Per un campo che si ripete, si potrebbero allocare in ogni record tanti spazi quant'è il *massimo numero di valori* che il campo può assumere. In entrambi i casi viene sprecato spazio quando certi record non presentano valori per tutti gli spazi fisici forniti in ciascun record. Si considerino ora altre opzioni per formattare record di un file di record a lunghezza variabile.

Per **campi di lunghezza variabile** ogni record presenta un valore in corrispondenza a ogni campo, ma non è nota la lunghezza esatta dei valori di alcuni campi. Per determinare i byte che all'interno di un particolare record rappresentano ogni campo, si possono usare speciali caratteri **separatori** (come ? o % o \$) – che non si presentano in nessun valore del campo – per terminare campi di lunghezza variabile (Figura 5.7b), oppure memorizzare la lunghezza in byte del campo in quel record, prima del valore del campo.

Un file di record con **campi opzionali** può essere formato in diversi modi. Se il numero totale di campi per il tipo di record è grande, ma il numero di campi che si presentano effettivamente in un record tipico è piccolo, è possibile inserire in ciascun record una sequenza di coppie <nome-campo, valore-campo> piuttosto che solo i valori del campo. In Figura 5.7(c) sono usati tre tipi di caratteri separatori, anche se si potrebbe usare lo stesso carattere separatore per i primi due scopi – separare il nome del campo dal valore del campo e separare un cam-

po dal successivo. Un'opzione più pratica è quella di assegnare un breve codice tipo di campo – ad esempio un numero intero – a ciascun campo e inserire in ciascun record una sequenza di coppie <tipo-campo, valore-campo> piuttosto che coppie <nome-campo, valore-campo>.

Un campo che si ripete ha bisogno di un carattere separatore per separare i valori del campo che si ripetono e di un altro carattere separatore per indicare la fine del campo. Infine, per un file che comprende record di tipi diversi, ogni record è preceduto da un indicatore di tipo di record. Comprensibilmente i programmi che elaborano file di record a lunghezza variabile – che sono di solito parte del file system e perciò nascosti ai programmatore comuni – devono essere più complessi di quelli per record a lunghezza fissa, dove la posizione di partenza e la dimensione di ogni campo sono note e fissate.⁸

5.5.3 Ripartizione dei record in blocchi e confronto tra record con spanning e record senza spanning

I record di un file devono essere ripartiti su blocchi di disco perché un blocco è l'unità di trasferimento dati tra disco e memoria. Quando la dimensione del blocco è maggiore di quella del record, ogni blocco conterrà numerosi record, anche se alcuni file possono avere record inusualmente grandi che non possono trovar posto in un solo blocco. Si supponga che la dimensione del blocco sia di B byte. Per un file di record a lunghezza fissa della dimensione di R byte, con $B \geq R$, è possibile inserire $bfr = \lfloor B/R \rfloor$ record per blocco, dove la $\lfloor x \rfloor$ (funzione floor) arrotonda per difetto il numero x ad un intero. Il valore bfr è detto fattore di blocco (blocking factor) per il file. In generale R può non dividere B esattamente, cosicché in ogni blocco si ha un certo spazio inutilizzato, pari a

$$B - (bfr * R) \text{ byte}$$

Per utilizzare questo spazio si può memorizzare parte di un record in un blocco e il resto in un altro. Un puntatore alla fine del primo blocco punta al blocco che contiene il resto del record nel caso in cui non sia il blocco consecutivo su disco. Questa organizzazione è detta spanned (estesa), perché i record possono estendersi su più di un blocco. Ogni volta che un record è più grande di un blocco si deve usare un'organizzazione spanned. Se ai record non è concesso di attraversare i confini di un blocco, l'organizzazione è detta unspanned. Essa è usata con record a lunghezza fissa che hanno $B > R$, perché consente a ciascun record di cominciare in una locazione nota del blocco, semplificando l'elaborazione dei record. Per record a lunghezza variabile può essere usata sia un'organizzazione spanned sia un'organizzazione unspanned. Se il record tipico è grande, è vantaggioso usare l'organizzazione con spanning per ridurre lo spazio perso in ciascun blocco. In Figura 5.8 sono messi a confronto i due tipi di organizzazione.

Per record a lunghezza variabile che usano l'organizzazione spanned, ogni blocco può memorizzare un diverso numero di record. In questo caso il fattore di blocco bfr rappresenta il

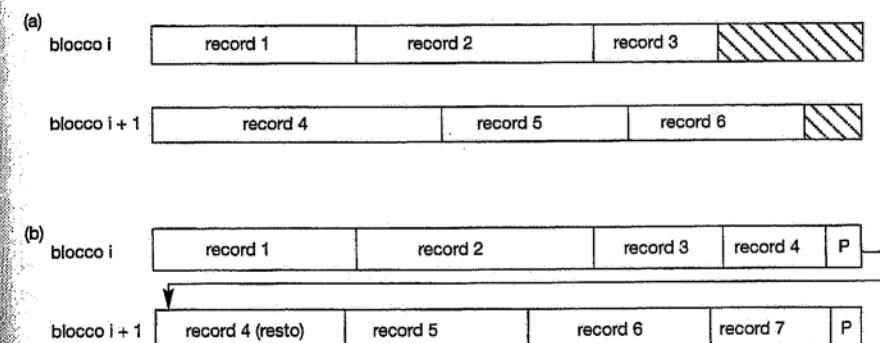


Figura 5.8 Tipi di organizzazione dei record. (a) Senza spanning. (b) Con spanning.

numero medio di record per blocco per il file. Si può usare bfr per calcolare il numero di blocchi b necessari per un file di r record:

$$b = \lceil (r/bfr) \rceil \text{ blocchi}$$

dove la $\lceil x \rceil$ (funzione ceiling) arrotonda per eccesso il valore di x al primo intero.

5.5.4 Allocazione dei blocchi di un file su disco

Ci sono molte tecniche standard per allocare i blocchi di un file su disco. Nell'allocazione contigua i blocchi del file sono allocati su blocchi di disco consecutivi: ciò rende molto veloce la lettura dell'intero file con l'uso della doppia bufferizzazione, ma nel contempo rende difficile l'aumento delle dimensioni del file. Nell'allocazione collegata ogni blocco contiene un puntatore al successivo blocco del file: ciò facilita l'aumento delle dimensioni del file ma rallenta la lettura dell'intero file. Una combinazione delle due alloca cluster di blocchi di disco consecutivi, con i cluster collegati tra loro. I cluster sono talora detti segmenti del file o estensioni del file. Un'altra possibilità è quella di usare un'allocazione indirizzata, dove uno o più blocchi di indici contengono puntatori agli effettivi blocchi del file. È anche comune usare combinazioni di queste tecniche.

5.5.5 Header dei file

Un header di file (header: intestazione) o descrittore di file contiene informazioni su un file, necessarie ai programmi di sistema che accedono ai record del file. L'header comprende informazioni per determinare gli indirizzi di disco dei blocchi del file, nonché per le descri-

zioni dei formati dei record, che possono comprendere lunghezze di campo e ordine dei campi all'interno di un record per record a lunghezza fissa senza spanning, e codici di tipo di campo, caratteri separatori e codici di tipo di record per record a lunghezza variabile.

Per la ricerca di un record su disco, uno o più blocchi sono copiati nei buffer di memoria centrale. Quindi i programmi ricercano il record o i record desiderati entro i buffer, usando le informazioni presenti nell'header del file. Se l'indirizzo del blocco che contiene il record desiderato non è noto, i programmi di ricerca devono effettuare una **ricerca lineare** attraverso i blocchi del file. Ogni blocco del file è copiato in un buffer ed esaminato fino a che il record è localizzato o tutti i blocchi del file sono stati esaminati senza successo. Ciò può essere molto dispendioso in termini di tempo per un file grande. Lo scopo di una buona organizzazione di file è quello di localizzare il blocco che contiene un record desiderato con il minimo numero di trasferimenti di blocco.

5.6 Operazioni sui file

Le operazioni sui file sono di solito raggruppate in **operazioni di recupero** (*retrieval operations*) e **operazioni di aggiornamento** (*update operations*). Le prime non cambiano alcun dato nel file, ma si limitano a localizzare certi record in modo tale che i valori dei loro campi possano essere esaminati ed elaborati. Le seconde cambiano il file con l'inserimento o la cancellazione di record o con la modifica dei valori di alcuni campi. In entrambi i casi è possibile che si debba **selezionare** uno o più record per il recupero, la cancellazione o la modifica basandosi su una **condizione di selezione** (o **condizione di filtraggio**), che specifica i criteri che il record o i record desiderati devono soddisfare.

Si consideri un file **IMPIEGATO** con campi **NOME**, **SSN**, **STIPENDIO**, **CODICE_LAVORO** e **DIPARTIMENTO**. Una **condizione di selezione semplice** può comportare un confronto di egualanza su un certo valore di campo – ad esempio (**SSN = '123456789'**) o (**DIPARTIMENTO = 'Ricerca'**). Condizioni più complesse possono coinvolgere altri tipi di operatori di confronto, come $>$ o \geq ; un esempio è (**STIPENDIO \geq 30000**). Il caso generale consiste nell'avere come condizione di selezione un'espressione booleana arbitraria sui campi del file.

Le operazioni di ricerca su file sono generalmente basate su condizioni di selezione semplici. Una condizione complessa deve essere decomposta dal DBMS (o dal programmatore) per estrarre una condizione semplice che possa essere usata per localizzare i record su disco. Ogni record localizzato viene poi esaminato per decidere se soddisfa l'intera condizione di selezione. Ad esempio, si può estrarre la condizione semplice (**DIPARTIMENTO = 'Ricerca'**) dalla condizione complessa ((**STIPENDIO \geq 30000**) AND (**DIPARTIMENTO = 'Ricerca'**)); ogni record che soddisfa (**DIPARTIMENTO = 'Ricerca'**) viene localizzato e poi esaminato per vedere se soddisfa anche (**STIPENDIO \geq 30000**).

Quando molti record di un file soddisfano una condizione di ricerca, viene inizialmente localizzato il **primo** record – relativamente alla sequenza fisica dei record del file – nominato **record corrente**. Le operazioni di ricerca seguenti cominciano da questo record e localizzano il **successivo** record del file che soddisfa la condizione.

Le operazioni effettive per localizzare e accedere ai record di un file variano da sistema

a sistema. Si presenta qui di seguito un insieme di operazioni rappresentative. Tipicamente programmi di alto livello, come i programmi software del DBMS, accedono ai record usando questi comandi, e perciò talvolta nelle seguenti descrizioni ci si riferirà a **variabili di programma**.

- **Open:** prepara il file per la lettura o la scrittura. Allocà buffer appropriati (tipicamente almeno due) per contenere blocchi del file prelevati da disco, e recupera l'header del file. Imposta il file pointer (puntatore al file) all'inizio del file.
- **Reset:** imposta il file pointer di un file aperto all'inizio del file.
- **Find** (o **Locate**): cerca il primo record che soddisfa una condizione di ricerca. Trasferisce il blocco che contiene quel record in un buffer in memoria centrale (se non è già là). Il file pointer punta al record nel buffer ed esso diventa il **record corrente**. Talvolta vengono usati verbi diversi per indicare se il record localizzato deve essere recuperato o aggiornato.
- **Read** (o **Get**): copia il record corrente dal buffer in una variabile di programma del programma utente. Questo comando può anche far avanzare il puntatore al record corrente al record successivo del file, il che può richiedere la lettura da disco del successivo blocco del file.
- **FindNext:** ricerca nel file il successivo record che soddisfa la condizione di ricerca. Trasferisce il blocco che contiene quel record in un buffer di memoria centrale (se non si trova già là). Viene localizzato il record nel buffer ed esso diventa il record corrente.
- **Delete:** cancella il record corrente e (alla fine) aggiorna il file su disco per rispecchiare la cancellazione.
- **Modify:** modifica i valori di alcuni campi del record corrente e (alla fine) aggiorna il file su disco per rispecchiare la modifica.
- **Insert:** inserisce un nuovo record nel file localizzando il blocco in cui deve essere inserito il record, trasferendo quel blocco in un buffer in memoria centrale (se non è già là), scrivendo il record nel buffer e (alla fine) scrivendo il contenuto del buffer su disco per rispecchiare l'inserimento.
- **Close:** completa l'accesso al file rilasciando i buffer ed eseguendo tutte le altre operazioni di pulizia necessarie.

Le precedenti operazioni (tranne Open e Close) sono dette operazioni **un-record-all-a volta**, perché ogni operazione si applica a un solo record. È possibile snellire le operazioni Find, FindNext e Read in una sola operazione, Scan.

- **Scan:** se sul file sono appena state applicate le operazioni Open o Reset, restituisce il primo record; altrimenti fornisce il record successivo. Se con l'operazione è specificata una condizione, il record restituito è il primo o il successivo record che soddisfa la condizione.

Nei sistemi di basi di dati possono essere applicate a un file operazioni aggiuntive di più alto livello **un-insieme-all-a-volta**. Esempi di queste operazioni sono i seguenti.

- **FindAll:** localizza *tutti* i record nel file che soddisfano una condizione di ricerca.
- **FindOrdered:** recupera tutti i record nel file in un certo ordine specificato.
- **Reorganize:** comincia il processo di riorganizzazione. Come si avrà modo di vedere, alcune organizzazioni di file richiedono una riorganizzazione periodica. Un esempio consiste nel riordinare i record del file classificandoli sulla base di un campo specificato.

A questo punto vale la pena di notare la differenza tra i termini *organizzazione di file* e *metodo di accesso*. Un'organizzazione di file si riferisce all'organizzazione dei dati di un file in record, blocchi e strutture di accesso; ciò comprende il modo in cui i record e i blocchi sono posti nel supporto di memorizzazione e il modo in cui sono collegati. Un metodo di accesso, d'altro lato, fornisce un gruppo di operazioni – come ad esempio quelle elencate sopra – che possono essere applicate a un file. In generale è possibile applicare molti metodi di accesso a una stessa organizzazione di file. Alcuni metodi di accesso, tuttavia, possono essere applicati solo a file organizzati in certi modi. Ad esempio, non è possibile applicare un metodo di accesso a indici a un file che non ha un indice (si veda il Capitolo 6).

Di solito ci si attende di usare alcune condizioni di ricerca piuttosto che altre. Alcuni file possono essere statici, nel senso che le operazioni di aggiornamento sono raramente eseguite su di essi; altri file, più dinamici, possono cambiare frequentemente, cosicché le operazioni di aggiornamento vengono costantemente applicate ad essi. Un'organizzazione di file di successo dovrebbe consentire di eseguire il più efficientemente possibile le operazioni che ci si attende di eseguire frequentemente sul file. Ad esempio, si consideri il file IMPIEGATO (Figura 5.7a) che contiene i record relativi agli attuali impiegati di un'azienda. Ciò che ci si attende è di dover inserire record (quando vengono assunti impiegati), cancellare record (quando impiegati lasciano l'azienda) e modificare record (ad esempio quando cambia lo stipendio o il lavoro di un impiegato). La cancellazione o la modifica di un record richiede una condizione di selezione per identificare un particolare record o insieme di record. Anche il recupero di uno o più record richiede una condizione di selezione.

Se gli utenti si aspettano di applicare principalmente una condizione di ricerca basata su SSN, il progettista deve scegliere un'organizzazione di file che faciliti la localizzazione di un record dato il valore del suo SSN. Ciò può comportare un ordinamento fisico dei record basato sul valore di SSN, o la definizione di un indice su SSN (si veda il Capitolo 6). Si supponga che una seconda applicazione usi il file per generare le paghe degli impiegati, e richieda che le paghe siano raggruppate per dipartimento. Per questa applicazione è meglio memorizzare in modo contiguo tutti i record impiegato che hanno lo stesso valore di dipartimento, raggruppandoli in blocchi e magari ordinandoli per nome all'interno di ogni dipartimento. Però questa disposizione è in conflitto con l'ordinamento dei record sulla base dei valori di SSN. Se entrambe le applicazioni sono importanti, il progettista dovrebbe scegliere un'organizzazione che consenta di eseguire efficientemente entrambe le operazioni. Purtroppo in molti casi ciò non è possibile e deve allora essere scelto un compromesso che tenga conto dell'importanza e della combinazione prevista di operazioni di recupero e aggiornamento.

Nei paragrafi successivi e nel Capitolo 6 si esamineranno vari metodi per organizzare i record di un file su disco. Per creare metodi di accesso sono usate molte tecniche generali, come l'ordinamento, l'hash e l'indicizzazione. Inoltre svariate tecniche generali per gestire gli inserimenti e le cancellazioni funzionano con molte organizzazioni di file.

5.7 File di record non ordinati (file heap)

In questo tipo di organizzazione, la più semplice e la più fondamentale, i record sono collocati nel file in ordine arbitrario, cioè i nuovi record sono inseriti alla fine del

file. Un'organizzazione di questo tipo è detta **file heap** o **file pile** (*pile*: pila).⁹ Essa viene spesso usata con percorsi di accesso opzionali, come gli indici secondari esaminati nel Capitolo 6, o per raccogliere e memorizzare record di dati in vista di un uso futuro.

L'inserimento di un nuovo record è molto efficiente: l'ultimo blocco di disco del file viene copiato in un buffer, viene aggiunto il nuovo record e poi il blocco viene riscritto sul disco. L'indirizzo dell'ultimo blocco del file viene tenuto nell'header del file. Però la ricerca di un record attraverso una qualsiasi condizione di ricerca comporta una ricerca lineare sul file blocco per blocco – una procedura dispendiosa. Se un solo record soddisfa la condizione di ricerca, allora, in media, un programma leggerà (portandoli in memoria) e ispezionerà metà dei blocchi del file prima di trovare il record. Per un file costituito da b blocchi, ciò può richiedere in media l'ispezione di $(b/2)$ blocchi. Se nessun record o viceversa molti record soddisfano la condizione di ricerca, il programma deve leggere e ispezionare tutti i b blocchi del file.

Per cancellare un record, un programma deve dapprima trovare il suo blocco, copiare il blocco in un buffer, quindi cancellare il record dal buffer e infine riscrivere il blocco su disco. Ciò lascia spazio inutilizzato nel blocco di disco. La cancellazione di un gran numero di record in questo modo ha come risultato uno spreco di spazio di memoria. Un'altra tecnica usata per la cancellazione di record consiste nel tenere memorizzato con ogni record un bit o byte supplementare, detto **indicatore di cancellazione**. Un record viene cancellato semplicemente ponendo l'indicatore di cancellazione a un certo valore. Un valore diverso dell'indicatore indica un record valido (cioè non cancellato). I programmi di ricerca considerano solo i record validi di ogni blocco, quando conducono la loro ricerca. Entrambe queste tecniche di cancellazione richiedono una **riorganizzazione** periodica del file per recuperare lo spazio inutilizzato dei record cancellati. Durante la riorganizzazione si accede consecutivamente ai blocchi del file, e i record vengono compattati rimuovendo i record cancellati. Dopo una riorganizzazione di questo tipo i blocchi sono di nuovo riempiti fino al limite delle loro capacità. Un'altra possibilità è quella di usare lo spazio lasciato libero dai record cancellati quando si inseriscono nuovi record, anche se ciò richiede una contabilità aggiuntiva per tener traccia delle locazioni vuote.

Per un file non ordinato è possibile usare sia un'organizzazione spanned sia un'organizzazione unspanned; inoltre esso può essere usato con record a lunghezza fissa o con record a lunghezza variabile. La modifica di un record a lunghezza variabile può richiedere la cancellazione del record vecchio e l'inserimento di un record modificato, perché il record modificato può non trovar posto nel suo vecchio spazio su disco.

Per leggere tutti i record ordinati secondo i valori di un certo campo si può creare una copia ordinata del file. L'ordinamento è un'operazione dispendiosa per un grande file su disco; vengono per questo usate tecniche speciali per l'**ordinamento esterno**.

Per un file di *record a lunghezza fissa* non ordinati, che usa *blocchi senza spanning* e *allocazione contigua*, viene naturale accedere a ogni record a partire dalla sua **posizione** nel file. Se i record del file sono numerati 0, 1, 2, ..., $r - 1$ e i record in ogni blocco sono numerati 0, 1, ..., $bfr - 1$, dove bfr è il fattore di blocco, allora l' i -esimo record del file è posto nel blocco $\lfloor (i/bfr) \rfloor$ ed è l' $(i \bmod bfr)$ -esimo record di quel blocco. Un file di questo tipo viene spesso

⁹ Talvolta questa organizzazione è detta **file sequenziale**.

chiamato **file relativo** o **diretto** perché si può agevolmente accedere direttamente ai record a partire dalle loro posizioni relative. L'accesso a un record a partire dalla sua posizione non aiuta a localizzare un record basandosi su una condizione di ricerca, però facilita la costituzione di percorsi di accesso al file, come ad esempio gli indici studiati nel Capitolo 6.

5.8 File di record ordinati (file sorted)

È possibile ordinare fisicamente i record di un file su disco basandosi sui valori di uno dei loro campi, detto **campo di ordinamento**. Ciò porta a un **file ordinato**, o **file sequenziale**.¹⁰ Se il campo di ordinamento è anche un **campo chiave** del file – un campo per cui è garantita l'esistenza di un valore univoco in ogni record – allora esso è detto **chiave di ordinamento** del file. In Figura 5.9 è mostrato un file ordinato con **NOME** come campo chiave di ordinamento (supponendo che gli impiegati abbiano nomi distinti).

I file di record ordinati presentano alcuni vantaggi sui file di record non ordinati. In primo luogo, la lettura dei record secondo l'ordine dei valori della chiave di ordinamento diventa estremamente efficiente, perché non è richiesta alcuna azione di riordinamento. In secondo luogo, trovare il record successivo a quello corrente, secondo l'ordine della chiave di ordinamento, di solito non richiede accessi aggiuntivi ai blocchi, perché il record successivo è nello stesso blocco di quello corrente (a meno che il record corrente non sia l'ultimo del blocco). Infine, l'uso di una condizione di ricerca basata sul valore di un campo chiave di ordinamento ha come risultato un accesso più veloce quando viene usata la tecnica di ricerca binaria, che costituisce un miglioramento rispetto alle ricerche lineari, anche se non è usata spesso per i file su disco.

Una **ricerca binaria** per file su disco può essere fatta sui blocchi anziché sui record. Si supponga che il file abbia b blocchi numerati 1, 2, ..., b , che i record siano ordinati per valore crescente del loro campo chiave di ordinamento e che si stia cercando un record il cui valore di campo chiave di ordinamento sia K . Supponendo che gli indirizzi su disco dei blocchi del file siano disponibili nell'header del file, la ricerca binaria può essere descritta tramite l'Algoritmo 5.1. Una ricerca binaria di solito accede a $\log_2(b)$ blocchi, che il record sia trovato o meno – un miglioramento rispetto alle ricerche lineari, dove, in media, si accede a $(b/2)$ blocchi quando il record viene trovato e a b blocchi quando il record non viene trovato.

ALGORITMO 5.1 Ricerca binaria basata su una chiave di ordinamento di un file su disco.

```
I ← 1; u ← b; (*b è il numero di blocchi del file*)
while (u ≥ 1) do
begin i ← (I + u) div 2;
trasferisci il blocco i del file nel buffer;
```

¹⁰ La locuzione *file sequenziale* è stata anche usata per riferirsi a file non ordinati.

```
if K < (valore del campo chiave di ordinamento del PRIMO record
nel blocco i)
then u ← i - 1
else if K > (valore del campo chiave di ordinamento dell'ULTIMO
record nel blocco i)
then I ← i + 1
else if il record con valore del campo chiave di
ordinamento = K è nel buffer
then goto trovato
else goto nontrovato;
end;
goto nontrovato;
```

Un criterio di ricerca che coinvolge le condizioni $>$, $<$, \geq e \leq sul campo di ordinamento è abbastanza efficiente, dal momento che l'ordinamento fisico dei record implica che tutti i record che soddisfano la condizione siano contigui nel file. Per esempio, relativamente alla Figura 5.9, se il criterio di ricerca è (**NOME** < 'G') – dove < significa *alfabeticamente precedente* – i record che soddisfano il criterio di ricerca sono quelli che vanno dall'inizio del file fino al primo record che ha un valore **NOME** che comincia con la lettera G.

L'ordinamento non fornisce alcun vantaggio per un accesso casuale od ordinato ai record basato sui valori degli altri *campi non di ordinamento* del file. In questi casi per l'accesso casuale si fa una ricerca lineare. Per accedere ai record in un ordine basato su un campo non di ordinamento è necessario creare un'altra copia ordinata – in ordine diverso – del file.

Per un file ordinato l'inserimento e la cancellazione di record sono operazioni dispendiose perché i record devono rimanere fisicamente ordinati. Per inserire un record si deve trovare la sua posizione corretta nel file, basandosi sul valore del suo campo di ordinamento, e quindi far spazio nel file per inserire il record in quella posizione. Per un file di grosse dimensioni ciò può essere molto dispendioso in termini di tempo perché, in media, metà dei record del file devono essere spostati per fare spazio al nuovo record. Ciò significa che metà dei blocchi del file devono essere letti e riscritti dopo che i record vengono spostati fra di loro. Per la cancellazione di un record il problema è meno grave se vengono usati gli indicatori di cancellazione e una riorganizzazione periodica.

Un'opzione per rendere più efficiente l'inserimento consiste nel tenere in ogni blocco un certo spazio inutilizzato per nuovi record. Comunque, una volta che questo spazio è stato utilizzato, il problema si ripresenta. Un altro metodo frequentemente usato consiste nel creare un file *non ordinato* temporaneo detto file di **overflow** (trabocco) o di **transazione**. Con questa tecnica il file ordinato vero e proprio è detto **file principale** o **file master**. I nuovi record vengono inseriti alla fine del file di overflow anziché nella loro posizione corretta nel file principale. Periodicamente, durante la riorganizzazione del file, il file di overflow viene ordinato e quindi fuso con il file master. L'inserimento diviene molto efficiente, ma al costo di una complessità aggiuntiva nell'algoritmo di ricerca. Il file di overflow deve essere ispezionato usando una ricerca lineare se, dopo la ricerca binaria, il record non è stato trovato nel file principale. Per applicazioni che non richiedono le informazioni più recenti, nel corso di una ricerca i record di overflow possono essere ignorati.

La modifica del valore del campo di un record dipende da due fattori: la condizione di ricerca per localizzare il record e il campo che deve essere modificato. Se la condizione di ri-

	NOME	SSN	DATA_NASCITA	LAVORO	STIPENDIO	SESSO
blocco 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
blocco 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
blocco 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
blocco 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
blocco 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
blocco 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
	⋮					
blocco n - 1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
blocco n	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

Figura 5.9 Alcuni blocchi di un file ordinato (sequenziale) di record IMPIEGATO, con NOME come campo chiave di ordinamento.

cerca coinvolge il campo chiave di ordinamento, allora si può localizzare il record usando una ricerca binaria, altrimenti occorre effettuare una ricerca lineare. Un campo non di ordinamento può essere modificato cambiando il record e riscrivendolo nella stessa locazione fisica su disco – supponendo di avere record a lunghezza fissa. La modifica del campo di ordinamento implica che il record possa cambiare la sua posizione nel file, il che richiederebbe la cancellazione del record vecchio seguita dall'inserimento del record modificato.

Leggere i record del file secondo l'ordine del campo di ordinamento è abbastanza efficiente se si trascurano i record in overflow, dal momento che i blocchi possono essere letti consecutivamente usando la doppia bufferizzazione. Per includere i record in overflow, bisogna inserirli nelle loro posizioni corrette; in questo caso si può dapprima riorganizzare il file, e poi leggere i suoi blocchi sequenzialmente. Per riorganizzare il file, prima di tutto si ordinano i record nel file di overflow, e poi li si fonda con il file master. Durante la riorganizzazione i record segnati per la cancellazione vengono rimossi.

I file ordinati sono usati raramente nelle applicazioni di basi di dati se non è usato un cammino di accesso aggiuntivo, detto **indice primario**; ciò ha come risultato un **file indicizzato-sequenziale**. Questo migliora ulteriormente il tempo di accesso casuale sul campo chiave di ordinamento (per una trattazione dettagliata degli indici si veda il Capitolo 6).

5.9 Tecniche hash

Un altro tipo di organizzazione primaria di file è basato sull'hash, che fornisce un accesso molto rapido ai record sotto certe condizioni di ricerca. Questa organizzazione è detta di solito **file hash**.¹¹ La condizione di ricerca deve essere una condizione di uguaglianza su un campo singolo, detto **campo hash** del file. Di solito il campo hash è anche un campo chiave del file; in questo caso è detto **chiave hash**. L'idea che sta dietro l'hash è quella di fornire una funzione h , detta **funzione hash** o **funzione di randomizzazione**, che è applicata al valore del campo hash di un record e fornisce l'*indirizzo* del blocco di disco in cui è memorizzato il record. Una ricerca del record all'interno del blocco può essere effettuata in un buffer in memoria centrale. Per la maggior parte dei record si ha bisogno solo di un accesso a un singolo blocco per recuperare quel record.

L'hash è usato anche come una struttura di ricerca interna in un programma, ogni volta che a un gruppo di record si accede esclusivamente usando il valore di un campo.

5.9.1 Hash interno

Per file interni, l'hash è tipicamente implementato con una **tavella hash** costruita usando un vettore di record. Si supponga che il campo di valori possibili per l'indice del vettore vada da

¹¹ Un file hash è stato anche detto *file diretto*.

(a)

	NOME	SSN	LAVORO	STIPENDIO
0				
1				
2				
3				
M-2				
M-1				

(b)

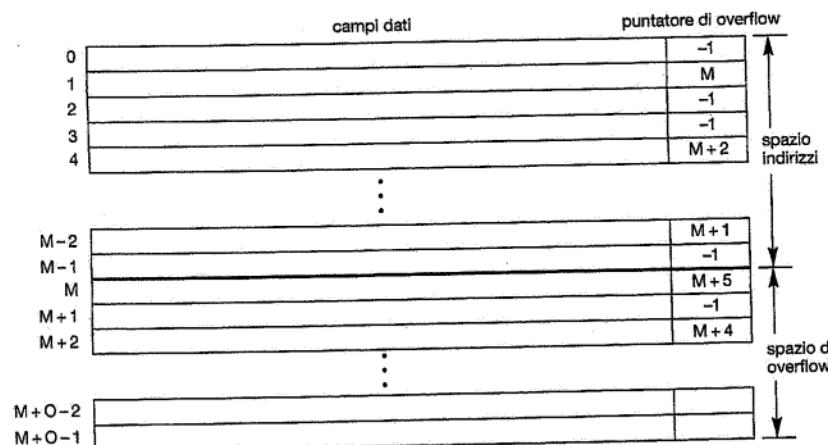


Figura 5.10 Strutture dati per l'hash interno. (a) Vettore di M posizioni usato nell'hash interno. (b) Risoluzione delle collisioni tramite concatenazione di record.

O a $M-1$ (Figura 5.10a); si ha pertanto M slot (*slot*: fessura) i cui indirizzi corrispondono agli indici del vettore. Si sceglierà allora una funzione hash che trasforma il valore del campo hash in un intero compreso tra 0 e $M-1$. Una funzione hash comune è la funzione $h(K) = K \bmod M$, che fornisce il resto di un valore intero di campo hash K dopo la divisione per M ; questo valore viene quindi usato per l'indirizzo del record.

Valori non interi del campo hash possono essere trasformati in valori interi prima che venga applicata la funzione mod. Per stringhe di caratteri possono essere usati nella trasformazione i codici numerici (ASCII) associati ai caratteri – ad esempio, moltiplicando fra loro questi valori di codice. Per un campo hash il cui tipo di dati è una stringa di 20 caratteri, per cal-

colare l'indirizzo hash può essere usato l'Algoritmo 5.2(a). Si suppone che la funzione codice restituisca il codice numerico di un carattere e che venga dato un valore K di campo hash di tipo $K: array[1..20] of char$ (in PASCAL) o $char K[20]$ (in C).

ALGORITMO 5.2 Due semplici algoritmi hash. (a) Applicazione della funzione mod hash a una stringa di caratteri K . (b) Risoluzione delle collisioni tramite l'indirizzamento aperto.

```
(a) TEMP ← 1;
   for i ← 1 to 20 do TEMP ← TEMP * CODICE (K[i]) mod M;
   INDIRIZZO_HASH ← TEMP mod M;

(b) i ← INDIRIZZO_HASH(K); a ← i;
   if la locazione i è occupata
      then begin i ← (i + 1) mod M;
            while (i ≠ a) and la locazione i è occupata
                  do i ← (i + 1) mod M;
            if (i = a) then tutte le posizioni sono piene
                  else NUOVO_INDIRIZZO_HASH ← i;
   end;
```

Possono essere usate anche altre funzioni hash.¹² Una tecnica, detta *folding (to fold: piegare)*, prevede di applicare una funzione aritmetica come l'*addizione* o una funzione logica come l'*or esclusivo* a parti diverse del valore del campo hash per calcolare l'indirizzo hash. Un'altra tecnica prevede la raccolta di alcune cifre del valore del campo hash – ad esempio la terza, la quinta e l'ottava cifra – per formare l'indirizzo hash. Il problema con la maggior parte delle funzioni hash è che esse non garantiscono che valori distinti saranno trasformati in indirizzi distinti, perché lo **spazio del campo hash** – il numero di valori possibili che un campo hash può assumere – è di solito molto più grande dello **spazio indirizzi** – il numero di indirizzi disponibili per i record. La funzione hash mappa lo spazio del campo hash nello spazio indirizzi.

Una **collisione** si verifica quando il valore del campo hash di un record che sta per essere inserito è trasformato da una funzione hash in un indirizzo che contiene già un record diverso. In questa situazione occorre inserire il nuovo record in un'altra posizione, dal momento che il suo indirizzo hash è occupato. Il processo di ricerca di un'altra posizione è detto **risoluzione delle collisioni**. Ci sono molti metodi per la risoluzione delle collisioni, fra cui quelli qui di seguito ricordati.

- **Indirizzamento aperto:** procedendo dalla posizione occupata specificata dall'indirizzo hash, il programma verifica in ordine le posizioni successive fino a che non si trova una posizione inutilizzata (vuota). L'Algoritmo 5.2(b) può essere usato allo scopo.
- **Concatenamento:** per questo metodo si tengono varie locazioni di overflow, di solito estendendo il vettore con un certo numero di posizioni di overflow. Inoltre a ogni locazione di record viene aggiunto un campo puntatore. Una collisione viene risolta ponendo il nuovo record in una locazione di overflow inutilizzata e imponendo il valore dell'indirizzo di ta-

¹² Una trattazione dettagliata delle funzioni hash esula dagli scopi di questa presentazione.

le locazione al puntatore presente nella locazione di indirizzo hash occupata. Si mantiene perciò una lista concatenata di record di overflow per ogni indirizzo hash, come mostrato in Figura 5.10(b).

- **Hash multiplo:** il programma applica una seconda funzione hash se la prima ha come risultato una collisione. Se ne risulta un'altra collisione, il programma usa l'indirizzamento aperto o applica una terza funzione hash e poi, se necessario, usa l'indirizzamento aperto.

Ogni metodo di risoluzione delle collisioni richiede il proprio algoritmo per l'inserimento, il recupero e la cancellazione di record. Gli algoritmi per il concatenamento sono i più semplici; gli algoritmi di cancellazione per l'indirizzamento aperto sono piuttosto complicati. I libri di testo sulle strutture dati esaminano gli algoritmi di hash interno in maggior dettaglio.

Lo scopo di una buona funzione hash è quello di distribuire i record uniformemente sullo spazio di indirizzi, in modo da minimizzare le collisioni senza lasciare molte locazioni inutilizzate. Studi di simulazione e di analisi hanno mostrato che di solito è meglio tenere una tabella hash piena tra il 70 e il 90 per cento; in questo modo il numero di collisioni rimane basso e nel contempo non si spreca troppo spazio. Perciò se si prevede di avere r record da memorizzare nella tabella, si dovranno scegliere M locazioni per lo spazio di indirizzi in modo tale che (r/M) sia compreso tra 0,7 e 0,9. Può essere anche utile scegliere un numero primo per M , dal momento che è stato dimostrato che ciò distribuisce meglio gli indirizzi hash sullo spazio di indirizzi quando viene usata la funzione mod hash. Altre funzioni hash possono invece richiedere che M sia una potenza di 2.

5.9.2 Hash esterno per file su disco

L'hash per file su disco è detto **hash esterno**. Per adattarsi alle caratteristiche della memorizzazione su disco, lo spazio indirizzi obiettivo è fatto di **bucket** (bucket: secchio, canestro), cioè

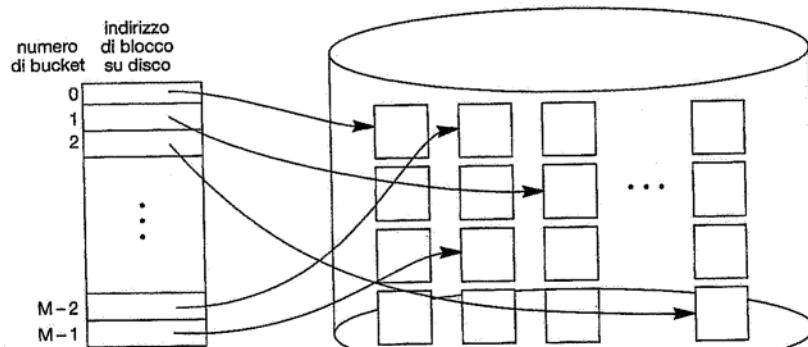


Figura 5.11 Accoppiamento tra numeri di bucket e indirizzi di blocchi di disco.

scuno dei quali contiene più record. Un bucket è un blocco di disco o un cluster di blocchi contigui. La funzione hash mappa una chiave in un numero relativo per il bucket, piuttosto di assegnare al bucket un indirizzo di blocco assoluto. Una tabella contenuta nell'header del file converte il numero del bucket nel corrispondente indirizzo di blocco di disco, come illustrato in Figura 5.11.

Con i bucket il problema delle collisioni è meno grave perché possono essere inviati nello stesso bucket senza causare problemi tanti record quanti ce ne stanno. Però occorre anche provvedere al caso in cui un bucket sia riempito fino al limite e un nuovo record che sta per essere inserito venga inviato dalla funzione hash in quel bucket. È possibile usare una variazione del concatenamento in cui in ogni bucket si mantiene un puntatore a una lista concatenata di record di overflow per il bucket, come mostrato in Figura 5.12. I puntatori nella lista concatenata dovrebbero essere **puntatori a record**, che comprendono sia un indirizzo di blocco sia una posizione relativa del record nel blocco.

L'hash fornisce il più rapido accesso possibile per il recupero di un record arbitrario dato il valore del suo campo hash. Anche se la maggior parte delle buone funzioni hash non mantengono i record nell'ordine fissato dai valori del campo chiave, ci sono alcune funzioni – dette **order preserving** (che preservano l'ordine) – che lo fanno. Un semplice esempio di funzione hash order preserving è fornito dalla funzione hash che prende come indirizzo hash le tre cifre più a sinistra di un campo numero di fattura, e che tiene in ogni bucket i record ordinati secondo il numero di fattura.

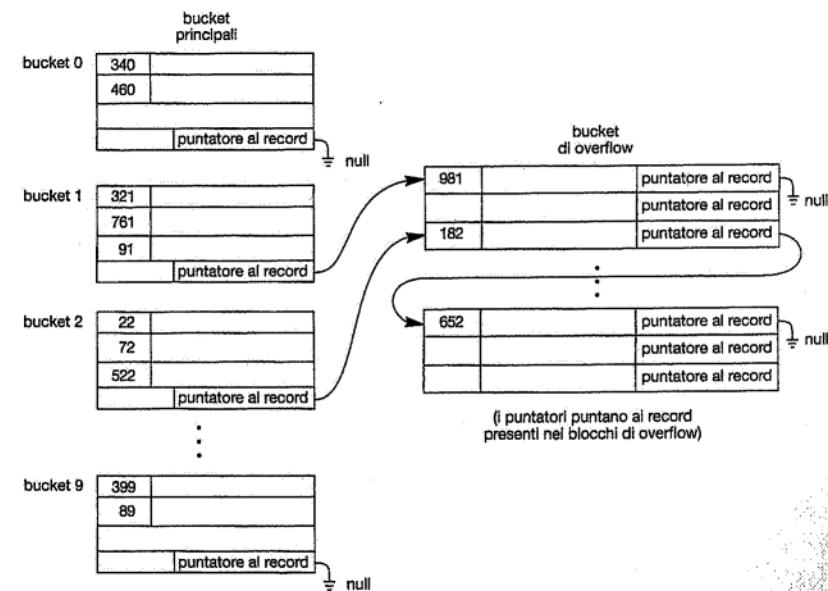


Figura 5.12 Gestione tramite concatenamento dell'overflow del bucket.

Un altro esempio consiste nell'usare una chiave hash intera direttamente come indice per un file relativo, se i valori della chiave hash riempiono un particolare intervallo; ad esempio, se i numeri di impiegato in un'azienda vengono assegnati come 1, 2, 3, ... fino al numero totale di impiegati, si può usare la funzione hash identità che mantiene l'ordine. Purtroppo ciò funziona solo se le chiavi sono generate in ordine da una certa applicazione.

Lo schema hash descritto è detto **hash statico** perché viene allocato un numero fisso M di bucket. Ciò può costituire un serio inconveniente per file dinamici. Si supponga di allocare M bucket per lo spazio di indirizzi, e sia m il numero massimo di record che possono trovar posto in un bucket; perciò nello spazio allocato troveranno posto al più $(m * M)$ record. Se succede che il numero di record è notevolmente minore di $(m * M)$, rimane molto spazio inutilizzato. D'altro lato, se il numero di record aumenta fino a molto più di $(m * M)$, si verifichino molte collisioni e il recupero sarà rallentato a causa delle lunghe liste di record di overflow. In entrambi i casi ci si potrebbe trovare nella situazione di dover cambiare il numero di blocchi allocati M e quindi usare una nuova funzione hash (basata sul nuovo valore di M) per ridistribuire i record. Queste riorganizzazioni possono essere piuttosto dispendiose in termini di tempo per file di grandi dimensioni. Organizzazioni più recenti per file dinamici basate sull'hash consentono al numero di bucket di variare dinamicamente, limitandosi a una riorganizzazione localizzata (si veda oltre).

Quando si usa l'hash esterno, la ricerca di un record, dato il valore di un certo campo diverso dal campo hash, è tanto dispendiosa quanto nel caso di un file non ordinato. La cancellazione di record può essere implementata rimuovendo il record dal suo bucket. Se il bucket ha una catena di overflow, si può spostare uno dei record di overflow nel bucket per sostituire il record cancellato. Se il record che deve essere cancellato è già in overflow, semplicemente lo si rimuove dalla lista concatenata. Si noti che la rimozione di un record in overflow implica di dover tener traccia delle posizioni di overflow vuote. Ciò può essere fatto facilmente mantenendo una lista concatenata di locazioni di overflow inutilizzate.

La modifica del valore di campo di un record dipende da due fattori: la condizione di ricerca per localizzare il record e il campo che deve essere modificato. Se la condizione di ricerca è un confronto di uguaglianza sul campo hash, è possibile localizzare il record efficientemente usando la funzione hash, altrimenti occorre fare una ricerca lineare. Un campo non hash può essere modificato cambiando il record e riscrivendolo nello stesso bucket. La modifica del campo hash implica che il record possa spostarsi in un altro bucket, il che richiede la cancellazione del record vecchio seguita dall'inserimento del record modificato.

5.9.3 Tecniche hash che consentono un'espansione dinamica dei file

Un grave inconveniente dello schema hash *statico* esaminato finora è che lo spazio indirizzi hash è fisso, e quindi è difficile espandere o ridurre il file dinamicamente. Gli schemi descritti in questo paragrafo tentano di porre rimedio a questa situazione. Il primo schema – hash estendibile – memorizza una struttura d'accesso insieme al file, e perciò è per certi versi simile all'indicizzazione (si veda il Capitolo 6). La differenza principale è che la struttura d'accesso è basata sui valori che risultano dopo l'applicazione della funzione hash al campo di ricerca. Nel-

l'indicizzazione, invece, la struttura d'accesso è basata sui valori del campo di ricerca stesso. La seconda tecnica, detta hash lineare, non richiede strutture d'accesso addizionali.

Questi schemi hash traggono profitto dal fatto che il risultato dell'applicazione di una funzione hash è un intero non negativo, e perciò può essere rappresentato come un numero binario. La struttura d'accesso è costruita sulla **rappresentazione binaria** del risultato della funzione hash, che è una stringa di bit. Tale rappresentazione sarà detta **valore hash** di un record. I record sono distribuiti fra i bucket sulla base dei valori dei *bit iniziali* (*leading bits*) presenti nei loro valori hash.

Hash estendibile. Nell'hash estendibile si mantiene una specie di **directory** (elenco) – un vettore di 2^d indirizzi di bucket, dove d è detta **profondità globale** (*global depth*) della directory. Il valore intero corrispondente ai primi d bit (i più significativi) di un valore hash è usato come un indice per il vettore per determinare un elemento della directory, e l'indirizzo in corrispondenza a quell'elemento determina il bucket in cui sono memorizzati i record corrispondenti. Però non ci deve necessariamente essere un bucket distinto per ciascuna delle 2^d locazioni della directory. Molte locazioni della directory con gli stessi d' primi bit per i loro valori hash puntano allo stesso bucket.

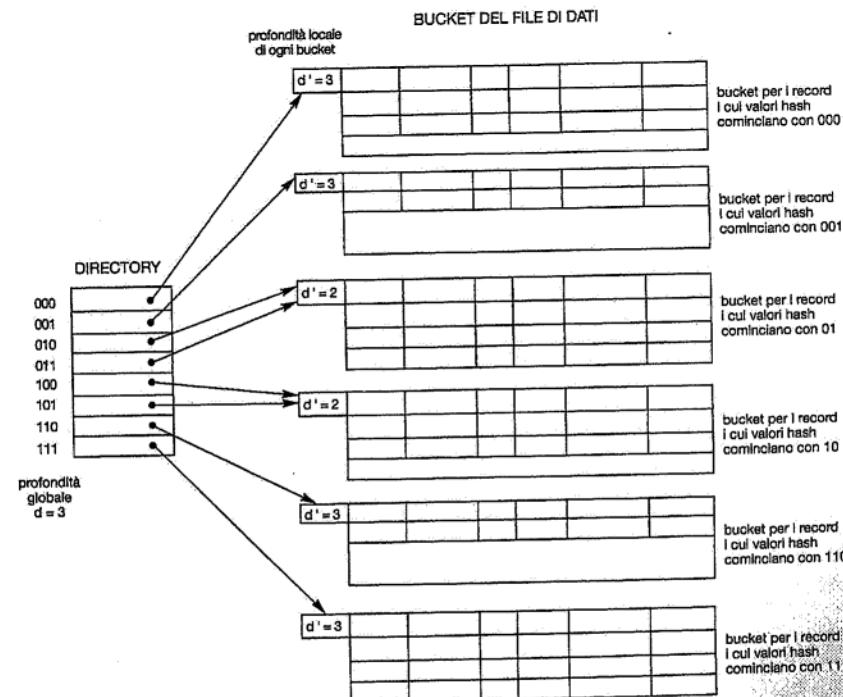


Figura 5.13 Struttura dello schema per hash estendibile.

ri hash possono contenere lo stesso indirizzo di bucket, se tutti i record che vengono inviati dalla funzione hash in queste locazioni trovano posto in un bucket singolo. Una **profondità locale (local depth) d'** – memorizzata con ogni bucket – specifica il numero di bit su cui è basato il contenuto del bucket. In Figura 5.13 viene mostrata una directory con profondità globale $d = 3$.

Il valore di d può essere incrementato o decrementato di un'unità alla volta, raddoppiando o dimezzando così il numero di elementi nel vettore directory. Il raddoppiamento è necessario se un bucket, la cui profondità locale d' è uguale alla profondità globale d , va in overflow. Il dimezzamento si verifica se $d > d'$ per tutti i bucket dopo che si verifica qualche cancellazione. La maggior parte dei recuperi di record richiede due accessi ai blocchi, uno alla directory e l'altro al bucket.

Per illustrare lo sdoppiamento dei bucket, si supponga che l'inserimento di un nuovo record causi overflow nel bucket i cui valori hash cominciano con 01 – il terzo bucket in Figura 5.13. I record saranno distribuiti tra due bucket: il primo contiene tutti i record i cui valori hash cominciano con 010, il secondo tutti quelli i cui valori hash cominciano con 011. Ora le due locazioni della directory per 010 e 011 puntano ai due nuovi bucket distinti. Prima dello sdoppiamento esse puntavano allo stesso bucket. La profondità locale d' dei due nuovi bucket è 3, che è di un'unità superiore alla profondità locale del bucket vecchio.

Se un bucket che va in overflow e deve essere sdoppiato aveva originariamente una profondità locale d' uguale alla profondità globale d della directory, allora la dimensione della directory deve essere raddoppiata; così si può usare un bit aggiuntivo per distinguere fra loro i due nuovi bucket. Ad esempio, se, con riferimento alla Figura 5.13, il bucket per i record i cui valori hash cominciano con 111 va in overflow, allora i due nuovi bucket hanno bisogno di una directory con profondità globale $d = 4$, perché i due bucket sono ora etichettati 1110 e 1111, e perciò le loro profondità locali sono entrambe pari a 4. La dimensione della directory è perciò raddoppiata, e anche ognuna delle altre locazioni originali nella directory è suddivisa in due locazioni, ciascuna delle quali ha lo stesso valore di puntatore della locazione originale.

Il principale vantaggio dell'hash estendibile, quello che lo rende attraente, è che le prestazioni del file non degradano man mano che il file cresce di dimensione, al contrario di quanto avviene con l'hash esterno statico, in cui le collisioni aumentano e il corrispondente concatenamento causa accessi aggiuntivi. Inoltre nell'hash estendibile non viene allocato nessuno spazio per una crescita futura, ma bucket aggiuntivi possono essere allocati dinamicamente quando necessario. Lo spazio in più per la tabella di directory è trascurabile. La dimensione massima della directory è 2^k , dove k è il numero di bit nel valore hash. Un altro vantaggio è che lo sdoppiamento causa in molti casi una minore riorganizzazione, dal momento che solo i record in un bucket vengono ridistribuiti sui due nuovi bucket. L'unica occasione in cui una riorganizzazione è più dispendiosa si presenta quando la directory deve essere raddoppiata (o dimezzata). Uno svantaggio è che la directory deve essere ispezionata prima di accedere ai bucket veri e propri, il che ha come risultato due accessi ai blocchi anziché uno solo, com'è invece nell'hash statico. Questa penalizzazione nelle prestazioni è considerata di minore importanza e perciò lo schema è giudicato piuttosto indicato per file dinamici.

Hash lineare. L'idea alla base dell'hash lineare è quella di consentire a un file hash di espandersi e ridurre dinamicamente il suo numero di bucket *senza* aver bisogno di una directory. Si

supponga che il file abbia inizialmente M bucket numerati 0, 1, ..., $M - 1$ e usi la funzione mod hash $h(K) = K \bmod M$; questa funzione hash è detta funzione hash iniziale h_i . L'overflow a seguito di collisioni è ancora necessario e può essere gestito mantenendo singole catene di overflow per ogni bucket. Però, quando una collisione porta a un record di overflow in *ogni* bucket del file, il *primo* bucket nel file – bucket 0 – viene sdoppiato in due bucket: il bucket originario 0 e un nuovo bucket M alla fine del file. I record originariamente nel bucket 0 sono distribuiti tra i due bucket sulla base di una diversa funzione hash $h_{i+1}(K) = K \bmod 2M$. Una proprietà fondamentale delle due funzioni hash h_i e h_{i+1} è che ogni record che era inviato al bucket 0 sulla base di h_i sarà inviato o al bucket 0 o al bucket M sulla base di h_{i+1} ; ciò è necessario perché l'hash lineare funzioni.

Quando ulteriori collisioni portano a record di overflow, vengono sdoppiati altri bucket nell'ordine *lineare* 1, 2, 3, ... Se si verificano abbastanza overflow, saranno stati sdoppiati tutti i bucket originari del file, 0, 1, 2, ..., $M - 1$, cosicché ora il file avrà $2M$ bucket anziché M , e tutti i bucket useranno la stessa funzione hash h_{i+1} . Perciò i record in overflow sono alla fine ridistribuiti in bucket regolari, usando la funzione h_{i+1} attraverso uno *sdoppiamento differente* dei loro bucket. Non c'è alcuna directory; è necessario solo un valore n – inizialmente posto a 0 e incrementato di 1 ogni volta che si verifica uno sdoppiamento – per determinare quali bucket sono stati sdoppiati. Per recuperare un record con valore di chiave hash K , dapprima si applica la funzione h_i a K ; se $h_i(K) < n$, allora si applica la funzione h_{i+1} a K perché il bucket è già stato sdoppiato. Inizialmente è $n = 0$, a indicare che la funzione h_i si applica a tutti i bucket; n cresce linearmente quando i bucket vengono sdoppiati.

Quando $n = M$ dopo essere stato incrementato, ciò significa che tutti i bucket originari sono stati sdoppiati e la funzione hash h_{i+1} si applica a tutti i record nel file. A questo punto, n è riportato a 0 (zero), e ogni nuova collisione che causa overflow porta all'uso di una nuova funzione hash $h_{i+2}(K) = K \bmod 4M$. In generale si usa una sequenza di funzioni hash $h_{i+j}(K) = K \bmod (2^j M)$, dove $j = 0, 1, 2, \dots$; è necessaria una nuova funzione hash h_{i+j+1} ogni volta che tutti i bucket 0, 1, ..., $(2^j M) - 1$ sono stati sdoppiati e n è riportato a 0. La ricerca di un record con valore della chiave hash K è eseguita dall'Algoritmo 5.3.

Lo sdoppiamento può essere controllato tramite la verifica del fattore di caricamento del file, senza doverlo così controllare ogni volta che si presenta un overflow. In generale, il **fattore di caricamento del file (file load factor)** l può essere definito come $l = r/(bfr * N)$, dove r è il numero corrente di record del file, bfr è il numero massimo di record che possono trovar posto in un bucket e N è il numero corrente di bucket del file. I bucket che sono stati sdoppiati possono anche essere riuniti se il caricamento del file scende sotto una certa soglia. I blocchi sono uniti linearmente e N è appropriatamente decrementato. Il caricamento del file può essere usato per innescare sia gli sdoppiamenti sia le unioni; in questo modo il caricamento del file può essere tenuto entro un campo desiderato. Gli sdoppiamenti possono essere provocati quando il caricamento supera una certa soglia – diciamo, 0,9 – e le unioni possono essere innestate quando il caricamento scende sotto un'altra soglia – diciamo, 0,7.

ALGORITMO 5.3 La procedura di ricerca per l'hash lineare.

```
if n = 0
    then m ← hi(K) (*m è il valore hash del record con chiave hash K*)
    else begin
```

```

 $m \leftarrow h_J(K);$ 
if  $m < n$  then  $m \leftarrow h_{J+1}(K)$ 
end;
ricerca il bucket il cui valore hash è  $m$  (e il suo overflow, se
esiste);

```

5.10 Altre organizzazioni primarie dei file

5.10.1 File di record misti

Le organizzazioni dei file studiate finora presumono che tutti i record di uno specifico file facciano parte dello stesso tipo di record. I record possono essere di tipo **IMPIEGATO**, **PROGETTO**, **STUDENTE** o **DIPARTIMENTO**, ma ogni file contiene record di un solo tipo. Nella maggior parte delle applicazioni di basi di dati si incontrano situazioni nelle quali numerosi tipi di entità sono collegati in modi diversi, come visto nel Capitolo 3. Le associazioni tra i record presenti nei vari file possono essere rappresentate da **campi di connessione**.¹³ Ad esempio, un record **STUDENTE** può avere un campo di connessione **DIP_SPECIALIZZAZIONE** il cui valore fornisce il nome del **DIPARTIMENTO** in cui lo studente si sta specializzando. Questo campo **DIP_SPECIALIZZAZIONE** rimanda a un'entità **DIPARTIMENTO**, che dovrebbe essere rappresentata da un record per conto suo nel file **DIPARTIMENTO**. Se si desidera recuperare i valori di campo da due record collegati, si deve prima recuperare uno dei record. Quindi si può usare il valore del suo campo di connessione per recuperare il record collegato nell'altro file. Perciò le associazioni sono implementate tramite **riferimenti logici di campo** tra i record presenti in file distinti.

Le organizzazioni dei file nei DBMS a oggetti, come d'altra parte nei sistemi legacy quali i DBMS gerarchico e reticolare, spesso implementano le associazioni tra i record come **associazioni fisiche**, realizzate tramite contiguità fisica (o clustering) dei record collegati o tramite puntatori fisici. Queste organizzazioni di file assegnano tipicamente un'area del disco per gestire record di più di un tipo, cosicché record di tipi diversi possono essere **raggruppati fisicamente** su disco. Se ci si aspetta che una specifica associazione venga usata molto frequentemente, l'implementazione fisica dell'associazione può aumentare l'efficienza del sistema nel recupero di record collegati. Ad esempio, se è molto frequente l'interrogazione per recuperare un record **DIPARTIMENTO** e tutti i record per ogni **STUDENTE** che si specializza in quel dipartimento, sarebbe desiderabile collocare ogni record **DIPARTIMENTO** e il relativo gruppo di record **STUDENTE** in modo contiguo su disco in un file misto. Il concetto di **raggruppamento fisico** di tipi oggetto è usato nei DBMS a oggetti per tenere insieme oggetti collegati in un file misto.

¹³ Il concetto di chiave esterna nel modello relazionale (si veda il Capitolo 7) e i riferimenti tra oggetti nei modelli a oggetti sono esempi di campi di connessione.

Per distinguere i record in un file misto ogni record ha – oltre ai suoi valori di campo – un campo **tipo di record**, che ne specifica il tipo. Questo è di solito il primo campo in ogni record ed è usato dal software di sistema per determinare il tipo di record che sta per elaborare. Usando le informazioni di catalogo il DBMS può determinare i campi di quel tipo di record e le loro dimensioni, per poter così interpretare i valori dei dati nel record.

5.10.2 Alberi B e altre strutture dati

Per le organizzazioni primarie dei file possono essere usate anche altre strutture dati. Ad esempio, se in un file sono piccoli sia la dimensione dei record sia il numero di record, alcuni DBMS offrono l'opzione di una struttura dati albero B come organizzazione primaria del file (si veda il Sottoparagrafo 6.3.1, in cui viene studiato l'uso di tale struttura per l'indicizzazione). In generale ogni struttura dati che possa essere adattata alle caratteristiche dei dispositivi a disco può essere usata come organizzazione primaria dei file per la collocazione dei record su disco.

Sommario

Abbiamo cominciato questo capitolo esaminando le caratteristiche delle gerarchie di memoria e quindi ci siamo concentrati sui dispositivi di memoria secondaria. In particolare abbiamo focalizzato la nostra attenzione sui dischi magnetici, perché essi sono usati di solito per memorizzare file in linea della base di dati. Abbiamo passato in rassegna i recenti progressi nella tecnologia dei dischi rappresentati dalla tecnologia RAID (Redundant Arrays of Inexpensive [Independent] Disks: vettori ridondanti di dischi economici [indipendenti]).

I dati su disco sono memorizzati in blocchi; l'accesso a un blocco di disco è dispendioso a causa del tempo di posizionamento, ritardo di rotazione e tempo di trasferimento di blocco. Quando si accede a blocchi di disco consecutivi si può usare la doppia bufferizzazione, per ridurre il tempo medio di accesso a un blocco. Altri parametri di disco sono esaminati nell'Appendice B. Abbiamo presentato modi diversi di memorizzazione dei record di un file su disco. I record di un file sono raggruppati in blocchi di disco e possono essere di lunghezza fissa o variabile, spanned o unspanned, e dello stesso tipo di record o di tipi diversi. Abbiamo studiato l'header dei file, che descrive i formati dei record e tiene traccia degli indirizzi di disco dei blocchi dei file. Le informazioni contenute nell'header dei file sono usate dal software di sistema che accede ai record dei file.

Abbiamo quindi presentato un insieme di tipici comandi per accedere ai singoli record di un file ed esaminato il concetto di record corrente di un file. Si è poi studiato come condizioni di ricerca di record complesse vengano trasformate in condizioni di ricerca semplici usate per localizzare i record nel file.

Abbiamo poi analizzato tre organizzazioni primarie di file: non ordinata, ordinata e hash. I file non ordinati richiedono una ricerca lineare per localizzare i record, ma l'inserimento dei



record è molto semplice. È stato studiato il problema della cancellazione e l'uso di indicatori di cancellazione.

I file ordinati riducono il tempo richiesto per leggere i record nell'ordine stabilito dal campo di ordinamento. È ridotto anche il tempo richiesto per ricercare un record arbitrario, dato il valore del suo campo chiave di ordinamento, se si usa una ricerca binaria. Però mantenere i record in ordine rende l'inserimento molto dispendioso; perciò è stata discussa la tecnica consistente nell'usare un file di overflow non ordinato. I record di overflow vengono periodicamente fusi con il file principale durante la riorganizzazione del file.

L'hash fornisce un accesso molto veloce a un record arbitrario di un file, dato il valore della sua chiave hash. Il metodo più appropriato per la realizzazione dell'hash esterno è fornito dalla tecnica dei bucket, con uno o più blocchi contigui che corrispondono a ciascun bucket. Le collisioni che causano l'overflow dei bucket sono gestite tramite il concatenamento. L'accesso a qualsiasi campo non hash è lento, come lo è l'accesso ordinato ai record su un campo qualsiasi. Abbiamo quindi esaminato due tecniche hash per file che crescono e si riducono dinamicamente nel numero di record, vale a dire l'hash estendibile e l'hash lineare.

Infine, abbiamo brevemente esaminato altre possibilità per le organizzazioni primarie dei file, come gli alberi B e i file di record misti, che implementano fisicamente le associazioni fra record di tipi diversi come parte della struttura di memorizzazione.

Questionario di verifica

- 5.1. Qual è la differenza tra memoria principale e memoria secondaria?
- 5.2. Perché vengono utilizzati i dischi e non i nastri per memorizzare i file in linea di una base di dati?
- 5.3. Si definiscono i seguenti termini: *disco, pila di dischi, traccia, blocco, cilindro, settore, spazio tra blocchi, testina di lettura/scrittura*.
- 5.4. Si esamini il processo di inizializzazione di disco.
- 5.5. Si discuta il meccanismo usato per leggere dati da disco o scrivere dati su disco.
- 5.6. Quali sono i componenti di un indirizzo di blocco di disco?
- 5.7. Perché è dispendioso l'accesso a un blocco di disco? Si discutano le componenti di tempo coinvolte nell'accesso a un blocco di disco.
- 5.8. Si descriva il disallineamento esistente tra le tecnologie dei processori e le tecnologie dei dischi.
- 5.9. Quali sono gli obiettivi principali della tecnologia RAID? Come riesce a ottenerli?
- 5.10. Come aiuta a migliorare l'affidabilità il mirroring di disco? Si dia un esempio quantitativo.
- 5.11. Quali sono le tecniche usate per migliorare le prestazioni dei dischi nella tecnologia RAID?
- 5.12. Cosa caratterizza i livelli nell'organizzazione RAID?
- 5.13. Come migliora il tempo di accesso ai blocchi la doppia bufferizzazione?
- 5.14. Quali sono le ragioni per avere record a lunghezza variabile? Quali tipi di caratteri separatori sono necessari per ciascuno di essi?
- 5.15. Si discutano le tecniche per allocare blocchi di file su disco.

- 5.16. Qual è la differenza tra un'organizzazione di file e un metodo di accesso?
- 5.17. Qual è la differenza tra file statici e file dinamici?
- 5.18. Quali sono le tipiche operazioni un-record-all-a-volta per accedere a un file? Quali di queste dipendono dal record corrente di un file?
- 5.19. Si esaminino le tecniche per la cancellazione di record.
- 5.20. Si esaminino i vantaggi e gli svantaggi nell'uso di (a) un file non ordinato, (b) un file ordinato e (c) un file hash statico con bucket e concatenazione. Quali operazioni possono essere eseguite efficientemente su ciascuna di queste organizzazioni, e quali invece sono dispendiose?
- 5.21. Si esaminino le tecniche per consentire a un file hash di espandersi e ridursi dinamicamente. Quali sono i vantaggi e gli svantaggi di ciascuna?
- 5.22. Per cosa sono usati i file misti? Quali sono gli altri tipi di organizzazioni primarie dei file?

Esercizi

- 5.23. Si consideri un disco con le seguenti caratteristiche (non si tratta di parametri di una specifica unità disco): dimensione del blocco $B = 512$ byte; dimensione dello spazio tra blocchi $G = 128$ byte; numero di blocchi per traccia = 20; numero di tracce per lato = 400. Una pila di dischi è costituita da 15 dischi a doppia faccia.
 - a. Qual è la capacità totale di una traccia e quale la sua capacità utile (escludendo gli spazi tra blocchi)?
 - b. Quanti cilindri ci sono?
 - c. Quali sono la capacità totale e la capacità utile di un cilindro?
 - d. Quali sono la capacità totale e la capacità utile di una pila di dischi?
 - e. Si supponga che l'unità disco faccia ruotare la pila di dischi a una velocità di 2400 rpm (revolutions per minute: giri al minuto); quali sono i tassi di trasferimento (*tr: transfer rate*) in byte/msec e il tempo di trasferimento di blocco (*btt: block transfer time*) in msec? Qual è il ritardo di rotazione (*rd: rotational delay*) medio in msec? Qual è il tasso di trasferimento di una mole di dati? (si veda l'Appendice B).
 - f. Si supponga che il tempo di posizionamento medio sia di 30 msec. Quanto tempo richiede (in media) in msec localizzare e trasferire un singolo blocco, dato il suo indirizzo di blocco?
 - g. Si calcoli il tempo medio che sarebbe richiesto per trasferire 20 blocchi a caso, e lo si confronti con il tempo che sarebbe richiesto per trasferire 20 blocchi consecutivi usando la doppia bufferizzazione per risparmiare tempo di posizionamento e ritardo di rotazione.
- 5.24. Un file ha $r = 20.000$ record STUDENTE di lunghezza fissa. Ogni record ha i seguenti campi: NOME (30 byte), SSN (9 byte), INDIRIZZO (40 byte), TELEFONO (9 byte), DATA_NASCITA (8 byte), SESSO (1 byte), CODICE_DIP_SPEC_PRINC (4 byte), CODICE_DIP_SPEC_SEC (4 byte), CODICE_ANNO_CORSO (4 byte, integer) e PROGRAMMA_LAUREA (3 byte). Un byte supplementare è usato come indicatore di cancellazione. Il file è memorizzato su quel disco i cui parametri sono forniti nell'Esercizio 5.23.

- a. Si calcoli la dimensione dei record R in byte.
 - b. Si calcoli il fattore di blocco bfr e il numero di blocchi del file b , supponendo di usare un'organizzazione senza spanning.
 - c. Si calcoli il tempo medio necessario per trovare un record effettuando una ricerca lineare sul file se: (i) i blocchi del file sono memorizzati in modo contiguo ed è usata la doppia bufferizzazione; (ii) i blocchi del file non sono memorizzati in modo contiguo.
 - d. Si supponga che il file sia ordinato per SSN; si calcoli il tempo richiesto per trovare un record, dato il valore del suo SSN ed effettuando una ricerca binaria.
- 5.25. Si supponga che solo l'80 per cento dei record STUDENTE dell'Esercizio 5.24 abbia un valore per TELEFONO, l'85 per cento per CODICE_DIP_SPEC_PRINC, il 15 per cento per CODICE_DIP_SPEC_SEC e il 90 per cento per PROGRAMMA_LAUREA e si supponga di usare un file di record a lunghezza variabile. Ogni record ha un *tipo di campo* di 1 byte per ogni campo presente nel record, più l'indicatore di cancellazione di 1 byte e un indicatore di fine del record di 1 byte. Si supponga di usare un'organizzazione di record *con spanning*, in cui ogni blocco ha un puntatore di 5 byte al blocco successivo (questo spazio non è usato per la memorizzazione dei record).
- a. Si calcoli la lunghezza media dei record R in byte.
 - b. Si calcoli il numero di blocchi necessari per memorizzare il file.
- 5.26. Si supponga che un'unità disco abbia i seguenti parametri: tempo di posizionamento $s = 20$ msec; ritardo di rotazione $rd = 10$ msec; tempo di trasferimento di blocco $btt = 1$ msec; dimensione di blocco $B = 2400$ byte; dimensione dello spazio tra blocchi $G = 600$ byte. Un file IMPIEGATO abbia i seguenti campi: SSN, 9 byte; COGNOME, 20 byte; NOME_BATT, 20 byte; INIZ_INT, 1 byte; DATA_NASCITA, 10 byte; INDIRIZZO, 35 byte; TELEFONO, 12 byte; SSN_SUPERVISORE, 9 byte; DIPARTIMENTO, 4 byte; CODICE_LAVORO, 4 byte; *indicatore di cancellazione*, 1 byte. Il file IMPIEGATO abbia $r = 30.000$ record, formato a lunghezza fissa e usi blocchi senza spanning. Si scrivano le formule appropriate e si calcolino i seguenti valori per il file IMPIEGATO sopra descritto.
- a. La dimensione dei record R (compreso l'indicatore di cancellazione), il fattore di blocco bfr e il numero di blocchi di disco b .
 - b. Lo spazio perso in ogni blocco di disco a causa dell'organizzazione senza spanning.
 - c. Il tasso di trasferimento tr e il tasso di trasferimento di una mole di dati btr per questa unità disco (si veda l'Appendice B per le definizioni di tr e btr).
 - d. Il *numero di accessi ai blocchi* medio necessario per trovare un record arbitrario del file usando la ricerca lineare.
 - e. Il *tempo* medio in msec necessario per trovare un record arbitrario del file usando la ricerca lineare, se i blocchi del file sono memorizzati su blocchi di disco consecutivi e si usa la doppia bufferizzazione.
 - f. Il *tempo* medio in msec necessario per trovare un record arbitrario nel file usando la ricerca lineare, se i blocchi del file *non* sono memorizzati su blocchi di disco consecutivi.
 - g. Si supponga che i record siano ordinati sulla base di un certo campo chiave. Si calcoli il *numero di accessi ai blocchi* medio e il *tempo medio* necessario per trovare un record arbitrario nel file, usando la ricerca binaria.
- 5.27. Un file PARTI con Parte# come chiave hash comprende record con i seguenti valori di

- Parte#: 2369, 3760, 4692, 4871, 5659, 1821, 1074, 7115, 1620, 2428, 3943, 4750, 6975, 4981, 9208. Il file usa otto bucket, numerati da 0 a 7. Ogni bucket è costituito da un blocco di disco e contiene due record. Si carichino questi record nel file secondo l'ordine dato, usando la funzione hash $h(K) = K \bmod 8$. Si calcoli il numero medio di accessi ai blocchi per un recupero casuale basato su Parte#.
- 5.28. Si carichino i record dell'esercizio 5.27 in file hash espandibili basati sull'hash estendibile. Si mostri la struttura della directory a ogni passo, e le profondità globali e locali. Si usi la funzione hash $h(K) = K \bmod 128$.
- 5.29. Si carichino i record dell'Esercizio 5.27 in un file hash espandibile, che usa l'hash lineare. Si cominci con un singolo blocco di disco, usando la funzione hash $h_0 = K \bmod 2^0$, e si mostri come cresce il file e come cambiano le funzioni hash man mano che i record vengono inseriti. Si supponga che i blocchi vengano sdoppiati ogni volta che si verifica un overflow, e si mostri il valore di n a ogni passo.
- 5.30. Si confrontino i comandi sui file elencati nel Paragrafo 5.6 con quelli disponibili in un metodo di accesso a file con cui si abbia familiarità.
- 5.31. Si supponga di avere un file non ordinato di record a lunghezza fissa che usi un'organizzazione a record senza spanning. Si delineino algoritmi per l'inserimento, la cancellazione e la modifica di un record del file. Si enunci esplicitamente ogni assunzione fatta.
- 5.32. Si supponga di avere un file ordinato di record a lunghezza fissa e un file di overflow non ordinato per gestire l'inserimento. Entrambi i file usano record senza spanning. Si delineino algoritmi per l'inserimento, la cancellazione e la modifica di un record del file e per riorganizzare il file. Si enunci esplicitamente ogni assunzione fatta.
- 5.33. Si può pensare a tecniche diverse dal file di overflow non ordinato, da usarsi per rendere più efficienti gli inserimenti in un file ordinato?
- 5.34. Si supponga di avere un file hash di record a lunghezza fissa e che l'overflow sia trattato con il concatenamento. Si delineino algoritmi per l'inserimento, la cancellazione e la modifica di un record del file. Si enunci esplicitamente ogni assunzione fatta.
- 5.35. Si può pensare a tecniche diverse dal concatenamento per gestire overflow di bucket nell'hash esterno?
- 5.36. Si scriva pseudo-codice per l'algoritmo di inserzione per l'hash lineare e per l'hash estendibile.
- 5.37. Si scriva codice di un programma per accedere a campi singoli di record in ognuna delle seguenti circostanze. Per ogni caso, si enuncino esplicitamente le assunzioni fatte riguardanti i puntatori, i caratteri separatori e così via. Si determini il tipo di informazioni necessarie nell'header del file perché il codice scritto sia generale in ogni caso.
 - a. Record di lunghezza fissa con blocchi senza spanning.
 - b. Record di lunghezza variabile con blocchi con spanning.
 - c. Record di lunghezza variabile con campi di lunghezza variabile e blocchi con spanning.
 - d. Record di lunghezza variabile con gruppi che si ripetono e blocchi con spanning.
 - e. Record di lunghezza variabile con campi opzionali e blocchi con spanning.
 - f. Record di lunghezza variabile che consentono tutti e tre i casi presenti nelle parti c, d ed e.
- 5.38. Si supponga che un file contenga inizialmente $r = 120.000$ record di $R = 200$ byte cia-

scuno in una struttura a file non ordinato (heap). La dimensione del blocco sia $B = 2400$ byte, il tempo di posizionamento medio $s = 16$ ms, la latenza di rotazione media $rd = 8,3$ ms e il tempo di trasferimento di blocco $btt = 0,8$ ms. Si supponga che venga cancellato 1 record ogni 2 aggiunti finché il numero totale di record attivi non sia di 240.000.

- a. Quanti trasferimenti di blocchi sono necessari per riorganizzare il file?
 - b. Quanto tempo richiede trovare un record subito prima della riorganizzazione?
 - c. Quanto tempo richiede trovare un record subito dopo la riorganizzazione?
- 5.39. Si supponga di avere un file sequenziale (ordinato) di 100.000 record, dove ciascun record è di 240 byte. Si ipotizzi che sia $B = 2400$ byte, $s = 16$ ms, $rd = 8,3$ ms e $btt = 0,8$ ms. Si supponga di voler effettuare X letture indipendenti casuali di record dal file. Si potrebbe effettuare X letture casuali di blocchi oppure eseguire una sola lettura esaustiva dell'intero file cercando quegli X record. Il problema è decidere quando sarebbe più efficiente eseguire una lettura esaustiva dell'intero file piuttosto di eseguire X letture casuali individuali. Cioè, qual è il valore di X per cui una lettura esaustiva del file è più efficiente di X letture casuali? Si sviluppi il problema esprimendo la soluzione come una funzione di X .
- 5.40. Si supponga che un file hash statico abbia inizialmente 600 bucket nell'area primaria e che vengano inseriti record che creano un'area di overflow di 600 bucket. Se si riorganizza il file hash, è possibile supporre che l'overflow sia eliminato. Se il costo di riorganizzazione del file è il costo dei trasferimenti di bucket (lettura e scrittura di tutti i bucket) e la sola operazione periodica sul file è l'operazione di fetch (prelievo), allora quante volte occorrerebbe eseguire un fetch (con successo) per rendere la riorganizzazione conveniente? Cioè, il costo della riorganizzazione e il successivo costo di ricerca devono essere inferiori al costo di ricerca prima della riorganizzazione. Si giustifichi la risposta. Si dia $s = 16$ ms, $rd = 8,3$ ms, $btt = 1$ ms.
- 5.41. Si supponga di voler creare un file ad hash lineare con un fattore di caricamento di file di 0,7 e un fattore di blocco di 20 record per bucket e che esso debba inizialmente contenere 112.000 record.
- a. Quanti bucket si dovranno allocare nell'area primaria?
 - b. Quale dovrà essere il numero di bit usato per gli indirizzi dei bucket?

Bibliografia selezionata

Wiederhold (1983) presenta una dettagliata discussione e analisi dei dispositivi di memoria secondaria e delle organizzazioni dei file. I dischi ottici sono descritti in Berg e Roth (1989) e analizzati in Ford e Christodoulakis (1991). La memoria flash è esaminata da Dipert e Levy (1993). Ruemmler e Wilkes (1994) presentano una relazione sulla tecnologia dei dischi magnetici. La maggior parte dei libri di testo sulle basi di dati comprende analisi sulla materia qui presentata. Molti libri di testo sulle strutture dati, fra cui Knuth (1973), discutono l'hash statico in maggiore dettaglio; Knuth presenta una discussione completa delle funzioni hash e delle tecniche di risoluzione delle collisioni, così come del confronto delle loro prestazioni.

di testo sulle strutture dei file comprendono Claybrook (1983), Smith e Barnes (1987) e Salzberg (1988); essi esaminano ulteriori organizzazioni di file tra cui i file strutturati ad albero, e presentano algoritmi dettagliati per operazioni sui file. Altri libri di testo sulle organizzazioni dei file sono Miller (1987) e Livadas (1989). Salzberg e altri (1990) descrivono un algoritmo distribuito di ordinamento esterno. Lo striping di disco è proposto in Salem e Garcia Molina (1986). Il primo lavoro sui vettori ridondanti di dischi economici (RAID) è stato presentato da Patterson e altri (1988). Ulteriori riferimenti sono Chen e Patterson (1990) e l'eccellente relazione su RAID di Chen e altri (1994). Grochowski e Hoyt (1996) discutono le tendenze future nelle unità a disco. Varie formule per l'architettura RAID compaiono in Chen e altri (1994).

Quello di Morris (1968) è uno dei primi lavori sull'hash. L'hash estendibile è descritto in Fagin e altri (1979), mentre l'hash lineare è descritto da Litwin (1980). L'hash dinamico, che qui non abbiamo discusso in dettaglio, è stato proposto da Larson (1978). Ci sono molte variazioni proposte per l'hash estendibile e l'hash lineare; ad esempio, si vedano Cesarini e Soda (1991), Du e Tong (1991) e Hachem e Berra (1992).

Capitolo 6

Le strutture di indici per i file

In questo capitolo diamo per scontato che esista già un file con una certa organizzazione primaria con dati non ordinati, ordinati od organizzati secondo una funzione hash come quelle che sono state descritte nel Capitolo 5. Descriveremo altre strutture ausiliarie di accesso chiamate **indici**, usate per velocizzare la ricerca dei record in risposta a determinate condizioni di ricerca. Gli indici tipicamente forniscono percorsi di accesso **secondari**, che offrono metodi alternativi per accedere ai record senza influenzare la posizione fisica dei record sul disco. Permettono un accesso efficace sulla base di campi d'indicizzazione che vengono utilizzati per costruire l'indice. *Qualsiasi campo* del file può essere usato per creare un indice e sullo stesso file possono essere costruiti *più indici* su campi differenti. È possibile immaginare una varietà di indici, ciascuno dei quali utilizza una particolare struttura dati per velocizzare la ricerca. Per trovare uno o più record nel file in base a un determinato criterio di selezione su un campo di indicizzazione, inizialmente si deve accedere all'indice per conoscere la posizione di uno o più blocchi nel file in cui si trovano i record richiesti. I tipi di indici principali si basano su file ordinati (indici a un solo livello) e su strutture di dati ad albero (indici a più livelli, alberi B⁺). Gli indici possono essere creati anche tramite funzioni hash oppure altre strutture dati.

Nel prossimo paragrafo descriveremo i diversi tipi di indici ordinati a un solo livello: primario, secondario e di cluster. Partendo da un indice a un solo livello memorizzato in un file ordinato, è possibile sviluppare ulteriori indici, dando origine al concetto di indici a più livelli. Il ben noto schema d'indicizzazione ISAM (Indexed Sequential Access Method, Metodo Indicizzato di Accesso Sequenziale) si basa su quest'idea. Nel Paragrafo 6.2 esamineremo gli indici a più livelli e nel Paragrafo 6.3 gli alberi B e B⁺, le strutture di dati usate comunemente nei DBMS per implementare indici a più livelli che cambiano dinamicamente. Gli alberi B⁺ sono diventati la struttura predefinita per creare indici dinamici nella maggior parte dei DBMS relazionali. Nel Paragrafo 6.4 tratteremo i metodi alternativi di accesso ai dati in base a una combinazione di più chiavi. Nel Paragrafo 6.5, infine, vedremo come per costrui-

re gli indici possono essere utilizzate altre strutture dati, ad esempio le funzioni hash. Sarà spiegato brevemente anche il concetto di indici logici, che forniscono un ulteriore livello di gestione e accesso agli indici fisici, permettendo all'indice fisico di essere flessibile ed estensibile.

6.1 Tipi di indici ordinati a un solo livello

L'idea su cui si basa la struttura di accesso degli indici ordinati è simile a quella che sta alla base dell'indice analitico di un libro di testo, il quale riporta in ordine alfabetico concetti fondamentali con indicazione del numero di pagina in cui compaiono. La ricerca nell'indice permette di consultare un elenco di *indirizzi*, in questo caso numeri di pagina, e di utilizzare questi ultimi per individuare un termine nel testo *cercandolo* alle pagine specificate. L'alternativa, se non ci fosse l'indice, sarebbe esaminare lentamente tutto il volume, parola per parola, per trovare il termine a cui si è interessati; quest'operazione corrisponde a eseguire una ricerca lineare su un file. Naturalmente la maggior parte dei libri contiene anche altre informazioni, ad esempio i titoli di capitoli e di paragrafi, che possono aiutare a trovare il termine in questione senza dover consultare tutto il libro. L'indice, però, fornisce l'unica indicazione esatta del punto in cui esso compare.

Per un file con una data struttura di record che consiste di parecchi campi (o attributi), la struttura di accesso a indice di solito è definita su un unico campo, chiamato **campo di indicizzazione** (oppure **attributo di indicizzazione**).¹ L'indice di solito memorizza ogni valore del campo di indicizzazione corredandolo di un elenco di puntatori a tutti i blocchi del disco che contengono record con quel valore del campo. I valori nell'indice sono *ordinati* in modo che si possa eseguire una ricerca binaria. Il file dell'indice è ovviamente molto più piccolo rispetto al file dei dati, quindi una ricerca binaria è ragionevolmente efficace. L'indicizzazione a più livelli (si veda il Paragrafo 6.2) evita la necessità di una ricerca binaria, ma richiede la creazione di ulteriori indici all'indice stesso.

Esistono molti tipi di indici ordinati. L'**indice primario** è specificato sul *campo chiave di ordinamento* di un file ordinato di record. Come si è detto nel Paragrafo 5.8, il campo chiave di ordinamento è utilizzato per *ordinare fisicamente* i record dei file su disco e tutti i record hanno un *valore univoco* per quel campo. Se il campo di ordinamento non è un campo chiave, cioè se molti record nel file possono avere lo stesso valore del campo di ordinamento, può essere usato un altro tipo di indice chiamato **indice di cluster**. Si noti che un file può avere al massimo un campo di ordinamento fisico, quindi può avere al massimo un indice primario oppure un indice di cluster, ma non entrambi. Su qualsiasi campo *non di ordinamento* di un file può essere specificato un terzo tipo di indice detto **indice secondario**. Un file può avere molti indici secondari oltre al suo metodo di accesso primario.

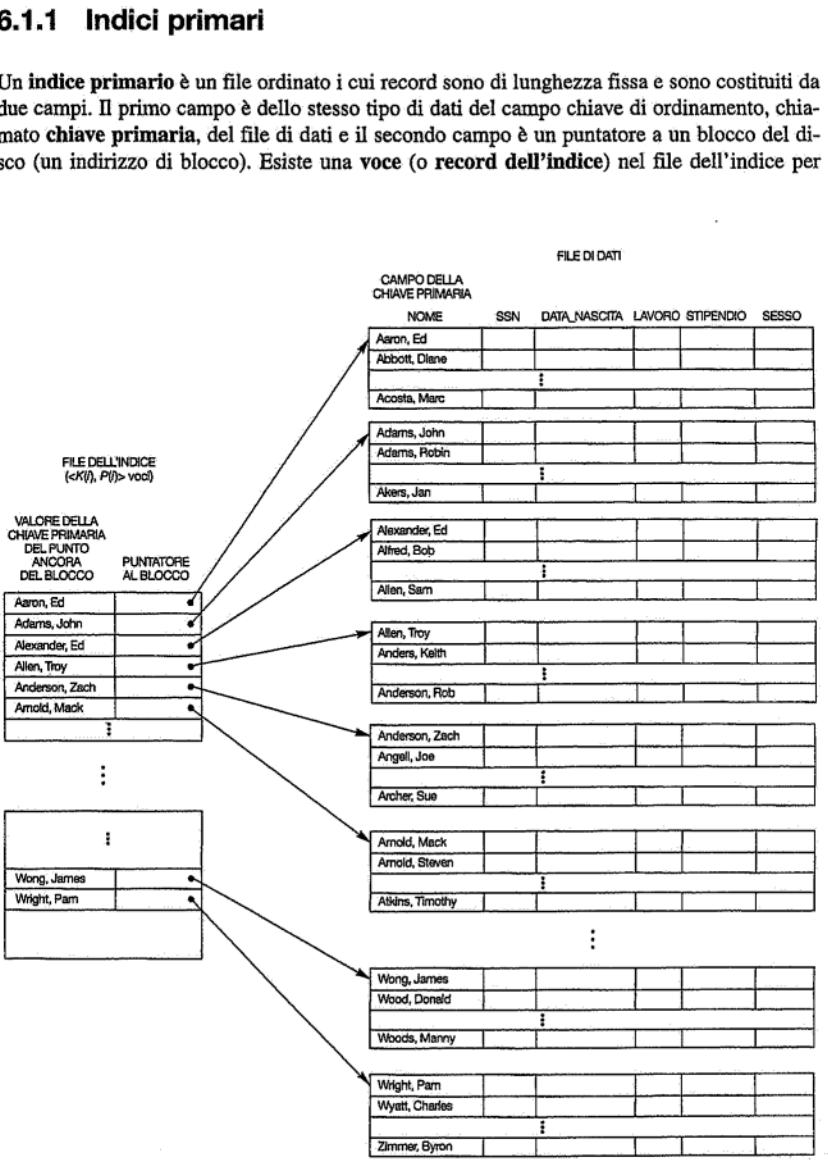


Figura 6.1 Indice primario sulla chiave di ordinamento del file mostrato in Figura 5.9.

I termini *campo* e *attributo* vengono usati in questo capitolo in modo intercambiabile.

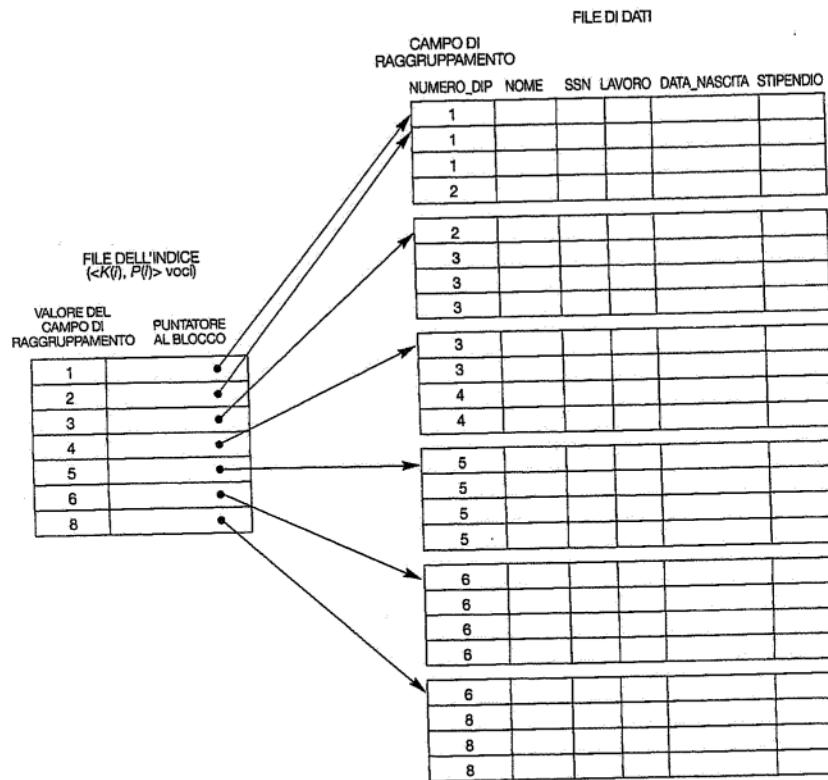


Figura 6.2 Un indice di cluster sul campo NUMERO_DIP (campo di ordinamento non chiave) di un file IMPiegato.

campo di raggruppamento; tutti i record con quel valore del campo sono posti nel blocco (o raggruppamento di blocchi). Questo rende l'inserimento e l'eliminazione relativamente semplici (Figura 6.3).

Un indice di cluster è un altro esempio di indice *non denso*, perché contiene una voce per ogni valore distinto del campo di indicizzazione piuttosto che per ogni record nel file. Vi è una certa somiglianza tra le Figure 6.1, 6.2 e 6.3 da una parte e la Figura 5.13 dall'altra. Un indice è qualcosa di simile alle strutture delle directory usate per l'hash estendibile, descritte nel Sottoparagrafo 5.9.3. In entrambi i casi, infatti, si esegue una ricerca per trovare il puntatore al blocco di dati che contiene il record desiderato. La differenza principale è che la ricerca tramite indice utilizza i valori del campo di ricerca, mentre la ricerca delle directory hash usa un valore hash che è calcolato applicando la funzione hash al campo di ricerca.

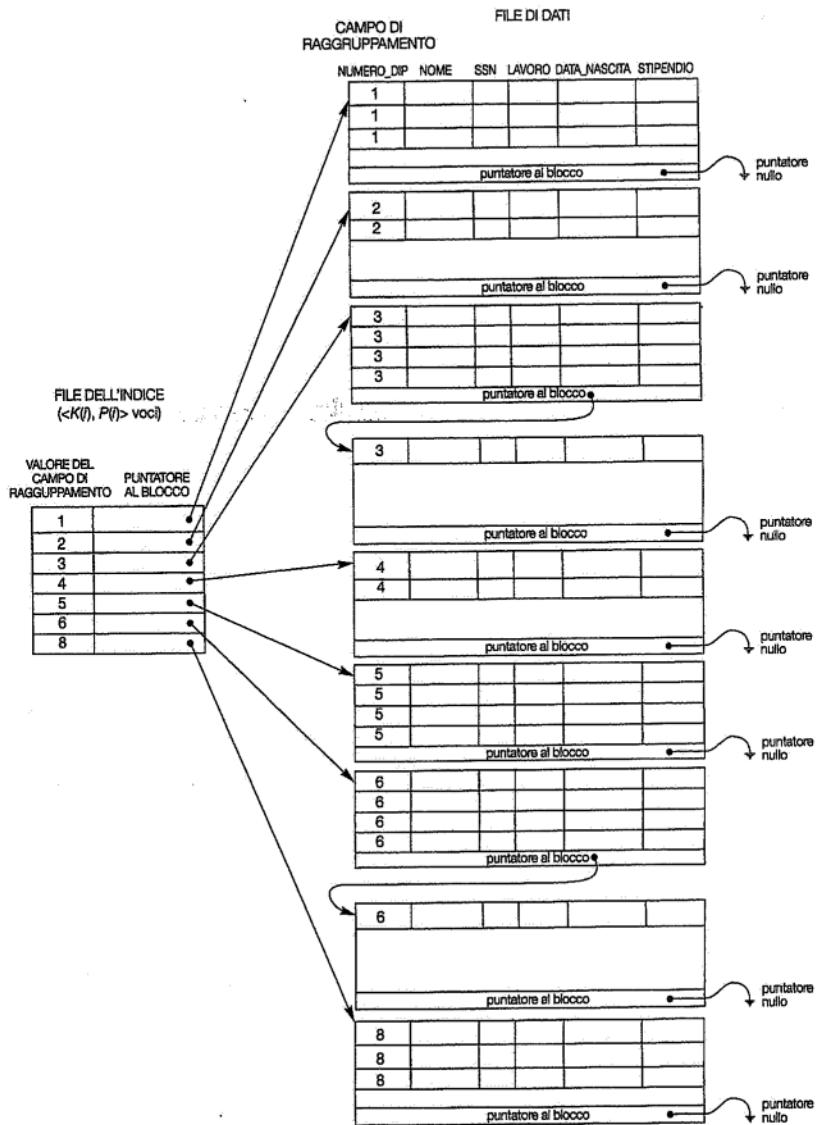


Figura 6.3 L'indice di cluster usa un blocco separato per ciascun gruppo di record che condividono lo stesso valore del campo di raggruppamento.

(senza dover accedere al blocco di dati), poiché è presente una voce dell'indice per *ogni record* nel file.

Un'organizzazione comune dei dati usata nell'elaborazione gestionale consiste in un file ordinato con un indice primario multilivello sul campo di ordinamento. Tale organizzazione è definita **file sequenziale indicizzato** ed è stata usata in molti dei primi sistemi IBM. L'inserimento è gestito da una forma di file overflow che periodicamente viene fuso nel file di dati, ricreando contestualmente l'indice. L'organizzazione ISAM dell'IBM sfrutta un indice a due livelli che è molto simile all'organizzazione fisica del disco. Il primo livello è un indice di cilindro, che contiene per ciascun valore della chiave un record di punti ancora ai cilindri di una unità disco e un puntatore all'indice delle tracce di ciascun cilindro. L'indice delle tracce a sua volta contiene il valore chiave di un record di punti ancora a ciascuna traccia del cilindro e un puntatore per traccia. È poi possibile cercare sequenzialmente il record o il blocco desiderato all'interno della traccia.

L'Algoritmo 6.1 schematizza la procedura di ricerca di un record in un file di dati che usa un indice primario multilivello non denso con t livelli. Si fa riferimento alla voce i del livello j dell'indice come $\langle K_i(i), P_j(i) \rangle$ e si cerca un record il cui valore della chiave primaria è K , ignorando la presenza di record di traboccamiento. Se il record si trova nel file, deve esserci una voce al livello 1 con $K_i(i) \leq K < K_i(i+1)$ e il record sarà nel blocco del file di dati il cui indirizzo è $P_j(i)$. L'Esercizio 6.19 illustra la modifica dell'algoritmo di ricerca per altri tipi di indici.

ALGORITMO 6.1 Ricerca in un indice primario multilevello non denso con t livelli.

```

p ← address of top level block of index;
for j ← t step - 1 to 1 do
    begin
        read the index block (at jth index level) whose address is p;
        search block p for entry i such that  $K_j(i) \leq K < K_j(i + 1)$  (i.e.
 $K_j(i)$  is the last entry in the block, it is sufficient to satisfy
 $K_j(i) \leq K$ );
        p ←  $P_j(i)$  (* picks appropriate pointer at jth index level *)
    end;
        read the data file block whose address is p;
        search block p for record with key = K;

```

- Questo algoritmo, così come i due successivi (6.2 e 6.3) non è stato tradotto in quanto scritto in pseudocodice.

Come si è visto, un indice multilivello riduce il numero degli accessi a blocchi durante la ricerca di un record, per un dato valore del campo di indicizzazione. Occorre ancora affrontare i problemi di gestione delle eliminazioni e degli inserimenti nell'indice, perché tutti i livelli dell'indice sono *file fisicamente ordinati*. Per mantenere i vantaggi dell'utilizzo dell'indicizzazione multilivello senza incorrere nei problemi di eliminazione e di inserimento, i progettisti hanno adottato un tipo di indice multilivello che lascia dello spazio libero in ciascuno dei suoi blocchi per l'immissione di nuove voci. Questo è detto **indice dinamico multilivello** ed è spesso implementato usando le strutture di dati chiamate alberi B e alberi B⁺ descritti nel paragrafo successivo.

6.3 Indici dinamici multilivello implementati da alberi B e alberi B⁺

Gli alberi B e gli alberi B⁺ sono casi speciali degli alberi della ben nota struttura di dati. Si presenta qui di seguito molto brevemente la terminologia usata nella loro descrizione. Un albero è formato da **nodi**. Ogni nodo, tranne uno speciale chiamato **radice**, ha un nodo **padre** e zero o più nodi **figlio**. La radice non ha il nodo padre. Un nodo che non ha alcun nodo figlio è chiamato **foglia**; un nodo che non è foglia è un nodo **interno**. Il **livello** di un nodo è sempre superiore di una unità rispetto al livello del suo nodo padre, mentre il livello del nodo radice è zero.⁵ Un **sottoalbero** di un nodo è costituito da quel nodo e da tutti i suoi nodi **discendenti**, i suoi nodi figli, i nodi figli dei suoi nodi figli e così via. Una precisa definizione ricorsiva di un sottoalbero dice che un sottoalbero è costituito da un nodo n e dai sottoalberi di tutti i nodi figli di n . In Figura 6.7 è illustrata una struttura di dati ad albero: il nodo radice è A e i suoi nodi figli sono B, C e D. I nodi E, J, C, G, H e K sono nodi foglia.

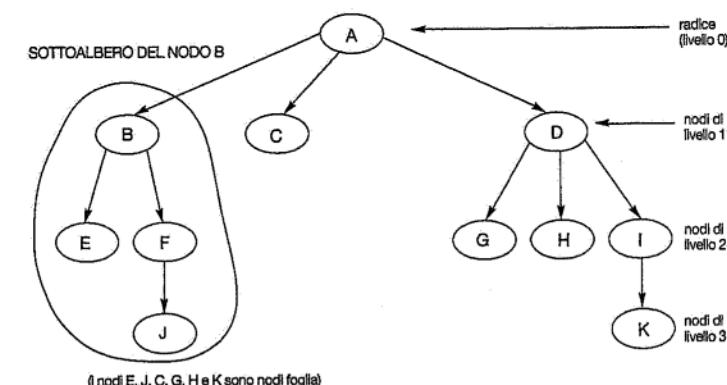


Figura 6.7 Una struttura di dati ad albero che mostra un albero non bilanciato.

Di solito si visualizza un albero con il nodo radice posto nella parte superiore, proprio come mostrato in figura. Un modo per implementare un albero è inserire in ciascun nodo tanti puntatori quanti sono i nodi figli di quel nodo. In alcuni casi in ciascun nodo è anche memorizzato un puntatore al padre. Oltre ai puntatori, un nodo di solito contiene qualche tipo di informazioni. Quando un indice multilivello viene implementato come albero, le informazioni

⁵ Questa definizione standard del livello di un nodo di un albero, che viene utilizzata in tutto il Paragrafo 6.3, è diversa da quella che è stata data per gli indici multilivello nel Paragrafo 6.2.

ni includono i valori del campo di indicizzazione usati per guidare la ricerca di un particolare record.

Nel Sottoparagrafo 6.3.1 sono presentati gli alberi di ricerca e gli alberi B, che possono essere utilizzati come indici dinamici multilivello per guidare la ricerca dei record in un file di dati. I nodi dell'albero B sono occupati fra il 50 e il 100 per cento e i puntatori ai blocchi di dati sono memorizzati sia nei nodi interni sia nei nodi foglia dell'albero. Nel Sottoparagrafo 6.3.2 sono presentati gli alberi B^+ , una variante degli alberi B in cui i puntatori ai blocchi dei dati di un file sono memorizzati solo nei nodi foglia; questo può richiedere meno livelli e permette di ottenere indici di capacità più elevata.

6.3.1 Alberi di ricerca e alberi B

Un albero di ricerca è un tipo di albero speciale che viene usato per guidare la ricerca di un record, dato il valore di uno dei suoi campi. Gli indici multilivello trattati nel Paragrafo 6.2 possono essere pensati come una variante di un albero di ricerca: ogni nodo dell'indice multilivello può avere fino a f_0 puntatori e f_0 valori chiave, dove f_0 è il fan-out dell'indice. I valori dei campi dell'indice di ciascun nodo guidano l'utente al nodo successivo finché raggiunge il blocco del file di dati che contiene i record richiesti. Seguendo un puntatore si limita a ogni passo la ricerca a un sottoalbero dell'albero di ricerca e si ignorano tutti i nodi che non sono di quel sottoalbero.

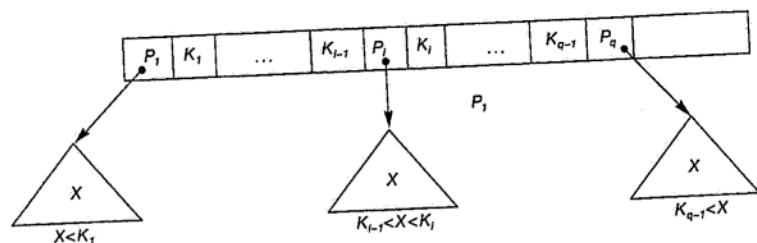


Figura 6.8 Un nodo di un albero di ricerca con i tre sottoalberi che da lui si dipartono.

Alberi di ricerca. Un albero di ricerca è leggermente differente da un indice multilivello. Un albero di ricerca di ordine p è un albero tale che ogni suo nodo contiene al massimo $p - 1$ valori di ricerca e p puntatori sono nell'ordine $\langle P_1, K_1, P_2, K_2, \dots, P_{q-p}, K_{q-p}, P_q \rangle$, in cui $q \leq p$; ogni P_i è un puntatore a un nodo figlio (oppure è un puntatore vuoto) e ogni K_i è un valore di ricerca preso da un insieme ordinato di valori.

Si suppone che tutti i valori di ricerca siano univoci.⁶ In Figura 6.8 viene mostrato un nodo in un albero di ricerca. Due vincoli devono sempre essere rispettati in un albero di ricerca:

⁶ Quando un albero è bilanciato, però non le formule che seguono devono essere modificate.

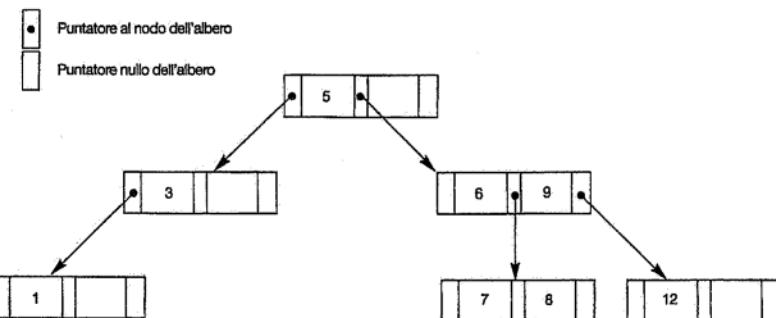


Figura 6.9 Un albero di ricerca di ordine $p = 3$.

1. all'interno di ciascun nodo, $K_1 < K_2 < \dots < K_{q-p}$;
2. per tutti i valori X del sottoalbero ai quali si fa riferimento da P_i , si ha $K_{i-1} < X < K_i$ per $1 < i < q$; $X < K_i$ per $i = 1$ e $K_{i-1} < X < K_i$ per $i = q$.

Ogni volta che si cerca un valore X , si segue il puntatore appropriato P_i secondo le formule esposte nel punto 2. In Figura 6.9 è rappresentato un albero di ricerca di ordine $p = 3$, i cui valori di ricerca sono interi. Si noti che alcuni dei puntatori P_i posti in un nodo possono essere puntatori nulli.

Si può usare un albero di ricerca come meccanismo per ricercare i record immagazzinati in un file su disco. I valori nell'albero possono essere i valori di uno dei campi del file chiamato **campo di ricerca** (che equivale al campo di indicizzazione se un indice multilivello viene utilizzato per guidare la ricerca).

Ogni valore chiave dell'albero è associato a un puntatore al record nel file di dati che ha quel valore. In alternativa il puntatore può puntare al blocco del disco che contiene quel record. Lo stesso albero di ricerca può essere memorizzato su disco assegnando ogni nodo dell'albero a un blocco del disco. Quando viene inserito un nuovo record si deve aggiornare l'albero di ricerca inserendo nell'albero una voce contenente il valore del campo di ricerca del nuovo record e un puntatore al nuovo record.

Nuovi algoritmi sono necessari per inserire e cancellare i valori di ricerca nell'albero di ricerca, mentre si continuano a rispettare i due vincoli precedenti. In generale questi algoritmi non garantiscono che un albero di ricerca sia **bilanciato**, cioè che tutti i suoi nodi foglia siano allo stesso livello.⁷ L'albero in Figura 6.7 non è bilanciato perché ha nodi foglia ai livelli 1, 2 e 3. Mantenere un albero di ricerca bilanciato è importante, perché ciò garantisce che nessun nodo si troverà a livelli molto alti e quindi richiederà molti accessi ai blocchi durante la ricerca. Un altro problema con gli alberi di ricerca è che l'eliminazione dei record possa la-

⁷ La definizione di **bilanciato** è diversa per gli alberi binari. Gli alberi binari bilanciati sono conosciuti come alberi AVL.

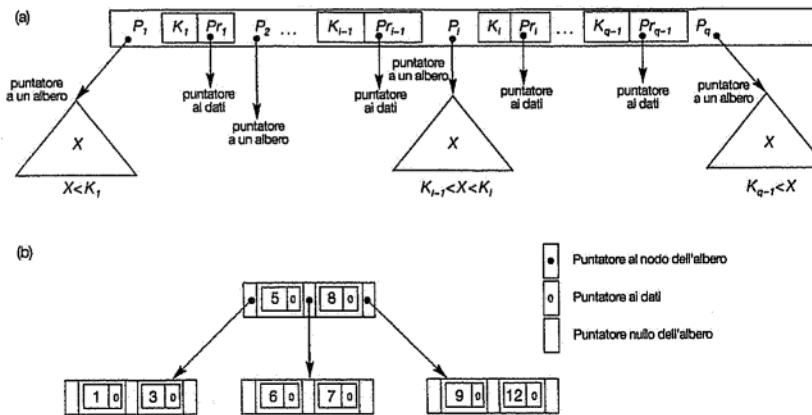


Figura 6.10 Le strutture dell'albero B. (a) Un nodo di un albero B con $(q-1)$ valori di ricerca. (b) Un albero B di ordine $p=3$. I valori sono stati inseriti nell'ordine 8, 5, 1, 7, 3, 12, 9, 6.

sciare alcuni nodi nell'albero quasi vuoti, sprecando quindi spazio per la memorizzazione e aumentando il numero dei livelli. La tecnica degli alberi B affronta entrambi questi problemi specificando ulteriori vincoli sull'albero di ricerca.

Gli alberi B. L'albero B soddisfa ulteriori vincoli che assicurano che l'albero sia sempre bilanciato e che lo spazio sprecato a seguito di cancellazione, se ce n'è, non diventi mai eccessivo. Gli algoritmi per l'inserimento e la cancellazione, però, diventano più complessi allo scopo di soddisfare questi vincoli. Ciononostante in genere gli inserimenti e le cancellazioni sono processi semplici; diventano complicati solo in particolari circostanze, cioè ogni volta che si tenta un inserimento in un nodo che è già completo oppure una cancellazione da un nodo che così diventa più della metà vuoto. In modo più formale, un **albero B di ordine p** , quando è usato come una struttura di accesso su un *campo chiave* per cercare i record in un file di dati, può essere definito nel seguente modo:

- ogni nodo interno dell'albero B (Figura 6.10a) ha questa forma:

$$<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, \dots, <K_{q-1}, Pr_{q-1}>, P_q>$$

dove $q \leq p$. Ogni P_i è un **puntatore a un albero**, cioè un puntatore a un altro nodo nell'albero B. Ogni Pr_i è un **puntatore ai dati**,⁸ cioè un puntatore al record il cui valore del campo chiave è uguale a K_i (oppure al blocco del file di dati che contiene quel record);

⁸ Un puntatore ai dati è un indirizzo di un blocco oppure l'indirizzo di un record; l'ultimo è essenzialmente l'indirizzo di un blocco e l'*offset* di un record dall'interno del blocco.

- all'interno di ogni nodo, $K_1 < K_2 < \dots < K_{q-1}$;
- per tutti i valori X del campo chiave di ricerca nel sottoalbero a cui si è fatto riferimento da P_i (il sottoalbero i -esimo, Figura 6.10a), si ha:

$$K_{i-1} < X < K_i \text{ per } 1 < i < q; X < K_i \text{ per } i = 1 \text{ e } K_{i-1} < X \text{ per } i = q;$$

- ogni nodo contiene al massimo p puntatori dell'albero;
- ogni nodo, tranne i nodi radice e i nodi foglia, ha almeno $\lceil (p/2) \rceil$ puntatori dell'albero; il nodo radice ha almeno due puntatori dell'albero a meno che sia l'unico nodo dell'albero;
- un nodo con q puntatori dell'albero, dove $q \leq p$, contiene $q-1$ valori del campo chiave di ricerca (e quindi ha $q-1$ puntatori ai dati);
- tutti i nodi foglia sono allo stesso livello e hanno la medesima struttura dei nodi interni, a parte il fatto che tutti i loro **puntatori a un albero** P_i sono nulli.

In Figura 6.10(b) è illustrato un albero B di ordine $p=3$. Si noti che tutti i valori di ricerca K di un albero B sono univoci perché si suppone che l'albero sia usato come una struttura di accesso su un campo chiave. Se si usa un albero B *su un campo non chiave*, si deve cambiare la definizione dei puntatori ai file Pr_i , perché i puntatori Pr_i devono far riferimento a un blocco o raggruppamento di blocchi, che contiene i puntatori ai record del file. Questo ulteriore livello di indici è simile all'opzione 3 presentata nel Sottoparagrafo 6.1.3 per gli indici secondari.

Un albero B inizia con un solo nodo radice (che è anche un nodo foglia) a livello 0 (zero). Quando il nodo radice diventa completo e con $p-1$ valori della chiave di ricerca e si tenta di inserire un'altra voce nell'albero, il nodo radice si divide in due nodi di livello 1. Solo il valore centrale è tenuto nel nodo radice, mentre il resto dei valori sono divisi equamente tra gli altri due nodi. Quando uno dei nodi non radice è completo e una nuova voce dovrebbe esservi inserita, il nodo è diviso in due nodi dello stesso livello e la voce centrale è spostata verso il nodo padre insieme ai due puntatori ai nuovi nodi ottenuti dalla divisione. Se il nodo padre è completo viene diviso anch'esso. La divisione può propagarsi per tutto il tragitto verso il nodo radice, creando un nuovo livello se anche la radice deve essere divisa (in questa sede non vengono trattati in dettaglio gli algoritmi di inserimento e cancellazione per gli alberi B; invece nel paragrafo successivo sono descritte le procedure di ricerca e di inserimento per gli alberi B⁺).

Se la cancellazione di un valore fa sì che un nodo sia completo per meno della metà, viene fuso con i suoi nodi vicini; anche questa fusione può propagarsi lungo tutto il tragitto verso la radice. La cancellazione, quindi, può ridurre il numero dei livelli dell'albero. È stato mostrato attraverso l'analisi e la simulazione che, dopo numerosi inserimenti e cancellazioni casuali in un albero B, i nodi sono approssimativamente completi al 69 per cento quando il numero dei valori nell'albero si stabilizza. Questo vale anche per gli alberi B⁺. Se ciò avviene, la divisione e l'unione dei nodi si verificano solo raramente, quindi l'inserimento e la cancellazione diventano abbastanza efficaci. Se il numero dei valori aumenta ancora, l'albero si espanderà senza problemi, anche se si può verificare la divisione dei nodi e così alcuni inserimenti richiederanno più tempo. L'Esempio 4 mostra come si effettua il calcolo dell'ordine p di un albero B memorizzato su disco.

ESEMPIO 4. Si supponga che il campo di ricerca sia lungo $V = 9$ byte, la dimensione del blocco del disco sia $B = 512$ byte, il puntatore a un record (di dati) sia $P_r = 7$ byte e il puntatore a un blocco sia $P = 6$ byte. Ogni nodo dell'albero B può avere al massimo p puntatori dell'albero, $p - 1$ puntatori ai dati e $p - 1$ valori del campo della chiave di ricerca (Figura 6.10a). Questi devono essere contenuti in un singolo blocco del disco, perché ciascun nodo dell'albero B deve corrispondere a un blocco del disco. Quindi si deve avere:

$$\begin{aligned}(p * P) + ((p - 1) * (P_r + V)) &\leq B \\(p * 6) + ((p - 1) * (7 + 9)) &\leq 512 \\(22 * p) &\leq 528\end{aligned}$$

Il valore p deve soddisfare la diseguaglianza precedente, il che dà $p = 23$ ($p = 24$ non viene scelto per motivi forniti in seguito).

In generale un nodo dell'albero B può contenere ulteriori informazioni necessarie per gli algoritmi che manipolano l'albero, ad esempio il numero di voci q nel nodo e un puntatore al nodo padre. Prima di eseguire il calcolo precedente per p , quindi, si dovrebbe ridurre la dimensione del blocco della quantità di spazio necessario per tutte queste informazioni. Nel seguito si illustra come calcolare il numero di blocchi e di livelli per un albero B.

ESEMPIO 5. Si supponga che il campo di ricerca dell'Esempio 4 sia un campo chiave ma non di ordinamento e che sia necessario costruire un albero B su questo campo. Si ipotizzi che ciascun nodo dell'albero B sia completo al 69 per cento. Ogni nodo, in media, avrà $p * 0,69 = 23 * 0,69$ oppure approssimativamente 16 puntatori e, quindi, 15 valori della chiave di ricerca. Il fan-out medio è $fo = 16$. Si può iniziare dalla radice e vedere quanti valori e puntatori possono esistere in media a ogni livello successivo:

Radice:	1 nodo	15 voci	16 puntatori
Livello 1:	16 nodi	240 voci	256 puntatori
Livello 2:	256 nodi	3840 voci	4096 puntatori
Livello 3:	4096 nodi	61.440 voci	

A ciascun livello si è calcolato il numero di voci moltiplicando il numero totale dei puntatori del livello precedente per 15, il numero medio di voci in ciascun nodo. Quindi, per le dimensione del blocco, dimensione del puntatore e dimensione del campo della chiave di ricerca, un albero B a due livelli contiene $3840 + 240 + 15 = 4095$ voci in media; un albero B a tre livelli contiene 65.535 voci in media.

Gli alberi B sono usati talvolta come organizzazione del file di dati. In questo caso tutti i record sono memorizzati all'interno dei nodi dell'albero B invece che solo le voci < chiave di ricerca, puntatore del record>. Questo funziona bene per i file con un numero relativamente piccolo di record e una dimensione ridotta del record, altrimenti il fan-out e il numero di livelli diventano troppo grandi per permettere un accesso efficace.

Riassumendo, gli alberi B forniscono una struttura di accesso multilivello che produce un albero bilanciato in cui ogni nodo è riempito almeno per metà. Ciascun nodo di un albero B di ordine p può avere al massimo $p - 1$ valori di ricerca.

6.3.2 Alberi B*

La maggior parte delle odiene implementazioni di indici multilivello dinamici utilizza una variante della struttura di dati ad albero B, chiamata **albero B***. In un albero B, tutti i valori del campo di ricerca appaiono una volta a un dato livello nell'albero, insieme a un puntatore ai dati. In un albero B* i puntatori ai dati sono memorizzati *solo nei nodi foglia dell'albero*; quindi la struttura dei nodi foglia differisce dalla struttura dei nodi interni. I nodi foglia contengono una voce per *ogni* valore del campo di ricerca, insieme a un puntatore al record di dati (oppure al blocco che contiene questo record) se il campo di ricerca è un campo chiave. Per un campo di ricerca non chiave, il puntatore fa riferimento a un blocco che contiene i puntatori ai record del file di dati, creando un livello ulteriore di indici.

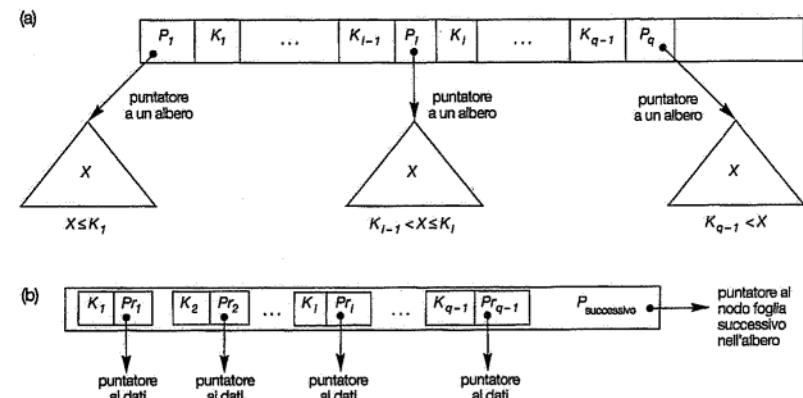


Figura 6.11 I nodi di un albero B*. (a) Un nodi interno di un albero B* con $q - 1$ valori di ricerca. (b) Un nodi foglia di un albero B* con $q - 1$ valori di ricerca e $q - 1$ puntatori ai dati.

I nodi foglia dell'albero B* di solito sono collegati tra loro per fornire un accesso ordinato al campo di ricerca e ai record. Questi nodi foglia sono simili al primo livello (base) di un indice. I nodi interni dell'albero B* corrispondono agli altri livelli di un indice multilivello. Alcuni valori del campo di ricerca dai nodi foglia sono ripetuti nei nodi interni dell'albero B* per guidare la ricerca. La struttura dei *nodi interni* di un albero B* di ordine p (Figura 6.11a) è la seguente:

- ogni nodo interno ha la seguente forma:

$$\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$$

- in cui $q \leq p$ e ogni P_i è un puntatore a un albero;
2. all'interno di ogni nodo interno, $K_1 < K_2 < \dots < K_{q-1}$;

3. per tutti i valori X del campo di ricerca nel sottoalbero a cui si fa riferimento da P_i , si ha $K_{i,1} < X \leq K_i$ per $1 < i < q$; $X \leq K_i$ per $i = 1$ e $K_{i,1} < X$ per $i = q$ (Figura 6.11a);⁹
4. ogni nodo interno ha al massimo p puntatori a un albero;
5. ogni nodo interno, tranne la radice, ha almeno $\lceil(p/2)\rceil$ puntatori dell'albero; il nodo radice ha almeno due puntatori dell'albero se è un nodo interno;
6. un nodo interno con q puntatori, $q \leq p$, ha $q - 1$ valori del campo di ricerca.

La struttura dei *nodi esterni* di un albero B^+ dell'ordine p (Figura 6.11b) è la seguente:

1. ogni nodo foglia ha la forma:

$$<\!K_1, Pr_1\!\!>, <\!K_2, Pr_2\!\!>, \dots, <\!K_q, Pr_q\!\!>, P_{successivo}\!>$$

in cui $q \leq p$, ogni Pr_i è un puntatore ai dati e $P_{successivo}$ si riferisce al *nodo foglia* successivo dell'albero B^+ ;

2. all'interno di ogni nodo foglia, $K_1 < K_2 < K_{q-1}, q \leq p$;
3. ciascun Pr_i è un *puntatore ai dati* che fa riferimento al record il cui valore del campo di ricerca è K_i o a un blocco del file che contiene il record (o a un blocco di puntatori a record che fa riferimento ai record il cui valore del campo di ricerca è K_i se il campo di ricerca non è una chiave);
4. ogni nodo foglia ha almeno $\lceil(p/2)\rceil$ valori;
5. tutti i nodi foglia sono allo stesso livello.

I puntatori nei nodi interni sono *puntatori a un albero* di blocchi che sono nodi dell'albero, mentre i puntatori nei nodi foglia sono *puntatori ai dati* che fanno riferimento ai record o ai blocchi del file di dati, a eccezione del puntatore $P_{successivo}$, che è un puntatore a un albero del successivo nodo esterno. Iniziando dal nodo foglia più a sinistra è possibile attraversare i nodi foglia come se fossero una lista concatenata, usando i puntatori $P_{successivo}$. Questo fornisce un accesso ordinato ai record di dati basato sul campo di indicizzazione. Può essere incluso anche un puntatore $P_{precedente}$. Per un albero B^+ su un campo non chiave, è necessario un livello ulteriore di indici come quello mostrato in Figura 6.5, così che i puntatori Pr sono puntatori ai blocchi che contengono un insieme di puntatori ai record effettivi del file di dati, come specificato nell'opzione 3 del Sottoparagrafo 6.1.3.

Poiché le voci nei *nodi interni* di un albero B^+ includono valori di ricerca e puntatori a un albero senza alcun puntatore ai dati, un nodo interno di un albero B^+ può contenere più voci del nodo corrispondente di un albero B . A parità di dimensione del blocco (nodo), l'ordine p sarà più grande per un albero B^+ rispetto a un albero B , come illustrato nell'Esempio 6. Questo fa sì che l'albero B^+ abbia meno livelli, migliorando il tempo di ricerca. Poiché le strutture dei nodi interni e foglia di un albero B^+ sono diverse, l'ordine p può essere differente nei due casi. Si userà p per denotare l'ordine dei *nodi interni* e p_{foglia} per denotare l'ordine dei *nodi foglia* e definito come il numero massimo dei puntatori ai dati presenti in un nodo foglia.

⁹ La definizione segue Knuth (1973). Si può definire un albero B^+ in modo differente cambiando i simboli $<$ e \leq per \leq e $<$. $K_i < K_j$ e $K_i \leq K_j$ ma i primi rimangono gli stessi.

ESEMPIO 6. Per calcolare l'ordine p di un albero B^+ si supponga che la lunghezza del campo della chiave di ricerca sia $V = 9$ byte, la dimensione del blocco sia $B = 512$ byte, il puntatore a un record sia $P_r = 7$ byte e il puntatore a un blocco sia $P = 6$ byte, come nell'Esempio 4. Un nodo interno dell'albero B^+ può avere fino a p puntatori a un albero e $p - 1$ valori del campo di ricerca; questi devono essere contenuti in un singolo blocco. Quindi si avrà:

$$\begin{aligned}(p * P) + ((p - 1) * V) &\leq B \\ (p * 6) + ((p - 1) * 9) &\leq 512 \\ (15 * p) &\leq 521\end{aligned}$$

Si può scegliere che p sia il valore più grande che soddisfa la diseguaglianza precedente, il che dà $p = 34$. Ciò è maggiore rispetto al valore 23 calcolato per l'albero B e dà come risultato un fan-out più ampio e un maggior numero di voci in ogni nodo interno di un albero B^+ rispetto al corrispondente albero B . I nodi foglia dell'albero B^+ avranno lo stesso numero di valori e puntatori, a eccezione del fatto che i puntatori sono puntatori ai dati, e un puntatore al nodo successivo. Quindi l'ordine p_{foglia} per i nodi foglia può essere calcolato nel modo seguente:

$$\begin{aligned}(p_{foglia} * (P_r + V)) + P &\leq B \\ (p_{foglia} * (7 + 9)) + 6 &\leq 512 \\ (16 * p_{foglia}) &\leq 506\end{aligned}$$

Ciascun nodo foglia può quindi contenere fino a $p_{foglia} = 31$ combinazioni di puntatori ai dati o valori chiave, dando per scontato che i puntatori ai dati siano i puntatori a record.

Anche nel caso dell'albero B^+ , può essere necessario che ogni nodo contenga ulteriori informazioni per implementare gli algoritmi d'inserimento e di cancellazione. Queste informazioni possono comprendere il tipo di nodo (interno o foglia), il numero di voci q correnti del nodo e i puntatori ai nodi padre e fratelli. Prima di eseguire i calcoli precedenti per p e p_{foglia} , quindi, si dovrebbe ridurre la dimensione del blocco della quantità di spazio necessario per contenere tutte queste informazioni. L'esempio successivo mostra come si calcola il numero di voci di un albero B^+ .

ESEMPIO 7. Si supponga di costruire un albero B^+ sul campo dell'Esempio 6. Per calcolare il numero approssimativo di voci nell'albero B^+ si supponga che ogni nodo sia completo al 69 per cento. In media ciascun nodo interno avrà $34 * 0,69 = 23$ puntatori e quindi 22 valori. Ogni nodo foglia in media conterrà $0,69 * p_{foglia} = 0,69 * 31 = 21$ puntatori ai record. L'albero B^+ avrà il seguente numero medio di voci a ogni livello:

Radice:	1 nodo	22 voci	23 puntatori
Livello1:	23 nodi	506 voci	529 puntatori
Livello2:	529 nodi	11.638 voci	12.167 puntatori
Livello esterno:	12.167 nodi	255.507 puntatori ai record	

Per la dimensione del blocco, la dimensione del puntatore e la dimensione del campo di ricerca indicati precedentemente, l'albero B⁺ a tre livelli contiene in media fino a 255.507 puntatori ai record. Si confronti questo valore con le 65.535 voci necessarie per il corrispondente albero B dell'Esempio 5.

Ricerca, inserimento e cancellazione con alberi B⁺. L'Algoritmo 6.2 schematizza la procedura che usa l'albero B⁺ come struttura di accesso per cercare un record. L'Algoritmo 6.3 illustra la procedura per inserire un record in un file che ha la struttura di accesso di un albero B⁺. Questi algoritmi presuppongono l'esistenza di un campo di ricerca che è una chiave e devono essere modificati in modo adeguato nel caso in cui si crei un albero B⁺ su un campo non chiave. Verranno illustrati ora l'inserimento e la cancellazione tramite un esempio.

ALGORITMO 6.2 La ricerca di un record con valore K del campo della chiave di ricerca, usando un albero B⁺.

```
n ← block containing root node of B+-tree;
read block n;
while (n is not a leaf node of the B+-tree) do
    begin
        q ← number of tree pointers in node n;
        if K ≤ n.K1 (*n.Ki refers to the ith search field value in node n*)
        then n ← n.P1 (*n.Pi refers to the ith tree pointer in node n*)
        else if K > n.Kq-1
        then n ← n.Pq
            else begin
                search node n for an entry i such that n.Ki-1 < K ≤ n.Ki;
                n ← n.Pi
                end;
        read block n
        end;
    search block n for entry (Ki,Pri) with K = Ki; (* search leaf node *)
if found
    then read data file block with address Pri and retrieve record
    else record with search field value K is not in the data file;
```

ALGORITMO 6.3 L'inserimento di un record con valore K del campo della chiave di ricerca in un albero B⁺ di ordine p.

```
n ← block containing root node of B+-tree;
read block n; set stack S to empty;
while (n is not a leaf node of the B+-tree) do
    begin
        push address of n on stack S;
        (*stack S holds parent nodes that are needed in case of split*)
```

```
q ← number of tree pointers in node n;
if K ≤ n.K1 (*n.Ki refers to the ith search field value in node n*)
then n ← n.P1 (*n.Pi refers to the ith tree pointer in node n*)
else if K > n.Kq-1
then n ← n.Pq
else begin
    search node n for an entry i such that n.Ki-1 < K ≤ n.Ki;
    n ← n.Pi
    end;
read block n
end;
search block n for entry (Ki,Pri) with K = Ki; (*search leaf node n*)
if found
    then record already in file-cannot insert
    else (*insert entry in B+-tree to point to record*)
begin
    create entry (K,Pr) where Pr points to the new record;
    if leaf node n is not full
    then insert entry (K, Pr) in correct position in leaf node n
    else
        begin (*leaf node n is full with pleaf record pointers-is split*)
        copy n to temp (*temp is an oversize leaf node to hold extra entry*);
        insert entry (K, Pr) in temp in correct position;
        (*temp now holds pleaf + 1 entries of the form (Ki, Pri)*)
        new ← a new empty leaf node for the tree; new.Pnext ← n.Pnext;
        j ← [pleaf + 1]/2];
        n ← first j entries in temp (up to entry (Kj,Prj)); n.Pnext ← new;
        new ← remaining entries in temp; K ← Kj;
        (*now we must move (K,new) and insert in parent internal node
        however, if parent is full, split may propagate*)
        finished ← false;
        repeat
            if stack S is empty
            then (*no parent node-new root node is created for the tree*)
            begin
                root ← a new empty internal node for the tree;
                root ← <n, K, new>; finished ← true;
                end
            else
            begin
                n ← pop stack S;
                if internal node n is not full
                then
                    begin (*parent node not full-no split*)
                    insert (K, new) in correct position in internal node n;
```

```

finished ← true
end
else
begin (*internal node n is full with p tree pointers-is split*)
copy n to temp (*temp is an oversize internal node*);
insert (K,new) in temp in correct position;
(*temp now has p+1 tree pointers*)
new ← a new empty internal node for the tree;
j ← ⌊(p + 1)/2⌋;
n ← entries up to tree pointer Pj in temp;
(*n contains <P1, K1, P2, K2, ..., Pj-1, Kj-1, Pj>)
new ← entries from tree pointer Pj+1 in temp;
(*new contains < Pj+1, Kj+1, ..., Kp-1, Pp, Kp, Pp+1 >*)
K ← Kj
(*now we must move (K,new) and insert in parent internal
node*)
end
end
until finished
end;
end;

```

In Figura 6.12 viene illustrato l'inserimento dei record in un albero B⁺ di ordine $p = 3$ e $p_{foglia} = 2$. Prima di tutto si osservi che la radice è l'unico nodo nell'albero, quindi è anche un nodo foglia. Non appena si crea più di un livello, l'albero viene diviso in nodi interni e foglia. Si noti che *ogni valore chiave deve esistere a livello delle foglie*, perché tutti i puntatori ai dati si trovano a livello delle foglie. Tuttavia solo alcuni valori sono presenti nei nodi interni per guidare la ricerca. Si consideri anche che ogni valore che appare in un nodo interno compare anche come *il valore più a destra* tra i nodi foglia del sottoalbero a cui fa riferimento il puntatore all'albero posto a sinistra del valore.

Quando un *nodo foglia* è completo e viene inserita una nuova voce, il nodo *trabocca* (overflow) e il suo contenuto viene diviso in due. Le prime $j = \lceil((p_{foglia} + 1)/2)\rceil$ voci del nodo originale sono mantenute in questo stesso nodo, mentre le restanti voci sono spostate in un nuovo nodo foglia. Il valore di ricerca j -esimo è replicato nel nodo interno padre e un ulteriore puntatore al nuovo nodo è creato nel padre. Questi valori devono essere inseriti nel nodo padre nella sequenza corretta. Se il nodo interno padre è completo, il nuovo valore gli causerà un trabocco, quindi anch'esso dovrà essere diviso in due. Le voci nel nodo interno fino a P_j , il puntatore a un albero j -esimo dopo l'inserimento del nuovo valore e del puntatore, con $j = \lfloor(p + 1)/2\rfloor$, sono mantenuti, mentre il valore di ricerca j -esimo è *spostato* nel padre, ma non ripetuto. Un nuovo nodo interno conterrà le voci rimanenti da P_{j+1} in poi (si veda Algoritmo 6.3). Questa divisione può propagarsi lungo tutto il tragitto fino a creare un nuovo nodo radice e quindi un nuovo livello dell'albero B⁺.

In Figura 6.13 è illustrata la cancellazione di una voce da un albero B⁺. Quando una voce è cancellata, è sempre rimossa dal livello delle foglie. Se capita che si trovi in un nodo inter-

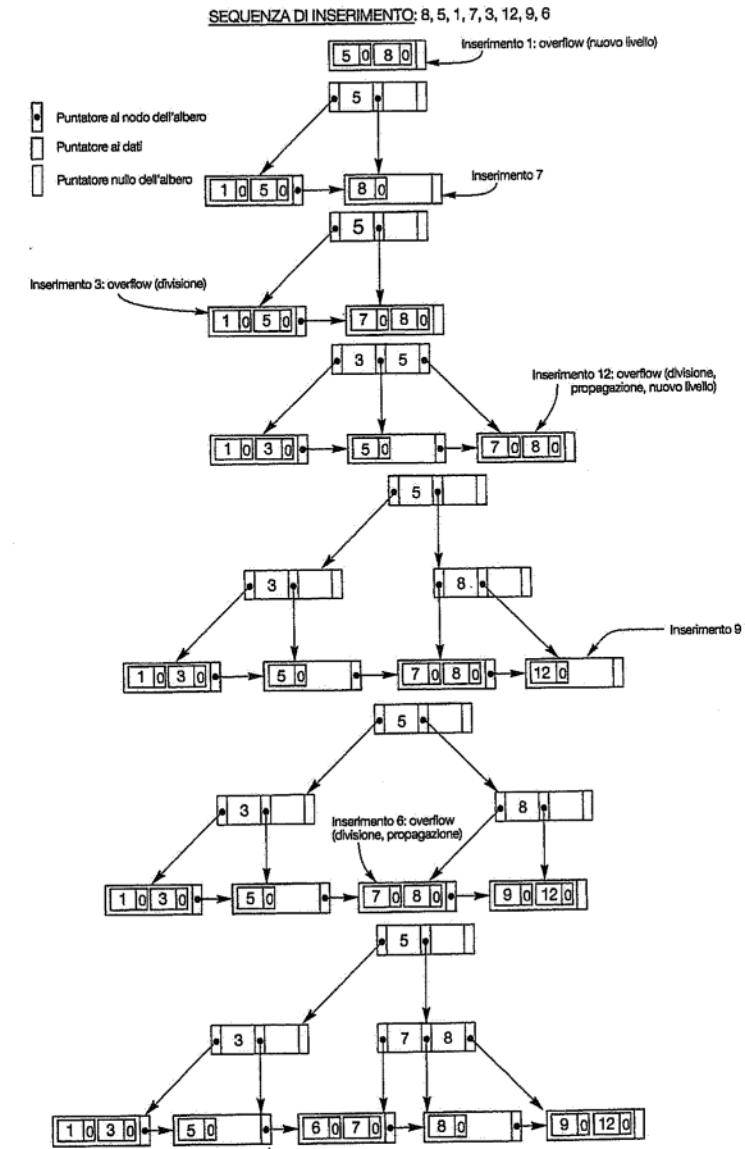
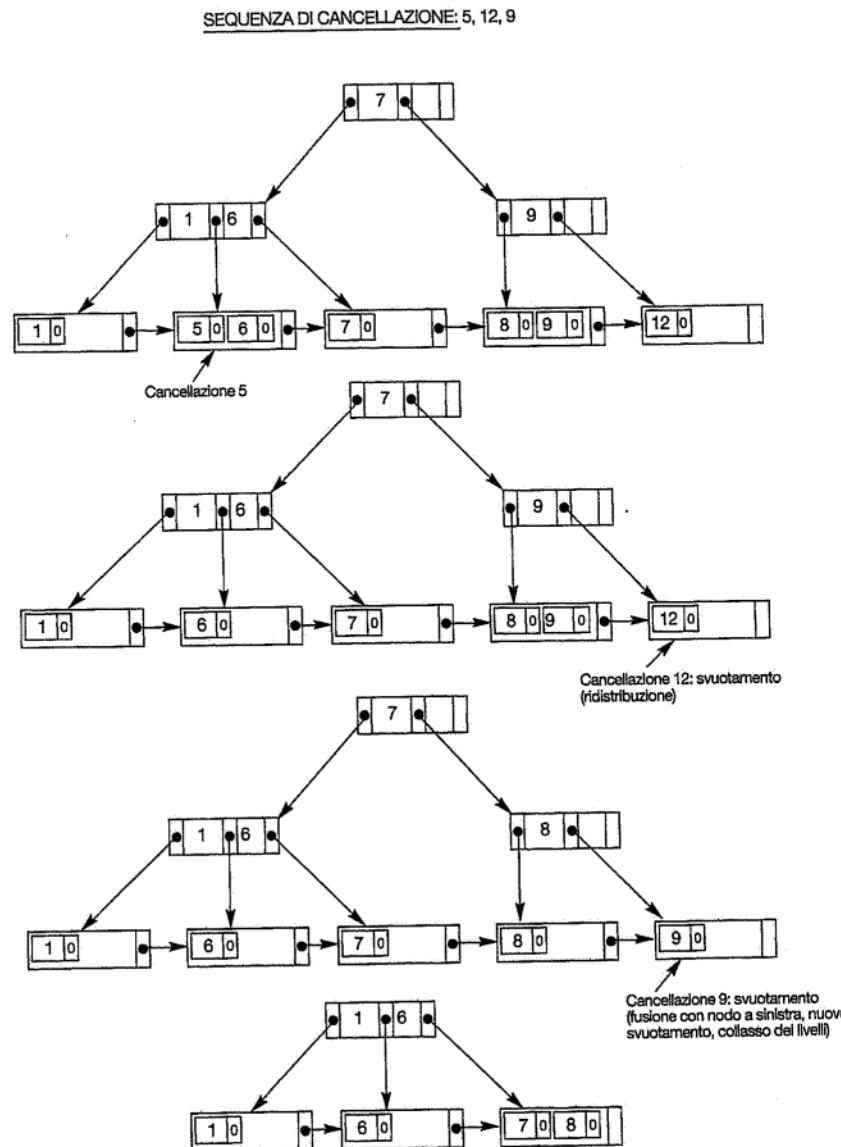


Figura 6.12 Un esempio di inserimento in un albero B⁺ con $p = 3$ e $p_{foglia} = 2$.

Figura 6.13 Un esempio di cancellazione da un albero B⁺.

no, deve essere eliminata anche da lì. In questo caso il valore alla sua sinistra nel nodo foglia deve sostituirla nel nodo interno, perché quel valore ora è la voce più a destra nel sottoalbero. La cancellazione può causare uno **svuotamento** riducendo il numero di voci nel nodo foglia sotto il minimo richiesto. In questo caso si cerca di trovare un nodo foglia fratello, cioè un nodo esterno esattamente a sinistra o a destra del nodo con lo svuotamento, e di **ridistribuire** le voci tra il nodo e il suo fratello in modo che entrambi siano almeno completi a metà; altrimenti il nodo è fuso con i suoi fratelli e il numero di nodi foglia si riduce. Un metodo comune è provare a ridistribuire le voci nel fratello a sinistra; se non è possibile, viene fatto un tentativo di ridistribuzione con il fratello a destra. Se anche questo non è possibile, i tre nodi vengono fusi ottenendo due nodi esterni. In questo caso lo svuotamento può propagarsi ai nodi interni perché sono necessari un puntatore a un albero e un valore di ricerca in meno. Questo effetto può propagarsi e ridurre i livelli dell'albero.

Si noti che l'implementazione degli algoritmi di inserimento e di cancellazione può richiedere puntatori a padre e fratelli per ciascun nodo oppure l'utilizzo di una pila come nell'Algoritmo 6.3. Ogni nodo dovrebbe includere anche il numero di voci che contiene e il suo tipo (foglia o interno). Un'altra alternativa è implementare l'inserimento e la cancellazione come procedure ricorsive.

Variazioni degli alberi B e degli alberi B⁺. Per concludere questo paragrafo si menzionano brevemente alcune varianti degli alberi B e degli alberi B⁺. In alcuni casi il vincolo 5 sull'albero B (o albero B⁺), che richiede che ciascun nodo sia almeno completo per metà, può essere modificato in modo che ogni nodo sia completo almeno per i due terzi. In questo caso l'albero B è stato chiamato albero B*. In generale alcuni sistemi permettono all'utente di scegliere un **fattore di riempimento** tra 0,5 e 1,0, in cui l'unità significa che i nodi dell'albero B (l'indice) devono essere completi. È possibile anche specificare due fattori di riempimento per un albero B⁺: uno per il livello delle foglie e uno per i nodi interni dell'albero. Quando si crea l'indice all'inizio, ciascun nodo viene riempito rispettando approssimativamente i fattori di riempimento specificati. Recentemente alcuni studiosi hanno suggerito di rilasciare il vincolo che un nodo sia completo a metà e consentire invece al nodo di diventare completamente vuoto prima della fusione, per semplificare l'algoritmo di cancellazione. Gli studi di simulazione mostrano che questo non spreca troppo spazio nel caso di inserimenti e cancellazioni casuali.

6.4 Indici su chiavi multiple

Finora si è supposto che le chiavi primarie o secondarie che si utilizzano per accedere ai file siano attributi (campi) singoli. In molte operazioni di aggiornamento e recupero, sono coinvolti simultaneamente più attributi. Se una certa combinazione di attributi è usata molto frequentemente, è vantaggioso impostare una struttura di accesso attraverso un valore chiave che è una combinazione di quegli attributi.

Ad esempio si consideri un file **IMPiegato** che contiene gli attributi **N_D** (Numero del dipartimento), **ETÀ**, **VIA**, **CITTÀ**, **CAP**, **STIPENDIO** e **CODICE_COMPETENZA**, con la chiave **SSN**.

per il valore 0 della scala, mentre a $N_D = 5$ corrisponde il valore di scala 2. In modo simile ETÀ è diviso in una scala da 0 a 5 raggruppando le età, così da distribuire uniformemente gli impiegati. La matrice a griglia mostrata per questo file ha un totale di 36 celle. Ogni cella fa riferimento a un indirizzo del bucket in cui sono memorizzati i record che corrispondono a quella cella. La Figura 6.14 mostra anche le corrispondenze tra celle e bucket (solo parzialmente).

La richiesta per $N_D = 4$ e ETÀ = 59 corrisponde alla cella (1,5) della matrice a griglia. I record che contengono questa combinazione si troveranno nel bucket corrispondente. Questo metodo è particolarmente utile per le interrogazioni su un intervallo che individuano un insieme di celle corrispondente a un gruppo di valori lungo le scale lineari. In teoria il concetto di file a griglia può essere applicato a un numero qualsiasi di chiavi di ricerca. Per n chiavi di ricerca, la matrice a griglia avrebbe n dimensioni. Essa consente una divisione del file lungo le dimensioni corrispondenti agli attributi delle chiavi di ricerca e fornisce un accesso rapido attraverso le combinazioni dei valori lungo quelle dimensioni. I file a griglia funzionano bene in termini di riduzione del tempo di accesso tramite chiavi multiple, tuttavia richiedono un consumo ulteriore di spazio in termini di struttura della matrice a griglia. Inoltre, con i file dinamici è richiesta una riorganizzazione frequente del file che si aggiunge al costo di manutenzione.¹⁰

6.5 Altri tipi di indici

6.5.1 Utilizzo di hash e di altre strutture di dati come indici

È possibile anche creare strutture di accesso simili agli indici che si basano sull'*hash*. Le voci dell'indice $\langle K, Pr \rangle$ (o $\langle K, P \rangle$) possono essere organizzate come un file hash espandibile dinamicamente usando una delle tecniche descritte nel Sottoparagrafo 5.9.3; per la ricerca di una voce si utilizza l'algoritmo hash di ricerca su K . Una volta che la voce è stata trovata, si usa il puntatore Pr (o P) per individuare il record corrispondente nel file di dati. Anche altre strutture di ricerca possono essere usate come indice.

6.5.2 Confronto tra indici logici e indici fisici

Finora si è supposto che le voci dell'indice $\langle K, Pr \rangle$ (o $\langle K, P \rangle$) includano sempre un puntatore fisico Pr (o P) che specifica l'indirizzo fisico dei record su disco come numero del blocco e offset. Questo metodo viene talvolta chiamato **indice fisico** e ha lo svantaggio che il puntatore deve essere cambiato se il record viene spostato in un'altra posizione del disco. Ad

¹⁰ Gli algoritmi d'inserimento e di cancellazione per i file a griglia si possono trovare in Nievergelt [1984].

esempio si supponga che un'organizzazione primaria di file si basi sull'*hash* lineare o sull'*hash* estendibile; ogni volta che un bucket viene diviso, alcuni record vengono assegnati a nuovi bucket e quindi hanno nuovi indirizzi fisici. Se esistesse anche un indice secondario sul file, si dovrebbero trovare e aggiornare i puntatori a quei record, un procedimento lungo e difficile.

Per rimediare a questa situazione si può usare una struttura chiamata **indice logico**, le cui voci assumono la forma $\langle K, K_p \rangle$. Ogni voce contiene un valore K per il campo secondario di indicizzazione che corrisponde al valore K_p del campo usato per l'organizzazione primaria del file. Cercando nell'indice secondario il valore di K , un programma può individuare il valore corrispondente di K_p e usarlo per accedere al record attraverso l'organizzazione primaria del file. Gli indici logici quindi introducono un ulteriore livello di indici tra la struttura di accesso e i dati. Essi vengono usati quando ci si aspetta che gli indirizzi fisici dei record cambino frequentemente. Il costo di questo ulteriore livello di indici è la ricerca ulteriore che si basa sull'organizzazione primaria del file.

6.5.3 Discussione

In molti sistemi l'indice non è una parte integrante del file di dati, ma può essere creato e scartato dinamicamente. Ecco perché spesso viene chiamato *struttura di accesso*. Ogni volta che ci si aspetta di accedere frequentemente a un file sulla base di condizioni di ricerca che coinvolgono un campo particolare, si può richiedere al DBMS di creare un indice su quel campo. Di solito viene creato un indice secondario per evitare di dover eseguire l'ordinamento fisico dei record nel file di dati sul disco.

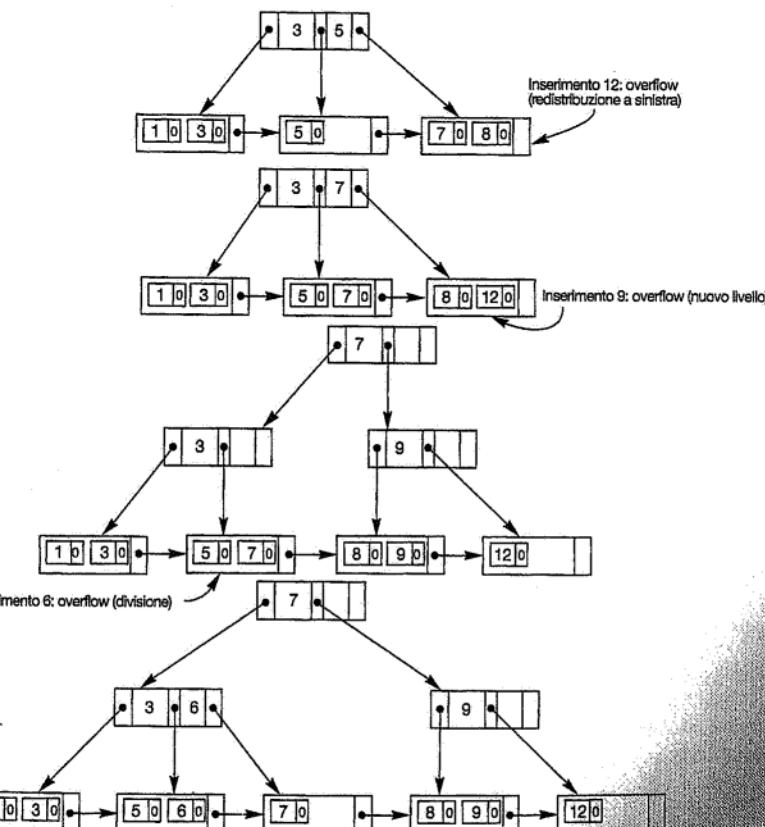
Il vantaggio principale degli indici secondari è che, almeno in teoria, possono essere creati insieme a *qualsiasi organizzazione primaria di record*. Quindi può essere utilizzato un indice secondario in aggiunta agli altri metodi di accesso primario come l'ordinamento o l'*hash* oppure con file misti. Per creare un indice secondario ad albero B^+ su un campo di un file, si devono esaminare tutti i record nel file per creare le voci a livello foglia dell'albero. Queste voci vengono poi ordinate e riempite secondo il fattore di riempimento specificato; simultaneamente vengono creati gli altri livelli dell'indice. È più costoso e molto più difficile creare dinamicamente degli indici primari e degli indici di cluster, perché i record del file di dati devono essere fisicamente ordinati sul disco secondo il campo di indicizzazione. Tuttavia alcuni sistemi consentono agli utenti di creare questi indici dinamicamente, ordinando il file durante la creazione dell'indice.

È un'operazione comune usare un indice per garantire che venga soddisfatto un *vincolo di chiave* su un attributo. Durante la ricerca nell'indice per inserire un nuovo record, è semplice controllare contemporaneamente se un altro record presente nel file, e quindi nell'albero dell'indice, ha lo stesso valore dell'attributo chiave del nuovo record. In caso affermativo l'inserimento può essere rifiutato.

Un file che ha un indice secondario su ognuno dei suoi campi spesso viene chiamato **file trasposto completo**. Visto che tutti gli indici sono secondari, i nuovi record vengono inseriti alla fine del file; perciò il file di dati stesso è un file non ordinato (*heap*). Gli indici di solito sono implementati come alberi B^+ , quindi vengono aggiornati dinamicamente per riflettere

- l'opzione 3 del Sottoparagrafo 6.1.3, con un livello ulteriore di indice per immagazzinare i puntatori ai record. Si supponga che vi siano 1000 valori distinti di NUMERO_DIPARTIMENTO e che i record IMPIEGATO siano distribuiti in modo uniforme lungo questi valori. Si calcoli: (i) i fattori di blocco dell'indice bfr_i (che è anche l'indice fan-out fo); (ii) il numero dei blocchi necessari per l'ulteriore livello di indice che memorizza i puntatori ai record; (iii) il numero di voci e il numero dei blocchi dell'indice del primo livello; (iv) il numero di livelli necessari se si vuole convertirlo in un indice multilivello; (v) il numero totale di blocchi richiesti dall'indice multilivello e i blocchi usati dal livello ulteriore di indice; (vi) il numero approssimativo di accessi necessari per cercare e recuperare tutti i record nel file che hanno un valore specifico di NUMERO_DIPARTIMENTO, usando l'indice.
- f. Si supponga che il file sia ordinato tramite il campo non chiave NUMERO_DIPARTIMENTO e che si voglia costruire un *indice di cluster* su NUMERO_DIPARTIMENTO che utilizzi i punti ancora dei blocchi (ogni nuovo valore di NUMERO_DIPARTIMENTO è posto all'inizio di un blocco). Si supponga che vi siano 1000 valori distinti di NUMERO_DIPARTIMENTO e che i record IMPIEGATO siano distribuiti uniformemente tra questi valori. Si calcoli: (i) il fattore di blocco dell'indice bfr_i (che è anche il fan-out dell'indice); (ii) il numero delle voci e il numero di blocchi dell'indice di primo livello; (iii) il numero di livelli necessari se si vuole passare a un indice multilivello; (iv) il numero totale di blocchi richiesti dall'indice multilivello; (v) il numero di accessi necessari per cercare e recuperare tutti i record del file che hanno uno specifico valore di NUMERO_DIPARTIMENTO usando l'indice di cluster (si supponga che più blocchi relativi allo stesso raggruppamento siano adiacenti).
- g. Si supponga che il file non sia ordinato sulla chiave SSN e che si voglia creare una struttura (indice) di accesso ad albero B⁺ su SSN. Si calcolino: (i) gli ordini p e p_{foglia} dell'albero B⁺; (ii) il numero dei blocchi di livello foglia necessari se i blocchi sono completi approssimativamente al 69 per cento; (iii) il numero di livelli necessari se anche i nodi interni sono completi al 69 per cento; (iv) il numero totale di blocchi richiesti dall'albero B⁺; (v) il numero di accessi necessari per cercare e recuperare un record dal file, dato il suo valore SSN, usando l'albero B⁺.
- h. Si ripeta la parte g, ma per un albero B piuttosto che per un albero B⁺. Si confrontino i risultati per l'albero B e per l'albero B⁺.
- 6.15. Un file PARTS con Part# come campo chiave comprende i record con i seguenti valori di Part#: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16, 20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38. Si supponga che i valori dei campi di ricerca siano inseriti nell'ordine dato in un albero B⁺ di ordine $p = 4$ e $p_{foglia} = 3$; si mostri come l'albero si espanderà e come apparirà l'albero finale.
- 6.16. Si ripeta l'Esercizio 6.15, ma si utilizzi un albero B di ordine $p = 4$ invece di un albero B⁺.
- 6.17. Si supponga che i seguenti valori del campo di ricerca vengano eliminati, nell'ordine dato, dall'albero B⁺ dell'Esercizio 6.15; si mostri come si ridurrà l'albero e l'aspetto finale che avrà. I valori sono: 65, 75, 43, 18, 20, 92, 59, 37.
- 6.18. Si ripeta l'Esercizio 6.17, ma per l'albero B dell'Esercizio 6.16.
- 6.19. L'Algoritmo 6.1 schematizza la procedura di ricerca in un indice primario multilivello non denso per recuperare un record del file. Si adatti l'algoritmo per ciascuno dei seguenti casi.

- a. Un indice secondario multilivello su un campo non di ordinamento non chiave di un file. Si supponga che venga usata l'opzione 3 del Sottoparagrafo 6.1.3, in cui un livello ulteriore di indice memorizza i puntatori ai singoli record con il valore corrispondente del campo di indicizzazione.
- b. Un indice secondario multilivello sul campo chiave (ma non di ordinamento) del file.
- c. Un indice di cluster multilivello su un campo non di ordinamento e non chiave di un file.
- 6.20. Si supponga che esistano più indici secondari su campi non chiave di un file, implementati usando l'opzione 3 del Sottoparagrafo 6.1.3; ad esempio, potrebbero esservi indici secondari sui campi NUMERO_DIPARTIMENTO, CODICE_LAVORO e STIPENDIO del fi-

Figura 6.15 Inserimento nell'albero B⁺ con redistribuzione a sinistra.

le IMPIEGATO dell'Esercizio 6.14. Si descriva un metodo efficace per cercare e recuperare i record che soddisfano una complessa condizione di selezione su questi campi, ad esempio (NUMERO_DIPARTIMENTO = 5 e CODICE_LAVORO = 12 e STIPENDIO = 50.000) usando i puntatori ai record dell'ulteriore livello di indice.

- 6.21. Si adattino gli Algoritmi 6.2 e 6.3, che schematizzano le procedure di ricerca e di inserimento per un albero B+ a un albero B.
- 6.22. È possibile modificare l'algoritmo d'inserimento in un albero B+ per gestire il caso in cui viene creato un nuovo livello individuando una possibile *ridistribuzione* dei valori tra i nodi esterni. In Figura 6.15 viene mostrato come ciò potrebbe essere eseguito per l'esempio della Figura 6.12: invece di dividere il nodo esterno più a sinistra quando viene inserito il valore 12, si esegue una *ridistribuzione a sinistra* spostando 7 nel nodo esterno alla sua sinistra (se c'è spazio in questo nodo). È possibile valutare anche la *ridistribuzione a destra*. Si provi a modificare l'algoritmo di inserimento nell'albero B+ per tener conto della ridistribuzione.
- 6.23. Si schematizzi un algoritmo per l'eliminazione da un albero B+.
- 6.24. Si ripeta l'Esercizio 6.23 per un albero B.

Bibliografia selezionata

Bayer e McCreight (1972) hanno introdotto gli alberi B e gli algoritmi associati. Comer (1979) fornisce i risultati di un'eccellente ricerca sugli alberi B, le loro varianti e la loro storia. Knuth (1973) mette a disposizione un'analisi dettagliata di molte tecniche di ricerca, tra cui gli alberi B e alcune delle loro varianti. Nievergelt (1974) discute l'utilizzo degli alberi di ricerca binaria per l'organizzazione dei file. I manuali sulle strutture dei file, tra cui Wirth (1972), Claybrook (1983), Smith e Barnes (1987), Miller (1987) e Salzberg (1988), trattano in dettaglio l'indicizzazione e possono essere consultati per algoritmi di ricerca, inserimento e cancellazione per alberi B e B+.

Larson (1981) analizza i file sequenziali indicizzati e Held e Stonebraker (1978) confrontano gli indici multilivello statici con gli indici dinamici che sfruttano gli alberi B. Lehman e Yao (1981) e Srinivasan e Carey (1991) hanno fatto un'ulteriore analisi sull'accesso simultaneo agli alberi B. I libri di Wiederhold (1983), Smith e Barnes (1987) e Salzberg (1988) trattano molte delle tecniche di ricerca descritte in questo capitolo. I file a griglia furono introdotti in Nievergelt (1984). Il recupero con confronto parziale che utilizza l'hash partizionato viene analizzato in Burkhard (1976, 1979).

Nuove tecniche e applicazioni degli indici e degli alberi B+ vengono discusse in Lanka e Mays (1991), Zobel e altri (1992) e Faloutsos e Jagadish (1992). Mohan e Narang (1992) si occupano della creazione degli indici. Le prestazioni di vari algoritmi relativi agli alberi B e B+ vengono valutate in Baeza-Yates e Larson (1989) e Johnson e Shasha (1993). La gestione dei buffer per gli indici viene descritta in Chan e altri (1992).

Capitolo 7

Il modello di dati, i vincoli e l'algebra relazionali

Il modello relazionale è stato introdotto nel 1970 da Ted Codd del Centro ricerche IBM in un articolo ormai classico [Codd 1970], e ha subito attirato l'attenzione per la sua semplicità e le sue basi matematiche. Il modello usa il concetto di *relazione matematica* – che assomiglia un po' a una tabella di valori – come suo componente elementare e ha il suo fondamento teorico nella teoria degli insiemi e nella logica dei predicati del primo ordine. In questo capitolo studieremo le caratteristiche di base del modello, i suoi vincoli e l'algebra relazionale, che è un insieme di operazioni per tale modello. Nel corso degli ultimi vent'anni esso è stato implementato in un gran numero di sistemi commerciali.

I modelli di dati che hanno preceduto il modello relazionale – chiamati *sistemi legacy* (lassicati in eredità) – comprendono i modelli gerarchico e reticolare. Essi sono stati proposti negli anni sessanta e sono stati implementati nei primi DBMS durante gli anni settanta e ottanta; ora hanno un'importanza storica, un'ampia base di utenti e saranno utilizzati ancora per molti anni.

In questo capitolo ci concentreremo sulla descrizione dei principi di base del modello relazionale di dati. Cominceremo col definire, nel Paragrafo 7.1, i concetti di modellazione e la notazione, mentre nel Paragrafo 7.2 discuteremo i vincoli relazionali, che sono ora considerati una parte importante del modello relazionale e sono fatti automaticamente rispettare dalla maggior parte dei DBMS relazionali. Nel Paragrafo 7.3 definiremo le operazioni di aggiornamento del modello ed esamineremo come sono trattate le violazioni dei vincoli di integrità.

Nel Paragrafo 7.4 presenteremo un'analisi dettagliata dell'algebra relazionale, che è costituita da un insieme di operazioni per manipolare le relazioni e specificare le interrogazioni. L'algebra relazionale è parte integrante del modello di dati relazionale. Nel Paragrafo 7.5 definiremo altre operazioni relazionali, aggiunte all'algebra relazionale di base per la loro importanza in molte applicazioni di basi di dati. Nel Paragrafo 7.6 forniremo esempi di specificazione di interrogazioni che usano operazioni relazionali. Le stesse interrogazioni saranno usate nei capitoli successivi per illustrare vari linguaggi.

Il lettore interessato a un'introduzione meno particolareggiata dei concetti relazionali può escludere dalla lettura i Sottoparagrafi 7.1.2, 7.4.7 e il Paragrafo 7.5.

7.1 Concetti del modello relazionale

Il modello relazionale rappresenta la base di dati come una collezione di *relazioni*. Informalmente, ogni relazione assomiglia a una tabella di valori o, per certi versi, a un file di record "piatto". Ad esempio, la base di dati di file mostrata in Figura 1.2 è da considerarsi in modo relazionale. Ci sono però importanti differenze tra relazioni e file, come si vedrà poco oltre.

Quando si pensa a una relazione come a una *tabella* di valori, ogni riga della tabella rappresenta una collezione di valori di dati collegati. Nel Capitolo 3 sono stati introdotti i tipi di entità e di associazione come concetti per modellare i dati del mondo reale. Nel modello relazionale ogni riga della tabella rappresenta un fatto che tipicamente corrisponde a un'entità o a un'associazione del mondo reale. Il nome della tabella e i nomi delle colonne sono usati per aiutare a interpretare il significato dei valori presenti in ogni riga. Ad esempio, la prima tabella di Figura 1.2 è chiamata STUDENTE perché ogni riga rappresenta fatti relativi a una specifica entità studente. I nomi delle colonne – Nome, NumeroStudente, AnnoCorso, CorsoLaurea – specificano come interpretare i valori dei dati presenti in ogni riga, sulla base della colonna in cui si trova ogni valore. Tutti i valori presenti in una colonna sono dello stesso tipo di dati.

Nella terminologia formale del modello relazionale, una riga è detta *nupla*, un'intestazione di colonna è detta *attributo* e la tabella è detta *relazione*. Il tipo di dati che descrive i tipi di valori che possono apparire in ogni colonna è detto *dominio*.

7.1.1 Domini, attributi, tuple e relazioni

Un **dominio** D è un insieme di valori atomici. Per atomici si intende che ogni valore nel dominio è indivisibile, per lo meno per quanto riguarda il modello relazionale. Un metodo comune per specificare un dominio consiste nello specificare un tipo di dati, da cui sono tratti i valori dei dati che formano il dominio. È anche utile specificare un nome per il dominio, per aiutare a interpretare i suoi valori. Alcuni esempi di domini sono:

- Numeri_telefonici_USA: l'insieme di numeri telefonici a 10 cifre validi negli Stati Uniti;
- Numeri_telefonici_locali: l'insieme di numeri telefonici a 7 cifre validi all'interno di uno specifico distretto telefonico negli Stati Uniti;
- Numeri_previdenza_sociale: l'insieme di numeri di previdenza sociale a 9 cifre validi;
- Nomi: l'insieme dei nomi di persona;
- Medie_voti: possibili valori delle medie calcolate dei voti; ogni media deve essere un numero reale (a virgola mobile) compreso tra 0 e 4;

- Età_impiegati: possibili età degli impiegati di un'azienda; ogni età deve avere un valore compreso tra 15 e 80 anni;
- Nomi_dipartimenti_universitari: l'insieme dei nomi dei dipartimenti universitari in un'università, ad esempio Informatica, Economia, Fisica;
- Codici_dipartimenti_universitari: l'insieme dei codici dei dipartimenti universitari, come ad esempio INF, ECON e FIS.

Queste definizioni di domini sono dette *logiche*. Per ogni dominio viene anche specificato un **tipo di dati o formato**. Ad esempio: il tipo di dati per il dominio Numeri_telefonici_USA può essere dichiarato come una stringa di caratteri della forma (ddd)ddd-dddd, dove ogni d è una cifra numerica (decimale) e le prime tre cifre formano un prefisso telefonico valido; il tipo di dati per Età_impiegati è un numero intero compreso tra 15 e 80; il tipo di dati di Nomi_dipartimenti_universitari è l'insieme di tutte le stringhe di caratteri che rappresentano nomi validi di dipartimenti. A ogni dominio viene perciò assegnato un nome, un tipo di dati e un formato. Possono anche essere fornite informazioni aggiuntive per interpretare i valori di un dominio; ad esempio, a un dominio numerico come Pesi_persone dovrebbero essere associate le unità di misura – libbre o chilogrammi.

Uno **schema di relazione** R, indicato con $R(A_1, A_2, \dots, A_n)$, è costituito da un nome di relazione R e da un elenco di attributi A_1, A_2, \dots, A_n . Ogni **attributo** A_i fornisce il nome del ruolo interpretato da un certo dominio D nello schema di relazione R. D è detto **dominio** di A_i , ed è indicato con $\text{dom}(A_i)$. Uno schema di relazione è usato per *descrivere* una relazione; R è detto **nome** della relazione. Il **grado** di una relazione è il numero n di attributi presenti nel suo schema di relazione.

Un esempio di uno schema di relazione per una relazione di grado 7, che descrive studenti universitari, è il seguente:

STUDENTE (Nome, SSN, TelefonoDiCasa, Indirizzo, TelefonoUfficio, Età, MV)

STUDENTE è il nome della relazione, che ha sette attributi. È possibile specificare i seguenti domini precedentemente definiti per alcuni degli attributi della relazione STUDENTE: $\text{dom}(\text{Nome}) = \text{Nomi}$; $\text{dom}(\text{SSN}) = \text{Numeri_previdenza_sociale}$; $\text{dom}(\text{TelefonoDiCasa}) = \text{Numeri_telefonici_locali}$; $\text{dom}(\text{TelefonoUfficio}) = \text{Numeri_telefonici_locali}$, e $\text{dom}(\text{MV}) = \text{Medie_voti}$.

Una **relazione** (o **stato di relazione**)¹ r dello schema di relazione $R(A_1, A_2, \dots, A_n)$, indica anche con $r(R)$, è un insieme di n-tuple $r = \{t_1, t_2, \dots, t_m\}$. Ogni **n-tupla** t è un elenco ordinato di n valori $t = \langle v_1, v_2, \dots, v_n \rangle$, dove ogni valore v_i , $1 \leq i \leq n$, è un elemento di $\text{dom}(A_i)$ oppure è uno speciale valore **null**. Il valore i-esimo nella tupla t, che corrisponde all'attributo A_i , è indicato con $t[A_i]$. Sono anche usati comunemente il termine **intensione** di relazione per lo schema R ed **estensione** di relazione per uno stato di relazione r(R).

In Figura 7.1 è riportato un esempio di una relazione STUDENTE corrispondente allo schema STUDENTE specificato sopra. Ogni tupla della relazione rappresenta una particolare entità studente. La relazione è rappresentata graficamente con una tabella, in cui ogni tupla è rap-

¹ Si parla anche di **istanza** di relazione. Qui si ricorrerà a tale locuzione perché *istanza* è anche usata per riferirsi a una singola tupla o riga.

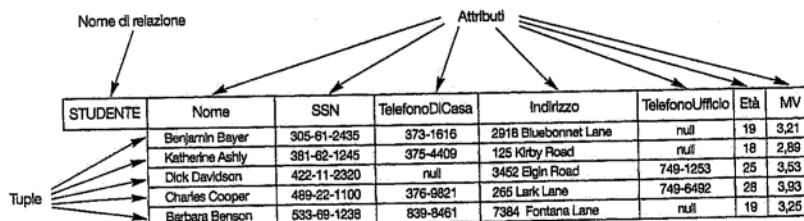


Figura 7.1 Gli attributi e le tuple di una relazione STUDENTE.

presentata con una riga e ogni attributo corrisponde a un'intestazione di colonna che indica un ruolo o un'interpretazione dei valori presenti in quella colonna. I valori null rappresentano attributi i cui valori sono sconosciuti o non esistono per alcune singole tuple di STUDENTE.

La definizione sopra data di relazione può essere *riformulata* nel modo seguente. Una relazione $r(R)$ è una relazione matematica di grado n sui domini $\text{dom}(A_1), \text{dom}(A_2), \dots, \text{dom}(A_n)$, cioè è un sottoinsieme del prodotto cartesiano dei domini che definiscono R :

$$r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n))$$

Il prodotto cartesiano specifica tutte le possibili combinazioni di valori dei domini sottostanti. Perciò, se si indica con $|D|$ il numero di valori o cardinalità di un dominio D , e supponiamo che tutti i domini siano finiti, il numero totale di tuple del prodotto cartesiano è:

$$|\text{dom}(A_1)| * |\text{dom}(A_2)| * \dots * |\text{dom}(A_n)|$$

Al di là di tutte queste possibili combinazioni, uno stato di relazione in un dato istante di tempo – lo stato corrente di relazione – rispecchia solo le tuple valide che rappresentano uno stato particolare del mondo reale. In generale, quando cambia lo stato del mondo reale, cambia anche la relazione, trasformandosi in un altro stato di relazione. Lo schema R è invece relativamente statico e non cambia se non molto raramente – ad esempio, come risultato dell'aggiunta di un attributo, per rappresentare nuove informazioni non memorizzate originariamente nella relazione.

È possibile che più attributi abbiano lo stesso dominio. Gli attributi indicano ruoli, o interpretazioni, diversi per il dominio. Ad esempio, nella relazione STUDENTE, lo stesso dominio Numeri_telefonici_locali svolge il ruolo di TelefonoDiCasa, che fa riferimento al “telefono dello studente”, e il ruolo di TelefonoUfficio, che fa riferimento al “telefono dell'ufficio dello studente”.

7.1.2 Caratteristiche delle relazioni

La definizione di relazione data in precedenza porta con sé certe caratteristiche che rendono una relazione diversa da un file o da una tabella.

STUDENTE	Nome	SSN	TelefonoDiCasa	Indirizzo	TelefonoUfficio	Età	MV
Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3,53	
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3,25	
Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3,93	
Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2,89	
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3,21	

Figura 7.2 La relazione STUDENTE di Figura 7.1, con un ordine diverso delle tuple.

Ordinamento delle tuple in una relazione. Una *relazione* è definita come un insieme di tuple. Da un punto di vista matematico gli elementi di un insieme non hanno *nessun ordine*; perciò le tuple in una relazione non presentano alcun ordine particolare. In un file, invece, i record sono memorizzati fisicamente su disco, cosicché c'è sempre un ordine tra i record. Questo ordinamento indica il primo, il secondo, l' i -esimo e l'ultimo record del file. Analogamente, quando si rappresenta graficamente una relazione con una tabella, le righe sono mostrate in un certo ordine.

L'ordinamento delle tuple non fa parte della definizione di una relazione, perché una relazione tenta di rappresentare fatti a livello logico o astratto. In una relazione possono essere specificati molti ordini logici; ad esempio, le tuple nella relazione STUDENTE di Figura 7.1 potrebbero essere logicamente ordinate per valori di Nome, SSN, Età o qualche altro attributo. La definizione di relazione non specifica alcun ordine: non c'è *nessuna preferenza* per un ordinamento logico rispetto a un altro. Perciò la relazione rappresentata graficamente in Figura 7.2 è considerata *identica* a quella mostrata in Figura 7.1. Quando una relazione viene implementata con un file, può essere specificato un ordinamento fisico sui record del file.

Ordinamento dei valori all'interno di una tupla e definizione alternativa di relazione. Secondo la definizione precedente di relazione, una n -tupla è un elenco ordinato di n valori, cosicché l'ordinamento dei valori in una tupla – e perciò degli attributi nella definizione di uno schema di relazione – è importante. Comunque, a livello logico, l'ordine degli attributi e dei loro valori *non* è realmente importante, purché sia mantenuta la corrispondenza tra attributi e valori.

Può essere data una **definizione alternativa** di relazione, che rende *non necessario* l'ordinamento dei valori in una tupla. In questa definizione uno schema di relazione $R = \{A_1, A_2, \dots, A_n\}$ è un insieme di attributi, e una relazione $r(R)$ è un insieme finito di trasformazioni $r = \{t_1, t_2, \dots, t_m\}$, dove ogni tupla t_i è una trasformazione da R a D , e D è l'unione dei domini degli attributi; cioè $D = \text{dom}(A_1) \cup \text{dom}(A_2) \cup \dots \cup \text{dom}(A_n)$. In questa definizione, $t[A_i]$ deve essere in $\text{dom}(A_i)$ per $1 \leq i \leq n$, qualsiasi sia la trasformazione t di r . Ogni trasformazione t_i è detta tupla.

Secondo questa definizione una tupla può essere considerata un insieme di coppie (\langle attributo \rangle, \langle valore \rangle), e ogni coppia fornisce il valore della trasformazione da un attributo A_i a un valore v_i del $\text{dom}(A_i)$. L'ordinamento degli attributi *non* è importante, perché con il valore dell'attributo compare il suo nome. Secondo questa definizione, le due tuple mostrate in Figura 7.3 sono identiche. Ciò ha senso a livello logico o astratto, dato che non c'è veramente nessuna ragione per preferire che in una tupla il valore di un attributo si presenti prima di un altro.



$t = \langle (\text{Nome}, \text{Dick Davidson}), (\text{SSN}, 422-11-2320), (\text{TelefonoDiCasa}, \text{null}), (\text{Indirizzo}, 3452 \text{ Elgin Road}), (\text{TelefonoUfficio}, 749-1253), (\text{Età}, 25), (\text{MV}, 3,53) \rangle$

$t = \langle (\text{Indirizzo}, 3452 \text{ Elgin Road}), (\text{Nome}, \text{Dick Davidson}), (\text{SSN}, 422-11-2320), (\text{Età}, 25), (\text{TelefonoUfficio}, 749-1253), (\text{CPA}, 3,53), (\text{TelefonoDiCasa}, \text{null}) \rangle$

Figura 7.3 Due tuple identiche quando l'ordine degli attributi e dei valori non fa parte della definizione di relazione.

Quando una relazione è implementata con un file, gli attributi sono ordinati fisicamente come campi di un record. Qui verrà usata la **prima definizione** di relazione, in cui gli attributi e i valori all'interno delle tuple *sono ordinati*, perché essa semplifica di molto la notazione. La definizione alternativa è però più generale.

Valori nelle tuple. Ogni valore in una tupla è un valore **atomico**, cioè non è divisibile in parti componenti, usando la struttura del modello relazionale di base. Perciò, gli attributi composti e multivalore (si veda il Capitolo 3) non sono consentiti. Molta della teoria che sta dietro il modello relazionale è stata sviluppata tenendo presente questa assunzione, detta assunzione di **prima forma normale**.² Gli attributi multivalore devono essere rappresentati tramite relazioni separate, e gli attributi composti sono rappresentati solo dagli attributi che ne costituiscono le componenti semplici. Recentissime ricerche sul modello relazionale tentano di rimuovere queste restrizioni usando il concetto di relazione **non in prima forma normale** o relazione **nested** (nidificata).

I valori di alcuni attributi all'interno di una specifica tupla possono essere sconosciuti o possono non essere pertinenti per quella tupla. In questi casi si usa un valore speciale, detto **null**. In Figura 7.1, ad esempio, alcune tuple studente presentano valori nulli per i rispettivi telefoni d'ufficio, perché questi studenti non hanno un ufficio (ossia il telefono d'ufficio *non è pertinente* per questi studenti). Un altro studente ha un valore nullo per il telefono di casa, presumibilmente perché a casa non ha telefono oppure perché ce l'ha ma non ne è noto il numero (il valore è *sconosciuto*). In generale si possono avere *molti tipi* di valori nulli, come ad esempio "valore sconosciuto", "valore esistente ma non disponibile", o "attributo non pertinente per questa tupla". È possibile inventare codici diversi per diversi tipi di valori nulli. L'inserimento di diversi tipi di valori nulli nelle operazioni del modello relazionale si è rivelato difficile, e un'analisi completa va al di là dello scopo di questo libro.

Interpretazione di una relazione. Lo schema di relazione può essere interpretato come una dichiarazione, ossia un tipo di **asserzione**. Ad esempio, lo schema della relazione STUDENTE di Figura 7.1 asserisce che, in generale, un'entità studente ha un Nome, un SSN, un Telefono-DiCasa, un Indirizzo, un TelefonoUfficio, un'Età e una MV. Ogni tupla nella relazione può però essere interpretata come un **fatto** o una particolare istanza dell'asserzione. Per esempio, la

prima tupla asserisce il fatto che c'è uno STUDENTE con nome Benjamin Bayer, SSN 305-61-2435, Età 19 e così via.

Si noti che alcune relazioni possono rappresentare fatti su *entità*, mentre altre relazioni possono rappresentare fatti su *associazioni*. Ad esempio, uno schema di relazione SPEC_PRINCIPALI (SSNStudente, CodiceDipartimento) asserisce che studenti si specializzano in dipartimenti universitari; una tupla in questa relazione collega uno studente al suo dipartimento di specializzazione principale. Quindi, il modello relazionale rappresenta *uniformemente*, come relazioni, fatti su entità e su associazioni.

Un'interpretazione alternativa di uno schema di relazione consiste nel vederlo come un **predicato**; in questo caso i valori in ogni tupla sono interpretati come valori che *soddisfano* il predicato. Questa interpretazione è piuttosto utile nel contesto dei linguaggi di programmazione logica, come il PROLOG, perché consente di usare il modello relazionale con questi linguaggi.

7.1.3 Notazione del modello relazionale

Nella nostra presentazione verrà usata la notazione seguente:

- uno schema di relazione R di grado n è indicato con $R(A_1, A_2, \dots, A_n)$;
- una n -tuple t di una relazione $r(R)$ è indicata con $t = \langle v_1, v_2, \dots, v_n \rangle$, dove v_i è il valore corrispondente all'attributo A_i . La notazione seguente si riferisce ai **valori componenti** delle tuple:
 - sia $t[A]$ sia $t.A$, si riferiscono al valore v_i di t per l'attributo A_i ;
 - sia $t[A_1, A_2, \dots, A_z]$ sia $t.(A_1, A_2, \dots, A_z)$, dove A_1, A_2, \dots, A_z è una lista di attributi di R , si riferiscono alla sottotupla di t di valori $\langle v_1, v_2, \dots, v_z \rangle$ corrispondenti agli attributi specificati nella lista;
 - le lettere Q, R e S indicano nomi di relazioni;
 - le lettere q, r, s indicano stati di relazione;
 - le lettere t, u, v indicano tuple.

In generale, il nome di uno schema di relazione, come ad esempio STUDENTE, *indica anche* l'insieme corrente di tuple in quella relazione – lo *stato corrente di relazione* – mentre STUDENTE (Nome, SSN, ...) fa riferimento *solo* allo schema di relazione;

un attributo A può essere contraddistinto con il nome di relazione cui appartiene usando la **notazione punto** (*dot notation*) $R.A$ – ad esempio, STUDENTE.Nome o STUDENTE.Età. Questo perché lo stesso nome può essere usato per due attributi presenti in relazioni diverse.

In una specifica relazione, invece, tutti i nomi di attributo devono essere diversi.

Ad esempio, si consideri la tupla $t = \langle \text{'Barbara Benson'}, '533-69-1238', '839-8461', \text{'Fontana Lane'}, \text{null}, 19, 3,25 \rangle$ della relazione STUDENTE di Figura 7.1; si ha $t[\text{Nome}] = \text{'Barbara Benson'}$ e $t[\text{SSN}, \text{MV}, \text{Età}] = \langle '533-69-1238', 3,25, 19 \rangle$.

² Questa assunzione verrà analizzata in maggior dettaglio nel Capitolo 14.

parte, attributi che rappresentano concetti diversi possono avere lo stesso nome in relazioni diverse. Ad esempio, si potrebbe aver usato il nome di attributo NOME sia per NOME_P di PROGETTO sia per NOME_D di DIPARTIMENTO: in questo caso si avrebbero due attributi che condividono lo stesso nome ma rappresentano concetti diversi del mondo reale – nomi di progetti e nomi di dipartimenti.

In qualche versione iniziale del modello relazionale è stata fatta l'assunzione che lo stesso concetto del mondo reale, quando rappresentato da un attributo, debba avere nomi di attributo *identici* in tutte le relazioni. Ciò crea problemi quando lo stesso concetto del mondo reale è usato con ruoli (significati) diversi nella stessa relazione. Ad esempio, il concetto di numero di previdenza sociale compare due volte nella relazione IMPiegato di Figura 7.5: una volta nel ruolo di numero di previdenza sociale dell'impiegato e una volta nel ruolo di numero di previdenza sociale del supervisore. Qui sono stati dati loro nomi diversi di attributo – SSN e SUPERSSN, rispettivamente – per poter distinguere il rispettivo significato.

Ogni DBMS relazionale deve avere un linguaggio di definizione di dati (DDL: Data Definition Language) per definire uno schema di base di dati relazionale. Generalmente i DBMS relazionali correnti usano a questo scopo SQL. Qui si presenterà SQL come DDL nel Paragrafo 8.1.

I vincoli di integrità vengono specificati su uno schema di base di dati e ci si aspetta che valgano su ogni stato di base di dati di quello schema. Oltre ai vincoli sul dominio e ai vincoli di chiave, altri due tipi di vincoli sono considerati parte del modello relazionale: integrità dell'entità e integrità referenziale.

7.2.4 Integrità dell'entità, integrità referenziale e chiavi esterne

Il vincolo di integrità dell'entità stabilisce che nessun valore di chiave primaria può essere nullo. Questo perché il valore di chiave primaria è usato per identificare le singole tuple di una relazione; avere valori nulli per la chiave primaria comporta il non poter identificare alcune tuple. Ad esempio, se due o più tuple avessero un valore nullo in corrispondenza alle loro chiavi primarie, si potrebbe non essere in grado di distinguerle.

I vincoli di chiave e i vincoli di integrità dell'entità sono specificati sulle singole relazioni. Il vincolo di integrità referenziale è specificato fra due relazioni ed è usato per mantenere la consistenza fra tuple delle due relazioni. Informalmente, il vincolo di integrità referenziale stabilisce che una tupla in una relazione che fa riferimento a un'altra relazione deve far riferimento a una *tupla esistente* in quella relazione. Ad esempio, in Figura 7.6, l'attributo N_D riferito a IMPiegato fornisce il numero del dipartimento per il quale ogni impiegato lavora; perciò il suo valore in ogni tupla IMPiegato deve accordarsi con il valore di NUMERO_D di una qualche tupla nella relazione DIPARTIMENTO.

Per definire più formalmente l'integrità referenziale, definiamo dapprima il concetto di *chiave esterna*. Le condizioni di chiave esterna, date sotto, specificano un vincolo di integrità referenziale tra i due schemi di relazione R_1 e R_2 . Un insieme di attributi FK (foreign key) nel schema di relazione R_1 è una *chiave esterna* di R_1 , che riferisce la relazione R_2 se soddisfa le seguenti due regole:

1. gli attributi presenti in FK hanno gli stessi domini degli attributi di chiave primaria PK (*primary key*) di R_2 ; si dice che gli attributi FK riferiscono o fanno riferimento alla relazione R_2 ;
2. un valore di FK in una tupla t_1 dello stato corrente $r_1(R_1)$ o è presente come valore di PK in una qualche tupla t_2 nello stato corrente $r_2(R_2)$ o è nullo. Nel primo caso si ha $t_1[FK] = t_2[PK]$ e si dice che la tupla t_1 riferisce o fa riferimento alla tupla t_2 . R_1 è detta *relazione riferente* (ossia che fa riferimento) e R_2 *relazione riferita* (o alla quale si fa riferimento).

In una base di dati con molte relazioni ci sono generalmente molti vincoli di integrità referenziale. Per specificare questi vincoli occorre prima di tutto avere una chiara comprensione del significato, o ruolo, proprio di ogni insieme di attributi nei vari schemi di relazione della base di dati. I vincoli di integrità referenziale nascono tipicamente dalle *associazioni fra le entità* rappresentate dagli schemi di relazione. Ad esempio, si consideri la base di dati mostrata in Figura 7.6. Nella relazione IMPiegato, l'attributo N_D fa riferimento al dipartimento per il quale l'impiegato lavora; perciò si designa N_D come chiave esterna di IMPiegato, che fa riferimento alla relazione DIPARTIMENTO. Ciò significa che un valore di N_D in ogni tupla t_1 della relazione IMPiegato deve accordarsi a un valore della chiave primaria di DIPARTIMENTO – l'attributo NUMERO_D – presente in qualche tupla t_2 della relazione DIPARTIMENTO, oppure il valore di N_D può essere nullo se l'impiegato non fa parte di nessun dipartimento. In figura la tupla per l'impiegato 'John Smith' riferisce la tupla del dipartimento 'Ricerca', indicando così che 'John Smith' lavora per questo dipartimento.

Si noti che una chiave esterna può *far riferimento alla sua stessa relazione di appartenenza*. Ad esempio, l'attributo SUPERSSN in IMPiegato fa riferimento al supervisore di un IMPiegato; questo è un altro impiegato, rappresentato da una tupla della relazione IMPiegato. Perciò SUPERSSN è una chiave esterna che riferisce la stessa relazione IMPiegato. In figura la tupla per l'impiegato 'John Smith' riferisce la tupla per l'impiegato 'Franklin Wong', indicando così che 'Franklin Wong' è il supervisore di 'John Smith'.

È possibile *rappresentare diagrammaticamente i vincoli di integrità referenziale* disegnando un arco orientato da ciascuna chiave esterna verso la relazione che essa riferisce. Per chiarezza la punta della freccia può puntare alla chiave primaria della relazione riferita. In Figura 7.7 viene mostrato lo schema di Figura 7.5 con i vincoli di integrità referenziale rappresentati graficamente in questo modo.

Sullo schema di base di dati relazionale dovrebbero essere specificati tutti i vincoli di integrità, se si desidera imporre questi vincoli sugli stati della base di dati. Perciò il DDL comprende strumenti per specificare i vari tipi di vincoli in modo tale che il DBMS possa automaticamente importarli. La maggior parte dei DBMS relazionali supporta i vincoli di chiave e di integrità dell'entità, e aiuta a supportare l'integrità referenziale. La specificazione di questi vincoli fa parte della definizione dei dati.

I vincoli di integrità precedentemente visti non comprendono un'ampia classe di vincoli generali, detti talora *vincoli di integrità semanticci*, che qualche volta devono essere specificati e imposti su una base di dati relazionale. Esempi di questi vincoli sono "lo stipendio di un impiegato non dovrebbe superare lo stipendio del supervisore dell'impiegato" e "il numero massimo di ore settimanali che un impiegato può lavorare su tutti i progetti è 56". Questi vincoli possono essere specificati e imposti usando un *linguaggio di specificazione dei vincoli di uso generale* (*general purpose*). Possono essere usati meccanismi detti trigger e asserzioni. In SQL2 si usa a questo scopo il comando CREATE ASSERTION (si veda il Capitolo 8).

ficare le richieste di recupero fondamentali. Il risultato del recupero è una nuova relazione, che può essere stata formata a partire da una o più relazioni. Le operazioni dell'algebra, perciò, producono nuove relazioni, le quali possono essere ulteriormente manipolate usando operazioni della stessa algebra. Una sequenza di operazioni di algebra relazionale forma un'**espressione dell'algebra relazionale**, il cui risultato è ancora una relazione.

Le operazioni dell'algebra relazionale sono generalmente divise in due gruppi. Un gruppo comprende operazioni insiemistiche proprie della teoria matematica degli insiemi; esse sono utilizzabili perché per definizione ogni relazione è un insieme di tuple e comprendono UNIONE, INTERSEZIONE, DIFFERENZA e PRODOTTO CARTESIANO. L'altro gruppo è composto da operazioni specificamente sviluppate per le basi di dati relazionali; esse comprendono, tra l'altro, SELEZIONE, PROIEZIONE e JOIN (GIUNZIONE). Si examineranno qui per prime le operazioni di SELEZIONE e PROIEZIONE, perché sono le più semplici, quindi si passerà alle operazioni insiemistiche e infine al JOIN e ad altre operazioni complesse. Per gli esempi verrà usata la base di dati relazionale mostrata in Figura 7.6.

Alcune comuni richieste alla base di dati non possono essere portate a termine con le operazioni di base dell'algebra relazionale e richiedono operazioni aggiuntive (si veda il Paragrafo 7.5).

7.4.1 L'operazione di SELEZIONE

L'operazione di SELEZIONE è usata per selezionare un *sottoinsieme* di tuple di una relazione che soddisfa una **condizione di selezione**. Si può considerare tale operazione come un *filtro* che trattiene solo quelle tuple che soddisfano una condizione qualificante. Ad esempio, per selezionare le tuple di IMPiegato il cui dipartimento è 4, oppure quelle il cui stipendio supera i 30.000 dollari, si può specificare individualmente ognuna di queste due condizioni con un'operazione di SELEZIONE nel modo seguente:

$$\sigma_{N_D=4}(\text{IMPIEGATO})$$

$$\sigma_{\text{STIPENDIO}>30000}(\text{IMPIEGATO})$$

In generale l'operazione è indicata con:

$$\sigma_{\langle\text{condizione di selezione}\rangle}(R)$$

dove il simbolo σ (sigma) è usato per denotare l'operatore di SELEZIONE, e la condizione di selezione è un'espressione booleana specificata sugli attributi della relazione R . Si noti che R è generalmente un'**espressione dell'algebra relazionale** il cui risultato è una relazione; l'espressione più semplice è costituita soltanto dal nome di una relazione della base di dati. La relazione risultante dall'operazione di SELEZIONE ha gli *stessi attributi* di R . L'espressione booleana specificata in **<condizione di selezione>** è costituita da un certo numero di clausole di forma:

<nome di attributo><op di confronto><valore costante>, o

<nome di attributo><op di confronto><nome di attributo>

dove **<nome di attributo>** è il nome di un attributo di R , **<op di confronto>** è di solito uno degli operatori $\{=, <, \leq, >, \geq, \neq\}$, e **<valore costante>** è un valore costante del dominio dell'attributo. Le clausole possono essere unite arbitrariamente tramite gli operatori booleani AND, OR, e NOT per formare una condizione di selezione generale. Ad esempio, per selezionare le tuple relative a tutti gli impiegati che lavorano nel dipartimento 4 e guadagnano più di 25.000 dollari all'anno, oppure che lavorano nel dipartimento 5 e guadagnano più di 30.000 dollari, è possibile specificare la seguente operazione di SELEZIONE:

$$\sigma_{(N_D=4 \text{ AND STIPENDIO}>25000) \text{ OR } (N_D=5 \text{ AND STIPENDIO}>30000)}(\text{IMPIEGATO})$$

Il risultato è mostrato in Figura 7.8(a). Si noti che gli operatori di confronto nell'insieme $\{=, <, \leq, >, \geq, \neq\}$ si applicano ad attributi i cui domini sono costituiti da *valori ordinati*, come ad esempio domini numerici o costituiti da date. I domini di stringhe di caratteri sono considerati ordinati sulla base del confronto ordinato dei caratteri della sequenza. Se il dominio di un attributo è un insieme di *valori non ordinati*, allora possono essere usati solamente gli operatori di confronto che fanno parte dell'insieme $\{=, \neq\}$. Un esempio di dominio non ordinato è il dominio Colore = {rosso, blu, verde, bianco, giallo, ...} in cui non è specificato nessun ordine fra i vari colori. Alcuni domini consentono tipi supplementari di operatori di confronto; ad esempio, un dominio di stringhe di caratteri può consentire l'operatore di confronto SOTTOSTRINGA_DI.

In generale, il risultato di un'operazione di SELEZIONE può essere determinato come segue. La **<condizione di selezione>** è applicata indipendentemente a ciascuna tupla t di R , sostituendo ogni occorrenza di un attributo A_i nella condizione di selezione con il suo valore nella tupla $t[A_i]$. Se la condizione è valutata vera, allora la tupla t viene **selezionata**. Nel risulta-

NOME_BATT	INIZ_INT	COGNOME	SSN	DATA_N	INDIRIZZO	SESSO	STIPENDIO	SUPERSSN	N_D
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1982-09-15	975 FireOak-Humble,TX	M	38000	333445555	5

COGNOME	NOME_BATT	STIPENDIO
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabber	Ahmad	25000
Borg	James	55000

SESSO	STIPENDIO
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Figura 7.8 Risultati delle operazioni di SELEZIONE e PROIEZIONE. (a) $\sigma_{(N_D=4 \text{ AND STIPENDIO}>25000) \text{ OR } (N_D=5 \text{ AND STIPENDIO}>30000)}(\text{IMPIEGATO})$. (b) $\pi_{\text{NOME_BATT}, \text{COGNOME}, \text{STIPENDIO}}(\text{IMPIEGATO})$. (c) $\pi_{\text{SESSO}, \text{STIPENDIO}}(\text{IMPIEGATO})$.

(a) $\pi_{\text{NOME_BATT}, \text{COGNOME}, \text{STIPENDIO}}(\sigma_{\text{N_D}=5}(\text{IMPIEGATO}))$			
John	Smith	30000	
Franklin	Wong	40000	
Ramesh	Narayan	38000	
Joyce	English	25000	

(b) La stessa espressione con l'uso di relazioni intermedie e della ridefinizione degli attributi.										
TEMP	NOME_BATT	INIZ_INT	COGNOME	SSN	DATA_N	INDRIZZO	SESSO	STIPENDIO	SUPERSSN	N.D
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5	
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5	
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5	
Joyce	A	English	453453453	1972-07-31	5631 Ries, Houston, TX	F	25000	333445555	5	

R	NOME_DI_BATTESIMO	NOME_DI_FAMIGLIA	STIPENDIO
John	Smith	30000	
Franklin	Wong	40000	
Ramesh	Narayan	38000	
Joyce	English	25000	

Figura 7.9 Risultati di espressioni di algebra relazionale. (a) $\pi_{\text{NOME_BATT}, \text{COGNOME}, \text{STIPENDIO}}(\sigma_{\text{N_D}=5}(\text{IMPIEGATO}))$. (b) La stessa espressione con l'uso di relazioni intermedie e della ridefinizione degli attributi.

ogni relazione intermedia:

$\text{IMP_DIP5} \leftarrow \sigma_{\text{N_D}=5}(\text{IMPIEGATO})$

$\text{RISULTATO} \leftarrow \pi_{\text{NOME_BATT}, \text{COGNOME}, \text{STIPENDIO}}(\text{IMP_DIP5})$

Spesso è più semplice suddividere una sequenza complessa di operazioni, specificando relazioni che forniscono un risultato intermedio, piuttosto che scrivere una singola espressione di algebra relazionale. È possibile usare questa tecnica anche per ridefinire gli attributi di relazioni intermedie e del risultato. Come si vedrà in seguito, ciò può essere utile con operazioni più complesse, come ad esempio l'UNIONE e il JOIN. Per ridefinire gli attributi di una relazione, si elencano semplicemente i nomi dei nuovi attributi tra parentesi, come nell'esempio seguente:

$\text{TEMP} \leftarrow \sigma_{\text{N_D}=5}(\text{IMPIEGATO})$

$\text{R}(\text{NOME_DI_BATTESIMO}, \text{NOME_DI_FAMIGLIA}, \text{STIPENDIO}) \leftarrow \pi_{\text{NOME_BATT}, \text{COGNOME}, \text{STIPENDIO}}(\text{TEMP})$

Le due operazioni sopra viste sono illustrate in Figura 7.9(b). Se non è stata eseguita nessuna ridefinizione, i nomi degli attributi nella relazione risultante di un'operazione di SELEZIONE sono uguali a quelli della relazione originale e sono presenti nello stesso ordine. Per un'operazione di PROIEZIONE senza ridefinizione, la relazione risultante ha nomi di attributi uguali a quelli presenti nella lista della proiezione e nello stesso ordine con cui compaiono nella lista.

È possibile anche definire un'operazione di RIDENOMINAZIONE – che può ridefinire i nomi di una relazione, o i nomi degli attributi, o entrambi – in modo analogo a quello

con cui si sono definite la SELEZIONE e la PROIEZIONE. L'operazione di RIDENOMINAZIONE generale, quando applicata a una relazione R di grado n , è denotata con

$\rho_{S(B_1, B_2, \dots, B_n)}(R)$ o $\rho_S(R)$ o $\rho_{(B_1, B_2, \dots, B_n)}(R)$

dove il simbolo ρ (rho) è usato per indicare l'operazione di RIDENOMINAZIONE, S è il nuovo nome della relazione, e B_1, B_2, \dots, B_n sono i nuovi nomi degli attributi. La prima espressione dà un nome nuovo sia alla relazione sia ai suoi attributi, la seconda solo alla relazione, e la terza solo agli attributi. Se gli attributi di R sono (A_1, A_2, \dots, A_n) in quest'ordine, allora ogni A_i è ridefinito come B_i .

7.4.4 Operazioni insiemistiche

Il gruppo successivo di operazioni di algebra relazionale è costituito dalle usuali operazioni matematiche sugli insiemi. Ad esempio, per recuperare i numeri di previdenza sociale di tutti gli impiegati che lavorano nel dipartimento 5 o sorvegliano direttamente un impiegato che lavora nel dipartimento 5, si può usare l'operazione di UNIONE nel modo seguente:

$\text{IMP_DIP5} \leftarrow \sigma_{\text{N_D}=5}(\text{IMPIEGATO})$

$\text{RISULTATO1} \leftarrow \pi_{\text{SSN}}(\text{IMP_DIP5})$

$\text{RISULTATO2}(\text{SSN}) \leftarrow \pi_{\text{SUPERSSN}}(\text{IMP_DIP5})$

$\text{RISULTATO} \leftarrow \text{RISULTATO1} \cup \text{RISULTATO2}$

La relazione RISULTATO1 contiene i numeri di previdenza sociale di tutti gli impiegati che lavorano nel dipartimento 5, mentre RISULTATO2 contiene i numeri di previdenza sociale di tutti gli impiegati che sorvegliano direttamente un impiegato che lavora nel dipartimento 5. L'operazione di UNIONE fornisce le tuple che sono o in RISULTATO1 o in RISULTATO2 o in entrambi (Figura 7.10).

Molte operazioni sugli insiemi vengono usate per fondere gli elementi di due insiemi in

RISULTATO1	SSN	RISULTATO2	SSN	RISULTATO	SSN
	123456789		333445555		123456789
	333445555		888665555		333445555
	666884444				666884444
	453453453				453453453
					888665555

Figura 7.10 Risultato di un'interrogazione dopo l'operazione di UNIONE: $\text{RISULTATO} \leftarrow \text{RISULTATO1} \cup \text{RISULTATO2}$.

modi diversi; fra queste, in particolare, l'**UNIONE**, l'**INTERSEZIONE** e la **DIFFERENZA**. Si tratta di operazioni binarie, cioè ciascuna di esse si applica a due insiemi. Quando queste operazioni sono adattate a basi di dati relazionali, le due relazioni su cui è eseguita ognuna delle tre operazioni sopra viste deve avere lo stesso tipo di tuple; questa condizione è detta *compatibilità all'unione*. Si dice che due relazioni $R(A_1, A_2, \dots, A_n)$ e $S(B_1, B_2, \dots, B_m)$ sono *compatibili all'unione* se hanno lo stesso grado n e se $\text{dom}(A_i) = \text{dom}(B_i)$ per $1 \leq i \leq n$. Ciò significa che le due relazioni hanno lo stesso numero di attributi e che ogni coppia di attributi corrispondenti ha lo stesso dominio.

È possibile definire queste tre operazioni su due relazioni R e S compatibili all'unione nel modo seguente:

- **UNIONE**: il risultato di questa operazione, indicato con $R \cup S$, è una relazione che comprende tutte le tuple che sono in R o in S o sia in R sia in S . Le tuple duplicate vengono eliminate;
- **INTERSEZIONE**: il risultato, indicato con $R \cap S$, è una relazione che comprende tutte le tuple che sono sia in R sia in S ;
- **DIFFERENZA**: il risultato, indicato con $R - S$, è una relazione che comprende tutte le tuple che sono in R ma non in S .

Si adotterà qui la convenzione secondo la quale la relazione risultante ha gli stessi nomi di attributi della *prima* relazione R . In Figura 7.11 sono illustrate le tre operazioni. Le relazioni STUDENTE e ASSISTENTE in Figura 7.11(a) sono compatibili all'unione, e le loro tuple rappresentano i nomi di studenti e assistenti, rispettivamente. Il risultato dell'operazione di UNIONE in Figura 7.11(b) mostra i nomi di tutti gli studenti e gli assistenti. Il risultato dell'operazione di INTERSEZIONE (Figura 7.11c) contiene solo coloro che sono sia studenti sia assistenti. Si noti che sia l'UNIONE sia l'INTERSEZIONE sono *operazioni commutative*, cioè

$$R \cup S = S \cup R \text{ e } R \cap S = S \cap R$$

Sia l'unione sia l'intersezione possono essere trattate come operazioni n -arie applicabili a qualsiasi numero di relazioni dal momento che entrambe sono *operazioni associative*, cioè:

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ e } (R \cap S) \cap T = R \cap (S \cap T)$$

L'operazione di DIFFERENZA è *non commutativa*, cioè, in generale

$$R - S \neq S - R$$

In Figura 7.11(d) sono mostrati i nomi degli studenti che non sono assistenti, e in Figura 7.11(e) quelli degli assistenti che non sono studenti.

Si consideri ora l'operazione di **PRODOTTO CARTESIANO** – noto anche come **PRODOTTO INCROCIATO (CROSS PRODUCT)** o **JOIN INCROCIATO (CROSS JOIN)** – indicato con \times ; si tratta ancora di un'operazione insiemistica binaria, ma le relazioni alle quali si applica *non* devono necessariamente essere compatibili all'unione. Questa operazione è utilizzata per unire tuple prese da due relazioni in modo combinatorio. In generale, il risultato

(a)		
STUDENTE	N_B	CO
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

(b)	
N_B	CO
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)	
N_B	CO
Susan	Yao
Ramesh	Shah

(d)	
N_B	CO
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)	
NOME_BATT	COGNOME
John	Smith
Ricardo	Browne
Francis	Johnson

Figura 7.11 Illustrazione delle operazioni insiemistiche UNIONE, INTERSEZIONE e DIFFERENZA. (a) Due relazioni compatibili all'unione. (b) STUDENTE \cup ASSISTENTE. (c) STUDENTE \cap ASSISTENTE. (d) STUDENTE - ASSISTENTE. (e) ASSISTENTE - STUDENTE.

di $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$ è una relazione Q con $n + m$ attributi $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, presi in quest'ordine. La relazione risultante Q presenta una tupla per ogni combinazione di tuple – una di R e una di S . Perciò, se R ha n_R tuple e S ne ha n_S , allora $R \times S$ avrà $n_R \times n_S$ tuple. L'operazione applicata da sola è generalmente priva di significato. È invece utile quando è seguita da una selezione che accoppia i valori degli attributi provenienti dalle relazioni componenti. Ad esempio, si supponga di voler recuperare per ogni impiegato di sesso femminile un elenco dei nomi delle persone a suo carico:

$\text{IMP_SESSO_FEMM} \leftarrow \sigma_{\text{SESSO}='F'}(\text{IMPIEGATO})$

$\text{NOMI_IMP} \leftarrow \pi_{\text{NOME_BATT}, \text{COGNOME}, \text{SSN}}(\text{IMP_SESSO_FEMM})$

$\text{IMP_PERSONE_A_CARICO} \leftarrow \text{NOMI_IMP} \times \text{PERSONA_A_CARICO}$

$$\text{PERSONE_A_CARICO_EFFETTIVE} \leftarrow \sigma_{\text{SSN}=\text{SSN}_I}(\text{IMP_PERSONE_A_CARICO})$$

$$\text{RISULTATO} \leftarrow \pi_{\text{NOME_BATT}, \text{COGNOME}, \text{NOME_PERSONA_A_CARICO}}(\text{PERSONE_A_CARICO_EFFETTIVE})$$

Le relazioni risultanti dalla sequenza di operazioni vista sopra sono mostrate in Figura 7.12. La relazione **IMP_PERSONE_A_CARICO** è il risultato dell'applicazione dell'operazione di PRODOTTO CARTESIANO a **NOMI_IMP** di Figura 7.12 con **PERSONE_A_CARICO** di Figura 7.6. In **IMP_PERSONE_A_CARICO**, ogni tupla di **NOMI_IMP** si combina con ogni tupla di **PERSONA_A_CARICO**, dando un risultato che non è molto significativo. Si vuole invece qui com-

IMP_SESSO_FEMMINILE	NOME_BATT	INIZ_INT	COGNOME	SSN	DATA_N	INDIRIZZO	SESSO	STIPENDIO	SUPERSSN	N_D
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle,Spring,TX	F	25000	987654321	4	
Jennifer	S	Wallace	987654321	1941-05-20	291 Berry,Belair,TX	F	43000	888665555	4	
Joyce	A	English	453453453	1972-07-31	5531 Rice,Houston,TX	F	25000	333445555	5	

NOMI-IMP	NOME_BATT	COGNOME	SSN
Alicia	Zelaya	999887777	
Jennifer	Wallace	987654321	
Joyce	English	453453453	

IMP_PERSONE_A_CARICO	NOME_BATT	COGNOME	SSN	SSN_I	NOME_PERSONA_A_CARICO	SESSO	DATA_N	***
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	***	
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	***	
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	***	
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	***	
Alicia	Zelaya	999887777	987654321	Michael	M	1988-01-04	***	
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	***	
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	***	
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	***	
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	***	
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	***	
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	***	
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	***	
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	***	
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	***	
Joyce	English	453453453	333445555	Alice	F	1986-04-05	***	
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	***	
Joyce	English	453453453	333445555	Joy	F	1958-05-03	***	
Joyce	English	453453453	987654321	Abner	M	1942-02-28	***	
Joyce	English	453453453	123456789	Michael	M	1988-01-04	***	
Joyce	English	453453453	123456789	Alice	F	1988-12-30	***	
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	***	

PERSONE_A_CARICO_EFFETTIVE	NOME_BATT	COGNOME	SSN	SSN_I	NOME_PERSONA_A_CARICO	SESSO	DATA_N	***
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	***	

RISULTATO	NOME_BATT	COGNOME	NOME_PERSONA_A_CARICO
Jennifer	Wallace	Abner	

Figura 7.12 Un'illustrazione dell'operazione di PRODOTTO CARTESIANO.

binare una tupla relativa a un impiegato di sesso femminile con le persone a suo carico – cioè con le tuple di **PERSONE_A_CARICO** i cui valori di **SSN_I** si accordano con il valore di **SSN** della tupla di **IMPIEGATO**. La relazione **PERSONE_A_CARICO_EFFETTIVE** realizza proprio questo.

Il PRODOTTO CARTESIANO dà luogo a tuple con tutti gli attributi propri delle due relazioni. È possibile poi effettuare una SELEZIONE delle sole tuple logicamente collegate delle due relazioni specificando un'appropriata condizione di selezione, come fatto nell'esempio precedente. Dato che questa sequenza costituita dal PRODOTTO CARTESIANO seguito da una SELEZIONE è usata piuttosto frequentemente per identificare e selezionare tuple collegate di due relazioni, è stata ideata un'operazione speciale, detta JOIN, per specificarla con una singola operazione.

7.4.5 L'operazione di JOIN

L'operazione di JOIN, indicata con \bowtie , è usata per unire *tuple logicamente collegate*, provenienti da due relazioni, in tuple singole. Essa è molto importante per ogni base di dati relazionale che abbia più di una relazione, perché consente di eseguire associazioni tra relazioni. Al fine di illustrare il join, si supponga di voler recuperare il nome del direttore di ogni dipartimento. A tal fine occorre unire ogni tupla di dipartimento con la tupla di impiegato il cui valore di **SSN** si accorda con il valore di **SSN_DIR** presente nella tupla del dipartimento. È possibile fare ciò usando l'operazione di JOIN, e quindi proiettando il risultato sugli attributi necessari nel modo seguente:

$$\text{DIR_DIP} \leftarrow \text{DIPARTIMENTO} \bowtie_{\text{SSN_DIR}=\text{SSN}} \text{IMPIEGATO}$$

$$\text{RISULTATO} \leftarrow \pi_{\text{NOME_DIP}, \text{COGNOME}, \text{NOME_BATT}}(\text{DIR_DIP})$$

La prima operazione è illustrata in Figura 7.13. Si noti che **SSN_DIR** è una chiave esterna e che il vincolo di integrità referenziale gioca un ruolo nell'avere tuple che si accordano nella relazione riferita **IMPIEGATO**. L'esempio sopra utilizzato per illustrare l'operazione di PRODOTTO CARTESIANO può essere specificato, usando l'operazione di JOIN, sostituendo le due operazioni:

$$\text{IMP_PERSONE_A_CARICO} \leftarrow \text{NOMI_IMP} \times \text{PERSONA_A_CARICO}$$

$$\text{PERSONE_A_CARICO_EFFETTIVE} \leftarrow \sigma_{\text{SSN}=\text{SSN}_I}(\text{IMP_PERSONE_A_CARICO})$$

DIR_DIP	NOME_D	NUMERO_D	SSN_DIR	***	NOME_BATT	INIZ_INT	COGNOME	SSN	***
Ricerca	5	333445555	***		Franklin	T	Wong	333445555	***
Amministrazione	4	987654321	***		Jennifer	S	Wallace	987654321	***
Sede centrale	1	888665555	***		James	E	Borg	888665555	***

Figura 7.13 Illustrazione dell'operazione di JOIN.

(a)	R	N_D	N_DI_IMPIEGATI	STIP_MEDIO
		5	4	33250
		4	3	31000
		1	1	55000

(b)	N_D	COUNT_SSN	AVERAGE_STIPENDIO
	5	4	33250
	4	3	31000
	1	1	55000

(c)	COUNT_SSN	AVERAGE_STIPENDIO
	8	35125

Figura 7.16 Un'illustrazione dell'operazione di FUNZIONE AGGREGATA. (a) $R(N_D, N_{DI_IMPIEGATO}, STIP_MEDIO) \leftarrow_{N_D} \bar{\cup}_{COUNT_SSN, AVERAGE_STIPENDIO}(IMPIEGATO)$. (b) $N_D \bar{\cup}_{COUNT_SSN, AVERAGE_STIPENDIO}(IMPIEGATO)$. (c) $\bar{\cup}_{COUNT_SSN, AVERAGE_STIPENDIO}(IMPIEGATO)$.

Il risultato di questa operazione è mostrato in Figura 7.16(a).

Nell'esempio sopra presentato è stata precisata una lista di nomi di attributi – tra parentesi nell'operazione di ridenominazione – per la relazione risultante R . Se non fosse eseguita nessuna ridenominazione, ogni attributo della relazione risultante corrispondente alla lista di funzioni sarebbe dato dalla concatenazione del nome di funzione con il nome di attributo nella forma $<\text{funzione}>_<\text{attributo}>$. Ad esempio, in Figura 7.16(b) è mostrato il risultato della seguente operazione:

$N_D \bar{\cup}_{COUNT_SSN, AVERAGE_STIPENDIO}(IMPIEGATO)$

Se non venissero specificati attributi di raggruppamento, le funzioni verrebbero applicate ai valori degli attributi di *tutte le tuple* della relazione, cosicché la relazione risultante avrebbe *una sola tupla*. Ad esempio, in Figura 7.16(c) è mostrato il risultato della seguente operazione:

$\bar{\cup}_{COUNT_SSN, AVERAGE_STIPENDIO}(IMPIEGATO)$

È importante notare che, in generale, i duplicati *non sono eliminati* quando viene applicata una funzione aggregata; così facendo si usa la solita interpretazione di funzioni come SUM e AVERAGE.¹¹ Occorre sottolineare che il risultato dell'applicazione di una funzione aggregata è una relazione, non uno scalare – anche se è costituito da un valore singolo.

¹¹ In SQL, l'opzione di eliminazione dei duplicati prima di applicare la funzione aggregata è resa disponibile tramite l'inserimento della parola chiave DISTINCT (si veda il Capitolo 8).

7.5.2 Operazioni di chiusura ricorsiva

Un altro tipo di operazione che, in generale, non può essere specificata nell'algebra relazionale di base è la **chiusura ricorsiva**. Questa operazione è applicata a un'associazione ricorsiva tra tuple dello stesso tipo, come ad esempio l'associazione tra un impiegato e un supervisore. Tale associazione è descritta dalla chiave esterna SUPERSSN della relazione IMPIEGATO nelle Figure 7.6 e 7.7, che collega ogni tupla impiegato (nel ruolo di chi è sorvegliato) a un'altra tupla impiegato (nel ruolo di supervisore). Un esempio di operazione ricorsiva consiste nel recuperare tutti gli impiegati sorvegliati da un impiegato e ("e" come employee) a tutti i livelli – cioè, tutti gli impiegati e' sorvegliati direttamente da e, tutti gli impiegati e" sorvegliati direttamente da ogni impiegato e', tutti gli impiegati e'' sorvegliati direttamente da ogni impiegato e'', e così via. Anche se in algebra relazionale è semplice specificare tutti gli impiegati sorvegliati da *a uno specifico livello*, è difficile specificare tutti quelli sorvegliati a *tutti i livelli*. Ad esempio, per specificare gli SSN di tutti gli impiegati e' sorvegliati direttamente – *al livello uno* – dall'impiegato e il cui nome è 'James Borg' (Figura 7.6), possono eseguire le seguenti operazioni:

```
BORG_SSN ← πSSN(σNOME_BATT='James' AND COGNOME='Borg(IMPIEGATO))
SUPERVISIONE(SSN1, SSN2) ← πSSN, SUPERSSN(IMPIEGATO)
RISULTATO1(SSN) ← πSSN1(SUPERVISIONE ▷SSN2=SSN BORG_SSN)
```

(Borg ha SSN uguale a 888665555)		
SUPERVISIONE	SSN1	SSN2
123456789	333445555	
333445555	888665555	
999887777	987654321	
987654321	888665555	
666884444	333445555	
453453453	333445555	
987987987	987654321	
888665555	null	

RISULTATO 1	SSN	RISULTATO 2	SSN	RISULTATO	SSN
	333445555		123456789		123456789
	987654321		999887777		999887777
			666884444		666884444
			453453453		453453453
			987987987		987987987
			333445555		333445555
			987654321		987654321

Figura 7.17 Un'interrogazione ricorsiva a due livelli.

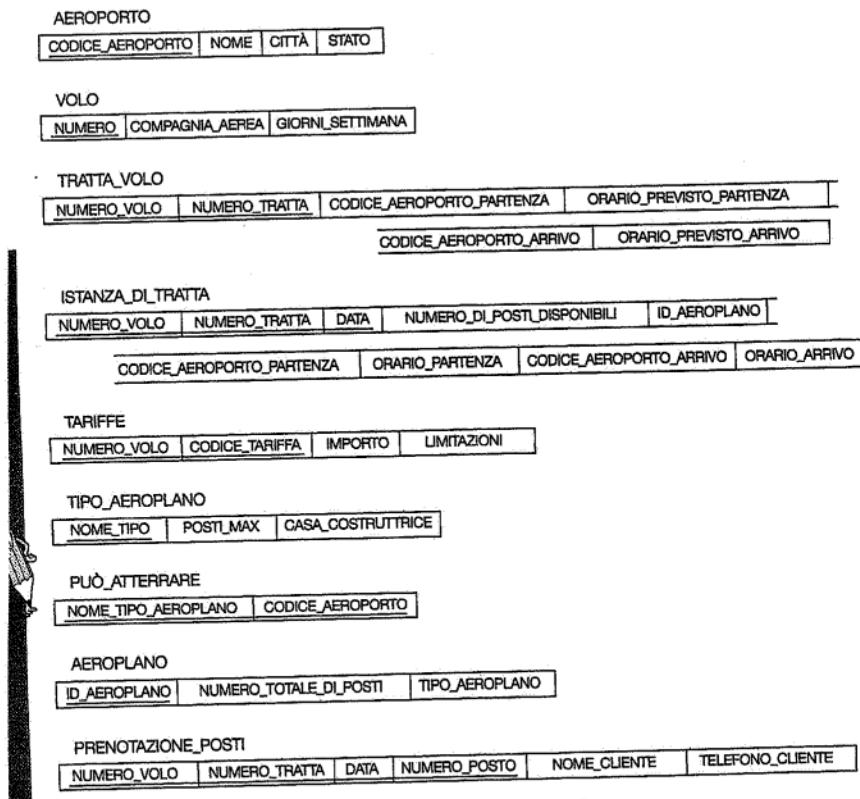


Figura 7.19 Lo schema di base di dati relazionale COMPAGNIA_AEREA.

- i. Si modifichino in '123456789' e in '1999-10-01', rispettivamente, i valori di SSN_DIR e di DATA_INIZIO_DIR della tupla DIPARTIMENTO con NUMERO_D = 5.
- j. Si modifichi in '943775543' il valore dell'attributo SUPERSSN della tupla IMPIEGATO con SSN = '999887777'.
- k. Si modifichi in '5,0' il valore dell'attributo ORE della tupla LAVORA_SU con SSN_I = '999887777' e N_P = 10.
- 7.20. Si consideri lo schema di base di dati relazionale COMPAGNIA_AEREA mostrato in Figura 7.19, che descrive una base di dati per le informazioni sui voli di una compagnia aerea. Ogni VOLO è identificato da un NUMERO del volo ed è composto da una o più tratte (TRATTA_VOLO) con NUMERO_TRATTA 1, 2, 3 ecc. Ogni tratta ha i suoi orari di partenza e di arrivo previsti nonché gli aeroporti interessati e presenta molte istanze (ISTAN-

ZA_DI_TRATTA) – una per ogni DATA nella quale c'è effettivamente il volo. Per ogni volo ci sono diverse TARIFFE. Per ogni istanza di tratta si memorizzano le prenotazioni dei posti (PRENOTAZIONE_POSTI), come anche l'AEROPLANO usato su quella tratta e gli orari effettivi di partenza e arrivo, nonché gli aeroporti effettivamente utilizzati. Un AEROPLANO è identificato da un ID_AEROPLANO e ha uno specifico TIPO_AEROPLANO. PUO_ATTERRARE collega un TIPO_AEROPLANO con l'AEROPORTO su cui un aeroplano di quel tipo può atterrare. Un AEROPORTO è identificato da un CODICE_AEROPORTO. Si specifichino le seguenti interrogazioni in algebra relazionale.

- a. Per ogni volo, si elenchi il numero del volo, l'aeroporto di partenza per la prima tratta del volo e l'aeroporto d'arrivo per l'ultima tratta del volo.
- b. Si prepari un elenco dei numeri di volo e dei giorni della settimana per tutti i voli o le tratte di volo che partono dall'Aeroporto Intercontinentale di Houston (codice aeroporto 'IAM') e arrivano all'Aeroporto Internazionale di Los Angeles (codice aeroporto 'LAX').
- c. Si elenchi il numero di volo, il codice dell'aeroporto di partenza, l'orario di partenza previsto, il codice dell'aeroporto di arrivo, l'orario di arrivo previsto nonché i giorni della settimana di tutti i voli o le tratte di volo che partono da uno degli aeroporti della città di Houston e arrivano a uno degli aeroporti della città di Los Angeles.
- d. Si elenchino tutte le informazioni sulle tariffe per il volo numero 'CO 197'.
- e. Si recuperi il numero di posti disponibili per il volo numero 'CO 197' relativo alla data '1999-10-09'.
- 7.21. Si consideri un aggiornamento della base di dati COMPAGNIA_AEREA, che inserisca una prenotazione su uno specifico volo o tratta per una data assegnata.
 - a. Si forniscano le operazioni per questo aggiornamento.
 - b. Quali tipi di vincoli è il caso di controllare?
 - c. Quali di questi vincoli sono vincoli di chiave, di integrità dell'entità e di integrità referenziale, e quali non lo sono?
 - d. Si specifichino tutti i vincoli di integrità referenziale di Figura 7.19.
- 7.22. Si consideri la relazione

INSEGNAMENTO (#Insegnamento, #Mod_Univ, NomeAssistente, Semestre, CodiceEdificio, #Aula, PeriodoTemporale, GiorniDellaSettimana, OreDiCredito).

Essa rappresenta insegnamenti tenuti in un'università, con #Mod_Univ univoco. Il lettore identifichi quelle che, a suo parere, dovrebbero essere le varie chiavi candidate, ed esprima i vincoli da soddisfare perché ogni chiave candidata sia valida.

- 7.23. Si consideri lo schema relazionale BIBLIOTECA di Figura 7.20, usato per tener traccia dei libri, di chi li prende a prestito e dei prestiti. In figura i vincoli di integrità referenziale sono rappresentati con archi orientati, usando una notazione analoga a quella di Figura 7.7. Si scrivano espressioni relazionali per le seguenti interrogazioni sulla base di dati BIBLIOTECA.
 - a. Quante copie del libro intitolato *La tribù perduta* sono in possesso della sede della biblioteca di nome 'Sharpstown'?
 - b. Quante copie del libro intitolato *La tribù perduta* sono in possesso di ogni singola sede della biblioteca?

Tabella T1			Tabella T2		
P	Q	R	A	B	C
10	a	5	10	b	6
15	b	8	25	c	3
25	a	6	10	b	5

Figura 7.21 Uno stato di base di dati per le relazioni T1 e T2.

- c. Si elenchi ogni dipartimento che ha *tutti* i libri adottati pubblicati da 'BC Publishing'.
- 7.27. Si considerino le due tabelle T1 e T2 presenti in Figura 7.21. Si mostrino i risultati delle seguenti operazioni:

- a. $T1 \bowtie_{T1.P=T2.A} T2$
- b. $T1 \bowtie_{T1.Q=T2.B} T2$
- c. $T1 \bowtie_{T1.P=T2.A} T2$
- d. $T1 \bowtie_{T1.Q=T2.B} T2$
- e. $T1 \cup T2$
- f. $T1 \bowtie_{(T1.P=T2.A \text{ AND } T1.R=T2.C)} T2$

- 7.28. Si considerino le relazioni seguenti per una base di dati che tiene traccia delle vendite di automobili di un rivenditore d'auto (Opzione fa riferimento a qualche equipaggiamento opzionale installato su un'automobile):

AUTOMOBILE (Num-Serie, Modello, Produttore, Prezzo)

OPZIONI (Num-Serie, Nome-Opzione, Prezzo)

VENDITE (Id-Venditore, Num-Serie, Data, Prezzo-Vendita)

VENDITORE (Id-Venditore, Nome, Telefono)

Prima di tutto si specifichino le chiavi esterne per lo schema sopra presentato, enunciando esplicitamente ogni assunzione fatta. Si popolino quindi le relazioni con qualche esempio di tupla, e poi si presenti un esempio di inserimento nelle relazioni VENDITE e VENDITORE che *viola* i vincoli di integrità referenziale, e un altro esempio che invece non dia luogo a questa violazione. Infine si specifichino le seguenti interrogazioni in algebra relazionale.

- a. Per il venditore di nome 'Jane Doe' si presenti un elenco con le seguenti informazioni per tutte le macchine che ha venduto: #Serie, Produttore, Prezzo-vendita.
- b. Si elenchi il #Serie e il Modello delle automobili che non hanno opzioni.
- c. Si consideri l'operazione di join naturale tra VENDITORE e VENDITE. Qual è il significato di un join esterno sinistro fra queste tabelle (non si cambi l'ordine delle relazioni)? Si illustri la risposta con un esempio.
- d. Si scriva un'interrogazione in algebra relazionale che utilizzi la selezione e un'operazione insiemistica e si illustri a parole cosa fa l'interrogazione.

Bibliografia selezionata

Il modello relazionale è stato introdotto da Codd (1970) in un articolo ormai classico. Codd ha introdotto anche l'algebra relazionale e ha posto i fondamenti teorici del modello relazionale in una serie di articoli (Codd 1971, 1972a, 1972b, 1974); in seguito gli è stato assegnato il premio Turing, la massima onorificenza dell'ACM, per il suo lavoro sul modello relazionale. In un lavoro successivo Codd (1979) ha studiato un'estensione del modello relazionale per inserire più metadati e semantica sulle relazioni; ha anche proposto una logica a tre valori per gestire l'incertezza nelle relazioni e inserire così i valori nulli nell'algebra relazionale. Il modello risultante è noto come RM/T. In precedenza Childs (1968) aveva usato la teoria degli insiemi per modellare le basi di dati. Più recentemente, Codd (1990) ha pubblicato un libro in cui si esaminano oltre 300 caratteristiche del modello di dati relazionale e dei sistemi di basi di dati relazionali.

A partire dal lavoro pionieristico di Codd sono state condotte molte ricerche su vari aspetti del modello relazionale. Todd (1976) descrive un DBMS sperimentale, detto PRTV, che implementa direttamente le operazioni dell'algebra relazionale. Date (1983) tratta i join esterni. Il lavoro di Schmidt e Swenson (1975) introduce semantica aggiuntiva nel modello relazionale classificando diversi tipi di relazioni. Il modello Entità-Associazione di Chen (1976), studiato nel Capitolo 3, è un mezzo per comunicare la semantica del mondo reale di una base di dati relazionale a livello concettuale. L'articolo di Wiederhold e Elmasri (1979) introduce vari tipi di collegamenti tra relazioni per aumentare i vincoli relativi. Il lavoro per ampliare le operazioni relazionali è stato trattato da Carlis (1986) e Ozsoyoglu e altri (1985). Il lavoro di Cammarata e altri (1989) estende i vincoli di integrità e i join del modello relazionale. Altre note bibliografiche relative ad altri aspetti del modello relazionale e ai suoi linguaggi, sistemi, estensioni, nonché alla teoria ad esso relativa sono fornite nei Capitoli 8, 9, 10, 12, 14 e 15.

SQL, invece, fornisce un'interfaccia *dichiarativa* di alto livello, permettendo all'utente di specificare solo *quale* deve essere il risultato e lasciando al DBMS l'ottimizzazione e le decisioni effettive su come eseguire l'interrogazione. SQL include alcune funzioni dell'algebra relazionale, ma si basa in misura maggiore sul *calcolo relazionale delle tuple*, che è un altro linguaggio formale di interrogazione per le basi di dati relazionali. La sintassi di SQL è più facile da usare rispetto a quella di linguaggi formali.

SQL è l'acronimo di Structured Query Language, cioè Linguaggio Strutturato di Interrogazione. In origine SQL si chiamava SEQUEL (Structured English QUERy Language, Linguaggio Inglese Strutturato di Interrogazione) ed è stato progettato e implementato presso il Centro di Ricerca dell'IBM come interfaccia verso un sistema di basi di dati relazionali sperimentale chiamato SYSTEM R. SQL è oggi il linguaggio standard per i DBMS relazionali commerciali. Gli sforzi comuni dell'ANSI (American National Standards Institute, Istituto Nazionale Americano di Standardizzazione) e dell'ISO (International Standards Organization, Organizzazione Internazionale di Standardizzazione) hanno portato a una versione standard di SQL (ANSI 1986), denominata SQL-86 o SQL1. Successivamente è stato sviluppato uno standard revisionato e più esteso chiamato SQL2 (detto anche SQL-92). Il progetto della terza versione del linguaggio, SQL3, è già in fase avanzata, e verrà ulteriormente migliorato con l'aggiunta di concetti orientati agli oggetti e di altri concetti recenti relativi ai database.

SQL è un linguaggio completo; comprende istruzioni per la definizione dei dati, l'interrogazione e l'aggiornamento; quindi è sia un DDL sia un DML. Inoltre, fornisce tutte le funzionalità per definire le viste sulla base di dati, per specificare la sicurezza e le autorizzazioni per definire le viste sulla base di dati, per specificare la sicurezza e le autorizzazioni di accesso, per definire i vincoli di integrità e per specificare i controlli sulle transazioni. Comprende persino le regole per incorporare le istruzioni SQL nei linguaggi di programmazione di uso generale come C o PASCAL.¹ La maggior parte di questi argomenti verrà trattata nei paragrafi successivi nei quali si prenderà in considerazione principalmente SQL2.

Nel Paragrafo 8.1 esamineremo i comandi DDL di SQL2 per creare e modificare gli schemi, le tabelle e i vincoli. Nel Paragrafo 8.2 analizzeremo i costrutti SQL fondamentali per specificare le interrogazioni, mentre nel Paragrafo 8.3 daremo una panoramica delle funzioni più complesse. Nel Paragrafo 8.4 descriveremo i comandi SQL per inserire, cancellare e aggiornare le informazioni e nel Paragrafo 8.5 le viste (tabelle virtuali). Nel Paragrafo 8.6 mostreremo come dei vincoli generali possano essere specificati come asserzioni o trigger, mentre nel Paragrafo 8.7 elencheremo alcune funzioni SQL presentate in altri capitoli del libro; queste includono istruzioni SQL encapsulate nei programmi scritti in altri linguaggi (si veda il Capitolo 12).

Il lettore che desidera un'introduzione meno approfondita a SQL, può evitare completamente o in parte la lettura dei Paragrafi 8.2.5, 8.3, 8.5, 8.6 e 8.7.

8.1 La definizione dei dati e le modifiche agli schemi in SQL2

SQL utilizza i termini tabella, riga e colonna rispettivamente come sinonimi di relazione, tupla e attributo; nel seguito si utilizzeranno i termini e questi sinonimi in modo intercambiabile. I comandi di SQL2 per la definizione dei dati sono CREATE, ALTER e DROP e sono esaminati qui nei Sottoparagrafi 8.1.2-8.1.4. Prima di tutto, però, nel Paragrafo 8.1.1 vengono trattati i concetti di schema e catalogo. Nel Paragrafo 8.1.2 invece viene spiegato come si creano le tabelle, quali sono i tipi di dati disponibili per gli attributi e come sono specificati i vincoli. Nei Paragrafi 8.1.3 e 8.1.4 sono analizzati i comandi per la modifica degli schemi disponibili in SQL2, che possono essere usati per aggiungere ed eliminare tabelle, attributi e vincoli. Viene data solo una panoramica delle funzioni più importanti; approfondimenti possono essere trovati nel documento che descrive lo standard SQL2.

8.1.1 I concetti di schema e di catalogo

Le versioni precedenti di SQL non includevano il concetto di schema relazionale; tutte le tabelle (relazioni) erano considerate parte del medesimo schema. Il concetto di schema SQL è stato incorporato in SQL2 con lo scopo di raggruppare le tabelle e altre costrutti, che appartengono alla medesima applicazione di basi di dati. Uno schema SQL è identificato da un nome di schema e contiene un identificatore di autorizzazione per indicare l'utente o l'account che ne è proprietario, nonché descrittori per ciascun elemento nello schema. I suoi elementi comprendono le tabelle, i vincoli, le viste, i domini e altri costrutti (quali ad esempio le autorizzazioni) che lo descrivono. Uno schema è creato attraverso l'istruzione CREATE SCHEMA, che può includere le definizioni di tutti gli elementi dello schema. In alternativa, è possibile limitarsi ad assegnare un nome e un identificatore di autorizzazione allo schema e definire gli elementi successivamente. Ad esempio, l'istruzione qui di seguito riportata crea uno schema chiamato AZIENDA, il cui proprietario è l'utente con l'identificatore di autorizzazione JSMITH:

```
CREATE SCHEMA AZIENDA AUTHORIZATION JSMITH;
```

Oltre al concetto di schema, in SQL2 esiste quello di catalogo, che consiste in una raccolta di schemi cui viene associato un nome in un ambiente SQL. Un catalogo contiene sempre uno schema speciale chiamato INFORMATION_SCHEMA, che fornisce agli utenti autorizzati le informazioni su tutti i descrittori degli elementi di tutti gli schemi del catalogo. I vincoli d'integrità, quali quelli di integrità referenziale, possono essere definiti tra relazioni solo se i loro schemi si trovano all'interno del medesimo catalogo. Gli schemi presenti nello stesso catalogo possono anche condividere certi elementi come le definizioni di dominio.

¹ In origine SQL comprendeva delle istruzioni per creare ed eliminare gli indici sui file che contengono le relazioni, poi rimosse dallo standard SQL2.

8.1.2 Il comando CREATE TABLE e i vincoli di tipi di dati

Il comando CREATE TABLE è usato per specificare una nuova relazione assegnandole un nome e specificando i suoi attributi e vincoli. Gli attributi sono specificati per primi e a ciascun attributo è assegnato un nome, un tipo di dati per specificare il suo dominio di valori e gli eventuali vincoli sui valori come, ad esempio, NOT NULL. La chiave, vincolo di integrità dell'entità, e i vincoli d'integrità referenziale possono essere specificati direttamente all'interno dell'istruzione CREATE TABLE, dopo che sono stati dichiarati gli attributi; in alternativa si può aggiungere successivamente usando il comando ALTER TABLE (si veda il Sottoparagrafo 8.1.4). In Figura 8.1(a) sono riportate le istruzioni di definizione dei dati SQL per lo schema di basi di dati relazionali presentato in Figura 7.7. Di solito, lo schema SQL in cui vengono dichiarate le relazioni è specificato implicitamente dall'ambiente in cui sono eseguite le istruzioni CREATE TABLE. In alternativa, si può esplicitamente associare il nome dello schema ai nomi delle relazioni, separandoli con un punto. Ad esempio, scrivendo:

```
CREATE TABLE AZIENDA.IMPiegato ...
```

invece di

```
CREATE TABLE IMPiegato ...
```

come in Figura 8.1(a), si può indicare esplicitamente (invece che implicitamente) che la tabella IMPiegato fa parte dello schema AZIENDA.

Domini e tipi di dati in SQL2. I tipi di dati disponibili per gli attributi comprendono tipi numerici, stringhe di caratteri, stringhe di bit, data e ora. I tipi di dati numerici comprendono numeri interi di diverse dimensioni (INTEGER o INT e SMALLINT) e numeri reali di diversa precisione (FLOAT, REAL, DOUBLE PRECISION). È possibile definire il formato dei numeri in modo preciso usando DECIMAL(*i,j*) o DEC(*i,j*) oppure NUMERIC(*i,j*), in cui *i*, la *precisione*, è il numero totale di cifre decimali e *j*, la *scala*, è il numero di cifre dopo la virgola decimale. Il valore predefinito per la scala è zero e quello per la precisione è definito dall'implementazione.

I tipi di dati stringhe di caratteri sono di lunghezza fissa, CHAR(*n*) o CHARACTER(*n*), in cui *n* è il numero di caratteri, oppure di lunghezza variabile, VARCHAR(*n*) o CHAR VARYING(*n*) o CHARACTER VARYING(*n*), in cui *n* è il numero massimo di caratteri. I tipi di dati stringhe di bit sono di lunghezza fissa *n*, BIT(*n*), oppure di lunghezza variabile, BIT VARYING(*n*), in cui *n* è il numero massimo di bit. Il valore predefinito per *n*, la lunghezza di una stringa di caratteri o di una stringa di bit, è uno.

Vi sono nuovi tipi di dati per data e ora in SQL2. Il tipo di dati DATE ha dieci posizioni e i suoi componenti sono YEAR, MONTH e DAY (cioè anno, mese e giorno) tipicamente nella forma YYYY-MM-DD. Il tipo di dati TIME ha almeno otto posizioni con i componenti HOUR, MINUTE e SECOND (cioè ora, minuto e secondo), tipicamente nella forma HH:MM:SS. L'implementazione SQL2 consente solo l'uso di data e ora valide.

```
CREATE TABLE IMPiegato
( NOME_BATT          VARCHAR(15)      NOT NULL,
  INIZ_INT            CHAR,
  COGNOME             VARCHAR(15)      NOT NULL,
  SSN                 CHAR(9)         NOT NULL,
  DATA_N              DATE,
  INDIRIZZO           VARCHAR(30),
  SESSO               CHAR,
  STIPENDIO           DECIMAL(10,2),
  SUPERSSN            CHAR(9),
  N_D                 INT             NOT NULL,
  PRIMARY KEY (SSN),
  FOREIGN KEY (SUPERSSN) REFERENCES IMPiegato(SSN),
  FOREIGN KEY (N_D) REFERENCES DIPARTIMENTO(NUMERO_D));
CREATE TABLE DIPARTIMENTO
( NOME_D              VARCHAR(15)      NOT NULL,
  NUMERO_D             INT             NOT NULL,
  SSN_DIR              CHAR(9)         NOT NULL,
  DATA_INIZIO_DIR      DATE,
  PRIMARY KEY (NUMERO_D),
  UNIQUE (NOME_D),
  FOREIGN KEY (SSN_DIR) REFERENCES IMPiegato(SSN));
CREATE TABLE SEDI_DIP
( NUMERO_D             INT             NOT NULL,
  SEDE_D               VARCHAR(15)      NOT NULL,
  PRIMARY KEY (NUMERO_D, SEDE_D),
  FOREIGN KEY (NUMERO_D) REFERENCES DIPARTIMENTO(NUMERO_D));
CREATE TABLE PROGETTO
( NOME_P               VARCHAR(15)      NOT NULL,
  NUMERO_P              INT             NOT NULL,
  SEDE_P               VARCHAR(15),
  NUM_D                INT             NOT NULL,
  PRIMARY KEY (NUMERO_P),
  UNIQUE (NOME_P),
  FOREIGN KEY (NUM_D) REFERENCES DIPARTIMENTO(NUMERO_D));
CREATE TABLE Lavora_su
( SSN_I                CHAR(9)         NOT NULL,
  N_P                  INT             NOT NULL,
  ORE                 DECIMAL(3,1),
  PRIMARY KEY (SSN_I, N_P),
  FOREIGN KEY (SSN_I) REFERENCES IMPiegato(SSN),
  FOREIGN KEY (N_P) REFERENCES PROGETTO(NUMERO_P));
CREATE TABLE PERSONA_A_CARICO
( SSN_I                CHAR(9)         NOT NULL,
  NOME_PERSONA_A_CARICO VARCHAR(15)      NOT NULL,
  SESSO                CHAR,
  DATA_N              DATE,
  PARENTELA           VARCHAR,
  PRIMARY KEY (SSN_I, NOME_PERSONA_A_CARICO),
  FOREIGN KEY (SSN_I) REFERENCES IMPiegato(SSN));
```

Figura 8.1(a) Definizione dei dati di SQL2: un'esemplificazione. Le istruzioni SQL2 definiscono lo schema AZIENDA di Figura 7.7.

Inoltre, un tipo di dati TIME(*i*), in cui *i* è definito *precisione frazionaria dei secondi dell'orario*, specifica *i* + 1 posizioni aggiuntive per il tipo TIME; una posizione serve per un carattere separatore addizionale e le restanti *i* posizioni per specificare frazioni decimali di secondo. Un tipo di dati TIME WITH TIME ZONE include altre sei posizioni per specificare uno *spostamento* dal fuso orario universale standard, che prende valori nell'intervallo da + 13:00 a - 12:59 nelle unità HOURS:MINUTES. Se la clausola WITH TIME ZONE non è specificata, il valore predefinito è il fuso orario locale della sessione SQL. Infine, il tipo di dati timestamp (TIMESTAMP) comprende i campi DATE e TIME più un minimo di sei posizioni per le frazioni di secondo e un qualificatore facoltativo WITH TIME ZONE.

Un altro tipo di dati analogo a DATE, TIME e TIMESTAMP è il tipo di dati INTERVAL, che specifica un intervallo temporale, cioè un *valore relativo* che può essere usato per aumentare o diminuire il valore assoluto di una data, ora o timestamp. Gli intervalli sono qualificati come intervalli YEAR/MONTH o DAY/TIME.

In SQL2 è possibile specificare direttamente il tipo di dati di ogni attributo, come in Figura 8.1(a); alternativamente, può essere dichiarato il dominio e il nome del dominio usato.

La seconda tecnica rende più facile cambiare il tipo di dati di un dominio utilizzato da numerosi attributi in uno schema e migliora la leggibilità dello schema. Ad esempio, si può creare un dominio SSN_TYPE tramite l'istruzione seguente:

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);
```

Si può usare SSN_TYPE al posto di CHAR(9) in Figura 8.1(a) per gli attributi SSN e SUPERSSN di IMPIEGATO, SSN_DIR di DIPARTIMENTO, SSN_I di LAVORA_SU e SSN_I di PERSONA_A_CARICO. Un dominio può avere anche una specificazione predefinita (di default) attraverso una clausola DEFAULT, come si vedrà successivamente per gli attributi.

Specificazione dei vincoli e dei valori predefiniti in SQL2. Poiché SQL accetta NULL come valori degli attributi, può essere specificato un vincolo NOT NULL se il valore NULL non è consentito per un particolare attributo. Questo vincolo dovrebbe essere sempre specificato per gli attributi della chiave primaria di ogni relazione e anche per qualsiasi altro attributo i cui valori non devono essere NULL, come mostrato in Figura 8.1(a). È possibile anche definire un valore predefinito per un attributo aggiungendo la clausola DEFAULT <value> alla sua definizione. Il valore predefinito è inserito in qualsiasi nuova tupla se non viene fornito un valore esplicito per quell'attributo. In Figura 8.1(b) è presentato un esempio di specificazione di un manager predefinito per ogni nuovo reparto e un reparto predefinito per ogni nuovo dipendente. Se non è specificata alcuna clausola predefinita, il valore predefinito di default è NULL.

Dopo aver specificato i vincoli sugli attributi (o colonne), è possibile specificare vincoli di tabella, tra cui vi sono quelli di chiave e di integrità referenziale, come illustrato in Figura 8.1(a).² La clausola PRIMARY KEY specifica uno o più attributi che costituiscono la chiave

CREATETABLE IMPIEGATO

```
(...,  
    N_D          INT NOT NULL DEFAULT 1,  
CONSTRAINT IMPIEGATO_PK  
    PRIMARY KEY (SSN),  
CONSTRAINT IMPIEGATO_SUPERFK  
    FOREIGN KEY (SUPERSSN) REFERENCES IMPIEGATO(SSN)  
        ON DELETE SET NULL ON UPDATE CASCADE,  
CONSTRAINT IMPIEGATO_DIP_FK  
    FOREIGN KEY (N_D) REFERENCES DIPARTIMENTO(NUMERO_D)  
        ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

CREATE TABLE DIPARTIMENTO

```
(...,  
    SSN_DIR     CHAR(9) NOT NULL DEFAULT '888665555',  
...,  
CONSTRAINT DIP_PK  
    PRIMARY KEY (NUMERO_D),  
CONSTRAINT DIP_SK  
    UNIQUE (NOME_D),  
CONSTRAINT DIP_DIR_FK  
    FOREIGN KEY (SSN_DIR) REFERENCES IMPIEGATO(SSN)  
        ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

CREATE TABLE SEDI_DIP

```
(...,  
    PRIMARY KEY (NUMERO_D, SEDE_D),  
    FOREIGN KEY (NUMERO_D) REFERENCES DIPARTIMENTO(NUMERO_D)  
        ON DELETE CASCADE ON UPDATE CASCADE);
```

Figura 8.1(b) Definizione dei dati di SQL2: un'esemplificazione. La specifica dei valori predefiniti per gli attributi e le azioni referenziali che vengono innescate.

primaria di una relazione. La clausola UNIQUE specifica le chiavi ulteriori (o secondarie). L'integrità referenziale è specificata attraverso la clausola FOREIGN KEY.

Come si è visto nel Sottoparagrafo 7.2.4, un vincolo d'integrità referenziale può essere violato quando le tuple sono inserite o cancellate o quando viene modificato il valore di un attributo di una chiave esterna. In SQL2, il progettista dello schema può specificare l'azione che deve essere eseguita se un vincolo d'integrità referenziale viene violato, perché si cancella una tupla o si modifica un valore della chiave primaria alla quale si fa riferimento, specificando una clausola di azione referenziale innescata (referential triggered action) associata a un qualsiasi vincolo di chiave esterna. Le opzioni includono SET NULL, CASCADE e SET DEFAULT e devono essere qualificate con ON DELETE (quando si effettua una cancellazione) oppure ON UPDATE (quando si effettua una modifica). L'uso delle opzioni è illustrato dall'esempio di Figura 8.1(b). Qui il progettista della base di dati ha selezionato le opzioni SET NULL ON DELETE e CASCADE ON UPDATE per la chiave esterna SUPERSSN di IMPIEGATO. Ciò significa che se la tupla di un dipendente che è anche supervisore viene cancellata, il valore di SUPERSSN viene automaticamente impostato a NULL per tutte le tuple dei dipendenti che facevano riferimento alla tupla cancellata di dipendente. D'altra parte se il valore di SSN di un dipendente che è anche supervisore viene aggiornato (ad esempio perché era stato

² I vincoli di chiave e d'integrità referenziale non erano inclusi nelle versioni precedenti di SQL. In alcune implementazioni precedenti, le chiavi erano specificate implicitamente a livello interno attraverso il comando CREATE INDEX.

inserito in modo non corretto), il nuovo valore viene *propagato* agli attributi SUPERSSN di tutte le tuple dei dipendenti dei quali il dipendente aggiornato è il supervisore.

In generale l'operazione eseguita dal DBMS come conseguenza delle opzioni SET NULL o SET DEFAULT è la stessa per ON DELETE o ON UPDATE; il valore degli attributi coinvolti nel vincolo viene modificato in NULL per SET NULL e nel valore predefinito specificato per SET DEFAULT. L'azione associata a CASCADE ON DELETE è quella di cancellare tutte le tuple a cui il vincolo si riferisce, mentre quella associata a CASCADE ON UPDATE consiste nel modificare il valore della chiave esterna facendolo diventare uguale a quello della chiave primaria aggiornata (nuova) in tutte le tuple a cui il vincolo si riferisce. È responsabilità del progettista della base di dati scegliere l'azione appropriata e specificarla in DDL. Come regola generale, l'opzione CASCADE è adatta a relazioni di "rapporto" come LAVORA_SU, a relazioni che rappresentano attributi con più valori come SEDI_DIP e a relazioni che rappresentano entità deboli come DIPENDENTE.

Nella Figura 8.1(b) si vede anche che a un vincolo può essere assegnato un nome, usando la parola chiave CONSTRAINT. I nomi di tutti i vincoli all'interno di un particolare schema devono essere unici.

Il nome del vincolo è usato per identificare un particolare vincolo nel caso in cui questo debba poi essere eliminato e sostituito con un altro, come verrà spiegato nel Sottoparagrafo 8.1.4. L'assegnazione dei nomi ai vincoli è però facoltativa.

Le relazioni che vengono dichiarate con le istruzioni CREATE TABLE sono chiamate **tabelle base** (o relazioni base); ciò significa che la relazione e le sue tuple sono effettivamente create e memorizzate dal DBMS sotto forma di un file. Queste relazioni base si distinguono dalle relazioni virtuali create con l'istruzione CREATE VIEW (si veda il Paragrafo 8.5), che possono o meno corrispondere a un file fisico. In SQL gli attributi in una tabella base sono considerati ordinati nella sequenza in cui sono specificati nell'istruzione CREATE TABLE, ma le righe (tuple), invece, non si considerano ordinate all'interno di una relazione.

8.1.3 I comandi DROP SCHEMA e DROP TABLE

Se un intero schema non è più necessario, può essere utilizzato il comando DROP SCHEMA. Esistono due opzioni relative al *comportamento di eliminazione*: CASCADE e RESTRICT. Ad esempio, per eliminare lo schema della base di dati AZIENDA e tutte le sue tabelle, i domini e gli altri elementi, l'opzione CASCADE va usata nel modo seguente:

```
DROP SCHEMA AZIENDA CASCADE;
```

Se viene scelta l'opzione RESTRICT al posto di CASCADE, lo schema è eliminato solo se non contiene *elementi*; altrimenti il comando DROP non viene eseguito.

Se una relazione base all'interno di uno schema non è più necessaria, la relazione e la sua definizione possono essere eliminate usando il comando DROP TABLE. Ad esempio, se non si desidera più tenere traccia delle persone a carico dei dipendenti nella base di dati AZIENDA di Figura 7.6, si può eliminare la relazione PERSONA_A_CARICO immettendo il comando:

```
DROP TABLE PERSONA_A_CARICO CASCADE;
```

Se si usa l'opzione RESTRICT al posto di CASCADE, la tabella viene eliminata solo se non viene fatto alcun riferimento a essa in un qualsiasi vincolo (ad esempio nelle definizioni delle chiavi esterne di un'altra relazione) o vista (si veda il Paragrafo 8.5). Con l'opzione CASCADE tutti i vincoli e le viste che si riferiscono alla tabella vengono eliminati automaticamente dallo schema insieme alla tabella stessa.

8.1.4 Il comando ALTER TABLE

La definizione di una tabella base può essere cambiata usando il comando ALTER TABLE, che è un comando di *evoluzione dello schema*. Le possibili *operazioni di modifica delle tabelle* comprendono l'aggiunta o l'eliminazione di una colonna (attributo), la modifica di una definizione di colonna e l'aggiunta o l'eliminazione di vincoli di tabella. Ad esempio, per inserire un attributo che tiene traccia dei compiti dei dipendenti nella relazione base IMPIEGATO dello schema AZIENDA si può usare il comando:

```
ALTER TABLE AZIENDA.IMPiegato ADD Lavoro VARCHAR(12);
```

Ovviamente, è necessario immettere un valore per il nuovo attributo LAVORO per ciascuna tupla di IMPIEGATO. Ciò può essere eseguito specificando una clausola che specifica un valore predefinito o utilizzando il comando UPDATE (si veda il Paragrafo 8.4). Se non è specificata una clausola per un valore predefinito, il nuovo attributo avrà il valore NULL in tutte le tuple della relazione subito dopo che il comando è eseguito; quindi in questo caso il vincolo NOT NULL non è consentito.

Anche l'eliminazione di una colonna può avvenire con le opzioni CASCADE o RESTRICT. Se si è scelta l'opzione CASCADE, tutti i vincoli e le viste che si riferiscono alla colonna vengono automaticamente eliminati dallo schema, insieme ad essa. Se si specifica l'opzione RESTRICT, il comando ha successo solo se non vi sono viste o vincoli che fanno riferimento alla colonna. Ad esempio, il seguente comando elimina l'attributo INDIRIZZO dalla tabella base IMPIEGATO:

```
ALTER TABLE AZIENDA.IMPiegato DROP INDIRIZZO CASCADE;
```

È possibile anche modificare una definizione di colonna eliminando la clausola che specifica il valore predefinito esistente oppure definendo una nuova clausola che specifica un valore predefinito. Gli esempi seguenti illustrano questa tecnica:

```
ALTER TABLE AZIENDA.DIPARTIMENTO ALTER SSN_DIR DROP  
DEFAULT;  
ALTER TABLE AZIENDA.DIPARTIMENTO ALTER SSN_DIR SET DEFAULT  
"33344555";
```

Infine si possono cambiare i vincoli specificati su una tabella aggiungendone di nuovi o eliminandone alcuni. Perché possa essere rimosso un vincolo, ad esso deve essere stato associato un nome quando è stato specificato. Ad esempio, per eliminare dalla relazione IMPIEGATO il vincolo denominato IMPTEGATO_SUPERFK in Figura 8.1(b) si scrive

```
ALTER TABLE AZIENDA.IMPiegato
DROP CONSTRAINT IMPiegato_SUPERFK CASCADE;
```

Una volta che l'eliminazione è stata eseguita, è possibile definire un vincolo sostitutivo aggiungendolo alla relazione. L'aggiunta è specificata usando la parola chiave **ADD** seguita dal nuovo vincolo, che può essere denominato o non denominato, ed essere di uno qualsiasi dei tipi trattati nel Sottoparagrafo 8.1.2.

Nei paragrafi precedenti è stata data una panoramica sui comandi di SQL2 per la definizione dei dati e l'evoluzione degli schemi. Vi sono molti altri dettagli e opzioni, per i quali il lettore può fare riferimento ai documenti che descrivono gli standard SQL e SQL2 indicati nelle note bibliografiche. Nei due paragrafi successivi sono esaminate le funzionalità d'interrogazione di SQL.

8.2 Interrogazioni fondamentali in SQL

SQL ha un'istruzione fondamentale per recuperare le informazioni da una base di dati: l'**istruzione SELECT**. Essa *non ha alcun rapporto* con l'operazione SELECT dell'algebra relazionale trattata nel Capitolo 7. Vi sono molte opzioni e varianti dell'istruzione SELECT in SQL, di conseguenza le sue funzionalità saranno presentate gradualmente. Verranno usate interrogazioni di esempio specificate sullo schema di Figura 7.5 e si farà riferimento allo stato della base di dati esemplificativa di Figura 7.6 per mostrare i risultati di alcune interrogazioni.

Prima di proseguire occorre mettere in evidenza un'importante distinzione tra SQL e il modello relazionale trattato nel Capitolo 7: SQL consente a una tabella (relazione) di avere due o più tuple identiche in tutti i valori dei loro attributi. In generale, quindi, una **tabella SQL** non è un *insieme di tuple*, perché un insieme non permette due membri uguali; piuttosto è un **multinsieme**, talvolta chiamata *sacca* (bag) di tuple. Alcune relazioni SQL non possono che essere insiemi perché è stato dichiarato un vincolo di chiave o perché l'opzione DISTINCT è stata usata con l'istruzione SELECT (descritta successivamente). Si tenga presente questa distinzione esaminando gli esempi.

8.2.1 La struttura SELECT-FROM-WHERE delle interrogazioni SQL

La forma fondamentale dell'istruzione SELECT, talvolta chiamata **blocco select-from-where**, è formata dalle tre clausole SELECT, FROM e WHERE e ha la seguente forma:

```
SELECT <elenco attributi>
FROM <elenco tabelle>
WHERE <condizione>;
```

in cui:

- <elenco attributi>: è un elenco dei nomi degli attributi i cui valori devono essere recuperati dall'interrogazione;
- <elenco tabelle>: è un elenco dei nomi delle relazioni necessarie per eseguire l'interrogazione;
- <condizione>: è un'espressione condizionale (Booleana) che identifica le tuple che devono essere recuperate dall'interrogazione.

L'istruzione fondamentale SELECT verrà ora illustrata tramite alcune interrogazioni esemplificative.

INTERROGAZIONE 1. Si recuperino la data di nascita e l'indirizzo del dipendente (o dei dipendenti) il cui nome è 'John Smith'.

```
I1:   SELECT DATA_N, INDIRIZZO
      FROM IMPiegato
     WHERE NOME_BATT='John' AND COGNOME='Smith';
```

Questa interrogazione coinvolge solo la relazione IMPiegato elencata nella clausola FROM. L'interrogazione *seleziona* le tuple di IMPiegato che soddisfano la condizione della clausola WHERE, poi *proietta* il risultato sugli attributi DATA_N e INDIRIZZO elencati nella clausola SELECT. I1 è simile alla seguente espressione dell'algebra relazionale, a eccezione del fatto che i duplicati, se ci sono, non vengono eliminati:

^aDATAN, INDIRIZZO ("NOME_BATT = 'John' AND COGNOME = 'Smith'" (IMPiegato))

Una semplice interrogazione PROJECT SQL con un singolo nome nella clausola FROM, quindi, è simile alla coppia SELECT-PROJECT di operazioni dell'algebra relazionale. La clausola SELECT di SQL specifica gli *attributi di proiezione* e la clausola WHERE specifica la *condizione di selezione*. L'unica differenza è che nell'interrogazione SQL è possibile ottenere delle tuple duplicate nel risultato dell'interrogazione, perché SQL non impone il vincolo che la relazione sia un insieme. In Figura 8.2(a) è mostrato il risultato dell'interrogazione I1 sulla base di dati di Figura 7.6.

INTERROGAZIONE 2. Si recuperino il nome e l'indirizzo di tutti i dipendenti che lavorano per il dipartimento 'Ricerca'.

```
I2:   SELECT NOME_BATT, COGNOME, INDIRIZZO
      FROM IMPiegato, DIPARTIMENTO
     WHERE NOME_D='Ricerca' AND NUMERO_D=N_D;
```

L'Interrogazione I2 è simile a una sequenza SELECT-PROJECT-JOIN di operazioni dell'algebra relazionale. Queste interrogazioni spesso sono chiamate **interrogazioni select-project-join**. Nella clausola WHERE di I2, la condizione NOME_D = 'Ricerca' è una *condizione di selezione* e corrisponde a un'operazione SELECT nell'algebra relazionale. La con-

(a) DATA_N		INDIRIZZO								
1965-01-09		731 Fondren, Houston, TX								
(b) NOME_BATT		COGNOME	INDIRIZZO							
John		Smith	731 Fondren, Houston, TX							
Franklin		Wong	638 Voss, Houston, TX							
Ramesh		Narayan	975 Fire Oak, Humble, TX							
Joyce		English	5631 Rice, Houston, TX							
(c) NUMERO_P		NUM_D	COGNOME	INDIRIZZO	DATA_N					
10		4	Wallace	291 Berry, Bellaire, TX	1941-06-20					
30		4	Wallace	291 Berry, Bellaire, TX	1941-06-20					
(d) I.NOME_BATT		I.COGNOME	S.NOME_BATT	S.COGNOME						
John		Smith	Franklin	Wong						
Franklin		Wong	James	Borg						
Alicia		Zelaya	Jennifer	Wallace						
Jennifer		Wallace	James	Borg						
Ramesh		Narayan	Franklin	Wong						
Joyce		English	Franklin	Wong						
Ahmed		Jabbar	Jennifer	Wallace						
(e) SSN										
123456789										
333445555										
999887777										
987654321										
666884444										
453453453										
987987987										
888665555										
(g) NOME_BATT		INIZ_INT	COGNOME	SSN	DATA_N	INDIRIZZO	SESSO	STIPENDIO	SUPERSSN	N_D
John		B	Smith	123456789	1965-09-01	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin		T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Ramesh		K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce		A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

Figura 8.2 I risultati delle interrogazioni specificate sulle basi di dati della Figura 7.6.
 (a) Risultato di I1.
 (b) Risultato di I2.
 (c) Risultato di I3.
 (d) Risultato di I4.
 (e) Risultato di I5.
 (f) Risultato di I6.
 (g) Risultato di I2C.

dizione $\text{NUMERO_D} = \text{N_D}$ è una **condizione di join** che corrisponde alla condizione JOIN dell'algebra relazionale. Il risultato dell'Interrogazione I2 è mostrato in Figura 8.2(b). In generale in una sola interrogazione SQL è possibile specificare un numero qualsiasi di condizioni di join e di selezione. L'esempio che segue è un'interrogazione select-project-join con *due* condizioni di join.

INTERROGAZIONE 3. Per ogni progetto con sede a 'Stafford', si elenchi il numero del progetto, il numero del dipartimento che lo controlla, nonché il cognome, l'indirizzo e la data di nascita del direttore del dipartimento.

I3: **SELECT** NUMERO_P, NUM_D, COGNOME, INDIRIZZO, DATA_N
FROM PROGETTO, DIPARTIMENTO, IMPIEGATO
WHERE NUM_D=NUMERO_D AND SSN_DIR=SSN AND
 SEDE_P='Stafford';

La condizione di join $\text{NUM_D} = \text{NUMERO_D}$ mette in relazione un progetto con il dipartimento che lo controlla, mentre la condizione di join $\text{SSN_DIR} = \text{SSN}$ collega il dipartimento che lo controlla al dipendente che dirige quel dipartimento. Il risultato dell'interrogazione I3 è mostrato in Figura 8.2 (c).

8.2.2 La problematica dei nomi di attributi ambigui e la loro ridenominazione (assegnando pseudonimi)

In SQL lo stesso nome può essere usato per due (o più) attributi purché questi si trovino in *relazioni diverse*. In questo caso, se un'interrogazione si riferisce a due o più attributi con il medesimo nome, si deve **qualificare** il nome dell'attributo con il nome della relazione per evitare ambiguità. Ciò viene fatto *ponendo come prefisso* il nome della relazione al nome dell'attributo e separando i due con un punto. Per illustrare questo caso, si supponga che nelle Figure 7.5 e 7.6 gli attributi N_D e COGNOME della relazione IMPIEGATO siano stati chiamati NUMERO_D e NOME e l'attributo NOME_D di DIPARTIMENTO sia stato denominato NOME; in seguito, per evitare ambiguità, l'Interrogazione I2 va riformulata come mostrato in I2A. Nell'Interrogazione I2A si sono dovuti premettere dei prefissi agli attributi NOME e NUMERO_D per evitare ambiguità, visto che gli stessi nomi di attributi sono usati in entrambe le relazioni:

I2A: **SELECT** NOME_BATT, IMPIEGATO.NOME, INDIRIZZO
FROM IMPIEGATO, DIPARTIMENTO
WHERE DIPARTIMENTO.NOME='Ricerca' AND
 DIPARTIMENTO.NUMERO_D=IMPIEGATO.NUMERO_D;

L'ambiguità sorge anche nel caso di interrogazioni che si riferiscono due volte alla medesima relazione, come nell'esempio seguente.

INTERROGAZIONE 4. Per ciascun dipendente si recuperi il suo nome e il suo cognome e il nome e il cognome del suo immediato supervisore.

I4: **SELECT** I.NOME_BATT, I.COGNOME, S.NOME_BATT, S.COGNOME
FROM IMPIEGATO AS I, IMPIEGATO AS S
WHERE I.SUPERSSN=S.SSN;

In questo caso è consentito dichiarare nomi di relazione alternativi I e S chiamati **pseudonimi (aliases)** o **variabili di tupla** per la relazione IMPIEGATO. Uno pseudonimo può seguire la parola chiave AS, come mostrato precedentemente in I4, oppure può seguire direttamente

(a) STIPENDIO	(b) STIPENDIO
30000	30000
40000	40000
25000	25000
43000	43000
38000	38000
25000	55000
25000	
55000	

(c) NOME_BATT COGNOME	(d) NOME_BATT COGNOME
	James
	Borg

Figura 8.3 I risultati di alcune interrogazioni specificate sulla base di dati mostrata in Figura 7.6. (a) Risultato di I7. (b) Risultato di I7A. (c) Risultato di I14. (d) Risultato di I19.

- l'utente può voler vedere le tuple duplicate nel risultato di un'interrogazione;
- quando una funzione di aggregazione (si veda il Sottoparagrafo 8.3.5) è applicata alle tuple, nella maggior parte dei casi non si vogliono eliminare i duplicati.

Una tabella SQL con una chiave deve però necessariamente essere un insieme, poiché il valore della chiave deve essere differente in ciascuna tupla.⁵ Quando *si desidera* eliminare le tuple duplicate dal risultato di un'interrogazione SQL, si utilizza la parola chiave **DISTINCT** nella clausola SELECT, il che significa che solo le tuple distinte dovrebbero rimanere nel risultato. In generale un'interrogazione con SELECT DISTINCT elimina i duplicati, mentre un'interrogazione con SELECT ALL non lo fa (specificando SELECT senza le opzioni ALL o DISTINCT, si ottiene un effetto equivalente a SELECT ALL). Ad esempio, l'Interrogazione I7 recupera lo stipendio di ogni dipendente; se più impiegati hanno lo stesso stipendio, il valore dello stipendio apparirà più volte nel risultato dell'interrogazione, come mostrato in Figura 8.3(a). Se si è interessati solo a valori dello stipendio distinti, si vuole che ciascun valore appaia solo una volta, indipendentemente da quanti dipendenti guadagnano quello stipendio. Usando la parola chiave **DISTINCT** come nella seconda delle interrogazioni I7 si ottiene tale risultato, mostrato in Figura 8.3(b).

INTERROGAZIONE 7. Si recuperi lo stipendio di ogni dipendente nella prima interrogazione riportata qui di seguito, poi, nelle successive, tutti i valori distinti dello stipendio.

```
I7:   SELECT ALL      STIPENDIO
      FROM        IMPiegato;
I7A:  SELECT DISTINCT STIPENDIO
      FROM        IMPiegato;
```

Alcune operazioni insiemistiche dell'algebra relazionale sono state incorporate direttamente in SQL. Vi è l'operazione di unione insiemistica (UNION), e in SQL2 vi sono anche

le operazioni differenza (EXCEPT) e intersezione (INTERSECT).⁶ Le relazioni da esse derivanti sono insiemi di tuple, cioè le *tuple duplicate sono eliminate dal risultato*. Poiché tali operazioni si applicano solo alle *relazioni compatibili all'unione*, ci si deve assicurare che le due relazioni su cui si applica un'operazione insiemistica abbiano gli stessi attributi e che gli attributi appaiano nel medesimo ordine in entrambe le relazioni. L'esempio successivo illustra l'utilizzo di UNION.

INTERROGAZIONE 8. Si crei un elenco di tutti i numeri di progetto dei progetti che coinvolgono un dipendente il cui cognome è 'Smith', come partecipante oppure come dirigente del dipartimento che controlla il progetto.

```
I8:   (SELECT DISTINCT NUMERO_P
      FROM    PROGETTO, DIPARTIMENTO, IMPiegato
      WHERE   NUM_D=NUMERO_D AND SSN_DIR=SSN AND
              COGNOME='Smith')
      UNION
      (SELECT DISTINCT NUMERO_P
      FROM    PROGETTO, Lavora_su, IMPiegato
      WHERE   NUMERO_P=N_P AND ISSN=SSN AND COGNOME='Smith');
```

La prima interrogazione SELECT recupera i progetti che coinvolgono 'Smith' come dirigente del dipartimento che controlla il progetto e la seconda recupera i progetti che coinvolgono uno 'Smith' come partecipante al progetto. Si noti che se più dipendenti hanno il cognome 'Smith', saranno recuperati i nomi del progetto che coinvolgono uno qualsiasi di essi. L'applicazione dell'operazione UNION alle due interrogazioni selezionate dà il risultato desiderato.

8.2.5 Confronti di sottostringhe, operatori aritmetici e ordinamento

In questo paragrafo verranno trattate altre funzioni di SQL. La prima funzione permette di eseguire un confronto solo su parti di una stringa di caratteri, usando l'operatore di confronto LIKE. Le stringhe parziali sono specificate utilizzando due caratteri riservati: il segno di per centuale (%) sostituisce un numero arbitrario di caratteri e il segno di sottolineatura (_) sostituisce un singolo carattere. Ad esempio si consideri l'interrogazione seguente.

INTERROGAZIONE 9. Si recuperino tutti i dipendenti il cui indirizzo è a Houston, in Texas.

⁵ In generale non è richiesto che una tabella SQL abbia una chiave anche se nella maggior parte dei casi ce ne sarà una.

⁶ SQL2 comprende anche delle operazioni sui multinsieme che sono seguite dalla parola chiave ALL (UNION ALL, EXCEPT ALL, INTERSEC ALL). I loro risultati sono multinsieme.

I9: SELECT NOME_BATT, COGNOME
 FROM IMPiegato
 WHERE INDIRIZZO LIKE "%Houston, TX%";

Per recuperare tutti i dipendenti che sono nati negli anni cinquanta, si può usare l'Interrogazione I27. Qui "5" deve essere il terzo carattere della stringa (secondo il formato visto per la data), quindi si utilizza il valore "_5_____", in cui ogni carattere di sottolineatura⁷ serve come indicatore di un carattere arbitrario.

INTERROGAZIONE 10A. Si trovino tutti i dipendenti che sono nati negli anni cinquanta.

I10A: SELECT NOME_BATT, COGNOME
 FROM IMPiegato
 WHERE DATA_N LIKE '_ _ 5 _____';

Altre funzionalità permettono l'utilizzo dell'aritmetica nelle interrogazioni. Gli operatori aritmetici standard per l'addizione (+), la sottrazione (-), la moltiplicazione (*) e la divisione (/) possono essere applicati ai valori numerici o agli attributi con domini numerici. Ad esempio, si supponga di voler vedere l'effetto di un aumento del 10 per cento dato a tutti i dipendenti che lavorano sul progetto 'ProdottoX'; si può applicare l'Interrogazione I11 per vedere come diventerebbero i loro stipendi.

INTERROGAZIONE 11. Si mostrino i salari risultanti, dando a ogni dipendente che lavora sul progetto "ProdottoX" un aumento del 10 per cento.

I11: SELECT NOME_BATT, COGNOME, 1.1* STIPENDIO
 FROM IMPiegato, Lavora_su, PROGETTO
 WHERE SSN=SSN_I AND N_P=NUMERO_P AND NOME_P='ProdottoX';

Per quanto riguarda le stringhe, l'operatore di concatenazione '||' può essere utilizzato in un'interrogazione per giustapporre i due valori stringa. Per i tipi di dati data, ora, timestamp e un intervallo, gli operatori disponibili sono il segno di aumento (+) o di diminuzione (-) di una intervallo, di un'ora o timestamp di una quantità compatibile con il tipo. Inoltre si può specificare un intervallo come differenza tra due valori data, ora o timestamp. Un altro operatore di confronto che può essere usato per comodità è **BETWEEN**, illustrato nell'Interrogazione I12.⁸

INTERROGAZIONE 12. Si recuperino tutti i dipendenti del dipartimento 5 il cui stipendio è compreso tra 30.000 e 40.000 dollari.

I12: SELECT *
 FROM IMPiegato
 WHERE (STIPENDIO BETWEEN 30000 AND 40000) AND N_D = 5;

SQL consente all'utente di ordinare le tuple del risultato di un'interrogazione rispetto ai valori di uno o più attributi, usando la clausola **ORDER BY**, come è illustrato nell'Interrogazione I13.

INTERROGAZIONE 13. Si recuperi un elenco di dipendenti e i progetti su cui lavorano, ordinati per dipartimento e, all'interno di ciascun dipartimento, li si ordini alfabeticamente per cognome e nome.

I13: SELECT NOME_D, COGNOME, NOME_BATT, NOME_P
 FROM DIPARTIMENTO, IMPiegato, Lavora_su, PROGETTO
 WHERE NUMERO_D=N_D AND SSN=SSN_I AND
 N_P=NUMERO_P
 ORDER BY NOME_D, COGNOME, NOME_BATT;

L'ordine predefinito è l'ordine crescente dei valori. Si può specificare la parola chiave **DESC** se si vuole l'ordine decrescente. La parola chiave **ASC** può essere utilizzata per specificare esplicitamente l'ordine crescente. Se si desidera un ordine decrescente su **NOME_D** e uno crescente su **COGNOME**, **NOME_BATT**, la clausola **ORDER BY** dell'interrogazione I13 diventa

ORDER BY NOME_D DESC, COGNOME ASC, NOME_BATT ASC

8.3 Interrogazioni SQL più complesse

Nel paragrafo precedente sono stati descritti i tipi fondamentali di interrogazioni SQL. A causa della generalità e del potere espressivo del linguaggio, vi sono molte altre funzioni che consentono agli utenti di specificare interrogazioni più complesse. Nel seguito, ne saranno esaminate alcune.

8.3.1 Interrogazioni nidificate e confronti di insiemi

Alcune interrogazioni richiedono di estrarre dei valori esistenti nella base di dati per poi usarli in una condizione di confronto. Tali interrogazioni possono essere convenientemente formulate usando delle **interrogazioni nidificate**, che sono blocchi completi **SELECT . . . FROM . . . WHERE** posti all'interno della clausola **WHERE** di un'altra interrogazione chiamata **interrogazione esterna**. L'Interrogazione I8 è stata formulata senza un'interrogazione

⁷ Se i simboli di sottolineatura o di percentuale % sono caratteri presenti nella stringa, dovrebbero essere preceduti da un *carattere escape*, specificato dopo la stringa; ad esempio, 'AB\CD%\EF' ESC '\V rappresenta la stringa 'AB_CD%\EF'.

⁸ La condizione (STIPENDIO BETWEEN 30000 AND 40000) equivale a ((STIPENDIO ≥ 30000) AND (STIPENDIO ≤ 40000)).

nidificata, ma può essere riformulata per utilizzare le interrogazioni nidificate come mostrato qui di seguito:

```
I8A:   SELECT DISTINCT NUMERO_P
      FROM PROGETTO
     WHERE NUMERO_P IN (SELECT NUMERO_P
                          FROM PROGETTO, DIPARTIMENTO,
                               IMPIEGATO
                         WHERE NUMERO_D=NUMERO_P AND
                               SSN_DIR=SSN AND
                               COGNOME='Smith')

          OR

      NUMERO_P IN (SELECT N_P
                    FROM LAVORA_SU, IMPIEGATO
                   WHERE SSN_I=SSN AND
                         COGNOME='Smith');
```

La prima interrogazione nidificata seleziona i numeri di progetto dei progetti di cui 'Smith' è dirigente, mentre la seconda seleziona i numeri dei progetti che hanno 'Smith' coinvolto come partecipante. Nell'interrogazione esterna si seleziona una tupla PROGETTO se il valore NUMERO_P di quella tupla fa parte del risultato di una delle due interrogazioni nidificate. L'operatore di confronto IN confronta un valore v con un insieme (o multinsieme) di valori V e restituisce il valore TRUE se v è uno degli elementi di V .

L'operatore IN può anche confrontare una tupla di valori tra parentesi con un insieme o un multinsieme di tuple compatibili all'unione. Ad esempio l'interrogazione:

```
SELECT DISTINCT SSN_I
      FROM LAVORA_SU
     WHERE (N_P, ORE) IN (SELECT N_P, ORE FROM LAVORA_SU
                           WHERE SSN='123456789');
```

selezionerà i numeri di servizio sanitario di tutti i dipendenti che hanno la stessa combinazione (progetto, ore) su un progetto in cui lavora il dipendente 'John Smith' (il cui SSN = "123456789").

Oltre all'operatore IN, possono essere usati altri operatori di confronto per confrontare un singolo valore v (tipicamente un nome di attributo) con un insieme o un multinsieme V (tipicamente il risultato di un'interrogazione nidificata). L'operatore = ANY (oppure = SOME) restituisce TRUE se il valore v è uguale a *un qualsiasi valore* nell'insieme V ed è quindi equivalente a IN. Le parole chiave ANY e SOME hanno lo stesso significato. Altri operatori che possono essere usati con ANY (o SOME) includono $>$, $>=$, $<$, $<=$ e \neq . Anche la parola chiave ALL può essere utilizzata insieme a ciascuno di questi operatori. Ad esempio la condizione di confronto ($v > \text{ALL } V$) restituisce TRUE se il valore v è maggiore di *tutti* i valori dell'insieme V . Un esempio è l'interrogazione seguente, che restituisce i nomi dei dipendenti il cui stipendio è superiore di quello di tutti i dipendenti del dipartimento 5:

```
SELECT COGNOME, NOME_BATT
      FROM IMPIEGATO
     WHERE STIPENDIO > ALL (SELECT STIPENDIO FROM IMPIEGATO WHERE
                                N_D=5);
```

In generale si possono avere parecchi livelli di interrogazioni nidificate. È possibile ancora una volta che vi sia ambiguità tra i nomi degli attributi se esistono attributi con lo stesso nome, una volta in una relazione nella clausola FROM dell'*interrogazione esterna* e la seconda volta nella relazione nella clausola FROM dell'*interrogazione nidificata*. La regola è che i riferimenti ad *attributi non qualificati* si riferiscono alla relazione dichiarata nell'*interrogazione nidificata più interna*. Ad esempio nella clausola SELECT e nella clausola WHERE della prima interrogazione nidificata di Q4A, un riferimento a un attributo non qualificato della relazione PROGETTO si riferisce alla relazione PROGETTO specificata nella clausola FROM dell'interrogazione nidificata. Per far riferimento a un attributo della relazione PROGETTO specificata nell'interrogazione esterna, si può utilizzare e fare riferimento a uno *pseudonimo* per quella relazione. Queste regole sono simili a quelle di visibilità per le variabili in un linguaggio di programmazione come PASCAL, che consente funzioni e procedure nidificate. Per illustrare la potenziale ambiguità dei nomi degli attributi nelle interrogazioni nidificate si prende in considerazione l'Interrogazione I14, il cui risultato è mostrato in Figura 8.3(c).

INTERROGAZIONE 14. Si recuperi il nome di ciascun dipendente che ha una persona a carico con il medesimo nome e lo stesso sesso del dipendente.

```
I14:   SELECT I.NOME_BATT, I.COGNOME
        FROM IMPIEGATO AS I
       WHERE I.SSN IN (SELECT SSN_I
                          FROM PERSONA_A_CARICO
                         WHERE I.NOME_BATT=NOME_PERSONA_A_CARICO
                               AND I_SESSO=SESSO);
```

Nell'interrogazione nidificata dell'Interrogazione I14 si deve qualificare I_SESSO perché si riferisce all'attributo SESSO della relazione IMPIEGATO dell'interrogazione esterna e anche la relazione PERSONA_A_CARICO ha un attributo chiamato SESSO. Tutti i riferimenti non qualificati a SESSO nell'interrogazione nidificata, si riferiscono all'attributo SESSO di PERSONA_A_CARICO. Tuttavia non si devono qualificare NOME_BATT e SSN perché la relazione PERSONA_A_CARICO non ha attributi chiamati NOME_BATT e SSN, quindi non vi è ambiguità.

Le interrogazioni nidificate correlate. Quando una condizione nella clausola WHERE di una interrogazione nidificata si riferisce ad attributi di una relazione dichiarata nell'interrogazione esterna, si dice che le due interrogazioni sono *correlate*. Si può capire meglio il concetto di interrogazione correlata considerando che l'*interrogazione nidificata è valutata una volta per ciascuna tupla (o combinazione di tuple) dell'interrogazione esterna*. Ad esempio, si può pensare a I14 nel seguente modo: per ciascuna tupla di IMPIEGATO, si valuta l'interrogazione nidificata, che recupera i valori di SSN_I per tutte le tuple PERSONA_A_CARICO con lo stesso sesso e nome della tupla di IMPIEGATO; se il valore di SSN della tupla di IMPIEGATO è nel risultato dell'interrogazione nidificata, si seleziona quella tupla da IMPIEGATO.

In generale un'interrogazione scritta con i blocchi SELECT ... FROM ... WHERE ... di EXISTS e usando gli operatori di confronto = o IN può *sempre* essere espressa come un'interrogazione di un singolo blocco. Ad esempio I14 può essere scritta come in I14A:

```
I14A: SELECT I.NOME_BATT, I.COGNOME
      FROM IMPIEGATO AS I, PERSONA_A_CARICO AS D
      WHERE I.SSN=D.SSN_I AND I.SESSO=D.SESSO AND
            I.NOME_BATT=D.NOME_PERSONA_A_CARICO;
```

L'implementazione originale di SQL di SYSTEM R aveva anche un operatore di confronto **CONTAINS**, usato per confrontare due insiemi o due multinsiemi. Questo operatore successivamente è stato eliminato dal linguaggio, probabilmente a causa della difficoltà di implementarlo in modo efficace, e la maggior parte delle implementazioni commerciali di SQL non lo comprende. L'operatore **CONTAINS** confronta due insiemi di valori e restituisce TRUE se un insieme contiene tutti i valori dell'altro insieme. L'Interrogazione I15 illustra l'utilizzo dell'operatore **CONTAINS**.

INTERROGAZIONE 15. Si recuperi il nome di ogni dipendente che lavora su *tutti* i progetti controllati dal dipartimento numero 5.

```
I15:  SELECT NOME_BATT, COGNOME
      FROM IMPIEGATO
      WHERE ((SELECT N_P
              FROM LAVORA_SU
              WHERE SSN=SSN_I)
             CONTAINS
             (SELECT NUMERO_P
              FROM PROGETTO
              WHERE NUM_D=5));
```

Nell'Interrogazione I15, la seconda interrogazione nidificata (che non è correlata all'interrogazione esterna) recupera i numeri di progetto di tutti i progetti controllati dal dipartimento 5. Per *ciascuna* tupla relativa a un impiegato, la prima interrogazione nidificata (che è correlata) recupera i numeri del progetto in cui lavora l'impiegato; se questi contengono progetti tutti gestiti dal dipartimento 5, la tupla dell'impiegato è selezionata e viene recuperato il nome dell'impiegato. Si noti che l'operatore di confronto **CONTAINS** è simile all'operazione **DIVISION** dell'algebra relazionale descritta nel Sottoparagrafo 7.4.7. Poiché l'operazione **CONTAINS** non fa parte di SQL, bisogna usare la funzione **EXISTS** per specificare questi tipi di interrogazioni, come si vedrà nel Sottoparagrafo 8.3.2.

8.3.2 Funzioni EXISTS e UNIQUE

La funzione **EXISTS** in SQL è usata per controllare se il risultato di un'interrogazione nidificata e correlata è vuoto (non contiene tuple) oppure no. Nel seguito, verrà illustrato l'utilizzo

di **EXISTS**, e anche di **NOT EXISTS**, tramite esempi. Prima di tutto viene formulata l'Interrogazione I14 in una forma alternativa che utilizza **EXISTS**. Questa formulazione è mostrata qui di seguito.

```
I14B: SELECT I.NOME_BATT, I.COGNOME
      FROM IMPIEGATO AS I
      WHERE EXISTS (SELECT *
                     FROM PERSONA_A_CARICO
                     WHERE I.SSN=SSN_I AND I.SESSO=SESSO AND
                           I.NOME_BATT=NOME_PERSONA_A_CARICO);
```

Le funzioni **EXISTS** e **NOT EXISTS** di solito sono usate insieme a un'interrogazione nidificata correlata. Nell'esempio sopra riportato, l'interrogazione nidificata si riferisce agli attributi **SSN**, **NOME_BATT** e **SESSO** della relazione **IMPIEGATO** dell'interrogazione esterna, quindi si può pensare nel seguente modo: per ciascuna tupla di **IMPIEGATO**, si valuta l'interrogazione nidificata, che recupera tutte le tuple **PERSONA_A_CARICO** con lo stesso numero di servizio sanitario, sesso e nome della tupla di **IMPIEGATO**; se nel risultato dell'interrogazione nidificata **EXISTS** almeno una tupla, allora si seleziono quella tupla di **IMPIEGATO**. In generale **EXISTS(Q)** restituisce **TRUE** se c'è *almeno una tupla* nel risultato dell'interrogazione **Q**, altrimenti restituisce **FALSE**. D'altra parte **NOT EXISTS(Q)** restituisce **TRUE** se *non vi sono tuple* nel risultato dell'interrogazione **Q**, altrimenti restituisce **FALSE**. Nel seguito verrà detto l'utilizzo di **NOT EXISTS**.

INTERROGAZIONE 16. Si recuperino i nomi dei dipendenti che non hanno persone a carico.

```
I16:  SELECT NOME_BATT, COGNOME
      FROM IMPIEGATO
      WHERE NOT EXISTS (SELECT *
                         FROM PERSONA_A_CARICO
                         WHERE SSN=SSN_I);
```

In I16 l'interrogazione nidificata correlata recupera tutte le tuple **DIPENDENTE** relative a una tupla di **IMPIEGATO**. Se *non ne esistono*, è selezionata la tupla di **IMPIEGATO**. Si può spiegare I16 nel modo seguente: per *ciascuna* tupla di **IMPIEGATO**, l'interrogazione nidificata correlata seleziona tutte le tuple **PERSONA_A_CARICO** il cui valore di **SSN_I** corrisponde al valore di **SSN** in **IMPIEGATO**; se il risultato è vuoto, l'impiegato non ha nessuna persona a carico, quindi si seleziona la tupla di **IMPIEGATO** e si recuperano il suo nome e il suo cognome, cioè **NOME_BATT** e **COGNOME**. Vi è un'altra funzione di SQL, **UNIQUE(Q)**, che restituisce **TRUE** se non vi sono tuple duplicate nel risultato dell'interrogazione **Q**; altrimenti restituisce **FALSE**.

INTERROGAZIONE 17. Si elenchino i nomi dei dipendenti dirigenti di dipartimento che hanno almeno un sottoposto.

```
I17:  SELECT NOME_BATT, COGNOME
      FROM IMPIEGATO
     WHERE EXISTS (SELECT *
                     FROM PERSONA_A_CARICO
                    WHERE SSN=SSN_I)
           AND
          EXISTS (SELECT *
                     FROM PERSONA_A_CARICO
                    WHERE SSN=SSN_DIR);
```

Un modo per scrivere questa interrogazione è mostrato in I17, in cui si specificano due interrogazioni nidificate correlate: la prima seleziona tutte le tuple PERSONA_A_CARICO collegate a un IMPIEGATO, la seconda seleziona tutte le tuple DIPARTIMENTO gestite da IMPIEGATO. Se esiste almeno una tupla nel risultato della prima interrogazione e almeno una in quello della seconda, si seleziona la tupla di IMPIEGATO. È possibile riscrivere questa interrogazione usando solo una oppure nessuna interrogazione nidificata?

L'Interrogazione I15, che è servita a illustrare l'operatore di confronto CONTAINS, può essere specificata usando EXISTS e NOT EXISTS nei sistemi SQL. Vi sono due opzioni. La prima è utilizzare la ben nota trasformazione della teoria degli insiemi che dice che se $(S1 \text{ } CONTAINS \text{ } S2)$ è logicamente equivalente a $(S2 \text{ } EXCEPT } S1)$, è vuoto,⁹ come presentato qui di seguito.

```
I15A: SELECT NOME_BATT, COGNOME
      FROM IMPIEGATO
     WHERE NOT EXISTS
           ((SELECT NUMERO_P
              FROM PROGETTO
             WHERE NUM_D=5)
            EXCEPT
           (SELECT N_P
              FROM LAVORA_SU
             WHERE SSN=SSN_I));
```

Un'ulteriore opzione è illustrata qui di seguito. Si noti che è necessaria una nidificazione a due livelli e che questa formulazione è un po' più complessa della I15, che usava l'operatore di confronto CONTAINS e anche della interrogazione sopra riportata, che utilizza NOT EXISTS e EXCEPT. Tuttavia, va notato che la clausola CONTAINS non fa parte di SQL e non tutti i sistemi relazionali hanno l'operatore EXCEPT anche se fa parte dello standard SQL2.

Nell'ulteriore variazione dell'Interrogazione I15 sotto riportata:

```
I15B: SELECT COGNOME, NOME_BATT
      FROM IMPIEGATO
```

```
WHERE NOT EXISTS
      (SELECT *
         FROM LAVORA_SU B
        WHERE (B.N_P IN (SELECT NUMERO_P
                          FROM PROGETTO
                         WHERE NUM_D=5))
              AND
             NOT EXISTS (SELECT*
                           FROM LAVORA_SU C
                          WHERE C.SSN_I=SSN
                            AND
                           C.N_P=B.N_P));
```

l'interrogazione nidificata esterna seleziona le tuple LAVORA_SU (B) il cui N_P è di un progetto controllato dal dipartimento 5, se non vi è una tupla LAVORA_SU (C) con lo stesso N_P e lo stesso SSN della tupla di IMPIEGATO specificata che l'interrogazione esterna sta prendendo in considerazione. Se non esiste alcuna tupla di questo tipo, si seleziona la tupla di IMPIEGATO. La forma di I15B corrisponde alla seguente riformulazione dell'Interrogazione I15: si seleziono ogni impiegato tale che non esista alcun progetto controllato dal dipartimento 5 in cui l'impiegato non stia lavorando.

Si noti che l'Interrogazione I15 può essere espressa nell'algebra relazionale usando l'operazione DIVISION.

8.3.3 Insiemi esplicativi e NULLS

Si sono viste diverse interrogazioni con un'interrogazione nidificata nella clausola WHERE. È possibile usare anche un insieme esplicito di valori nella clausola WHERE, piuttosto che un'interrogazione nidificata. Tale insieme va posto tra parentesi in SQL.

INTERROGAZIONE 18. Si recuperino i numeri del servizio sanitario di tutti i dipendenti che lavorano sui progetti numero 1, 2 o 3.

```
I18:  SELECT DISTINCT SSN_I
      FROM LAVORA_SU
     WHERE N_P IN (1, 2, 3);
```

SQL consente interrogazioni che controllano se un valore è **NULL**, mancante o non definito oppure non applicabile. Tuttavia, invece di usare = o ≠ per confrontare un attributo con NULL, SQL utilizza IS oppure IS NOT. Questo perché SQL considera ciascun valore nullo come distinto da ogni altro valore nullo, quindi il confronto di uguaglianza non è appropriato. Ne segue che, quando è specificata una condizione di join, le tuple con valori nulli per gli attributi di join non sono inclusi nel risultato (a meno che si tratti di un OUTER JOIN, cioè un

⁹ Si ricordi che EXCEPT è l'operatore di differenza fra insiemi.

8.3.5 Funzioni di aggregazione e raggruppamento

Nel Sottoparagrafo 7.5.1 si è introdotto il concetto della funzione di aggregazione come un'operazione relazionale. Poiché il raggruppamento e l'aggregazione sono richiesti in molte applicazioni di basi di dati, SQL comprende funzioni che implementano tali concetti, tra cui alcune funzioni *buil-in* (integrate): COUNT, SUM, MAX, MIN e AVG. La funzione COUNT restituisce il numero di tuple o valori individuati da un'interrogazione. Le funzioni SUM, MAX, MIN e AVG sono applicate a un insieme o a un multinsieme di valori numerici e restituiscono rispettivamente la somma, il valore massimo, il valore minimo e la media di quei valori. Queste funzioni possono essere usate nella clausola SELECT o nella clausola HAVING (si veda oltre). Le funzioni MAX e MIN possono essere utilizzate anche con attributi che hanno domini non numerici se tra i valori di dominio è definito un *ordinamento totale*.¹⁰

INTERROGAZIONE 20. Si calcoli la somma degli stipendi di tutti gli impiegati, lo stipendio massimo, quello minimo e quello medio.

```
I20:   SELECT SUM (STIPENDIO), MAX (STIPENDIO), MIN (STIPENDIO),
          AVG (STIPENDIO)
     FROM   IMPiegato;
```

Se si vogliono ottenere i valori delle funzioni precedenti per gli impiegati di uno specifico dipartimento, ad esempio il dipartimento 'Ricerca', si può scrivere l'Interrogazione 20, in cui le tuple di IMPiegato sono limitate dalla clausola WHERE a quegli impiegati che lavorano per il dipartimento 'Ricerca'.

INTERROGAZIONE 21. Si calcoli la somma degli stipendi di tutti gli impiegati del dipartimento 'Ricerca' e anche lo stipendio massimo, quello minimo e quello medio di questo dipartimento.

```
I21:   SELECT SUM (STIPENDIO), MAX (STIPENDIO), MIN (STIPENDIO),
          AVG (STIPENDIO)
     FROM   IMPiegato, DIPARTIMENTO
    WHERE  N_D=NUMERO_D AND NOME_D='Ricerca';
```

INTERROGAZIONE 22. Si recuperi il numero totale degli impiegati della società:

```
I22:   SELECT COUNT (*)
     FROM   IMPiegato;
```

¹⁰ Ordinamento totale significa che presi due valori qualsiasi nel dominio, si può determinare quale compare prima dell'altro nell'ordinamento; ad esempio DATE, TIME e i domini TIMESTAMP hanno degli ordinamenti totali sui loro valori, così come le stringhe alfabetiche.

e il numero dei dipendenti del dipartimento 'Ricerca' nell'interrogazione I23:

```
I23:   SELECT COUNT (*)
     FROM   IMPiegato, DIPARTIMENTO
    WHERE  N_D=NUMERO_D AND NOME_D='Ricerca';
```

Qui l'asterisco (*) si riferisce alle *righe* (tuple), quindi COUNT (*) restituisce il numero di righe nel risultato dell'interrogazione. Si può usare anche la funzione COUNT per calcolare i valori presenti in una colonna invece delle tuple come nell'esempio seguente.

INTERROGAZIONE 24. Si calcoli il numero dei valori distinti degli stipendi presenti nella base di dati.

```
I24:   SELECT COUNT (DISTINCT STIPENDIO)
     FROM   IMPiegato;
```

Si noti che, se si scrive COUNT(STIPENDIO) invece di COUNT(DISTINCT STIPENDIO) in I24, si ottiene lo stesso risultato di COUNT(*) perché i valori duplicati non vengono eliminati e quindi il numero dei valori risulta lo stesso del numero delle tuple.¹¹ Gli esempi precedenti mostrano come vanno applicate le funzioni per recuperare un valore di sintesi dalla base di dati. In alcuni casi può essere necessario usare le funzioni per selezionare particolari tuple. Si specifica allora un'interrogazione nidificata che applica la funzione desiderata e la si utilizza nella clausola WHERE di un'interrogazione esterna. Ad esempio, per recuperare i nomi di tutti gli impiegati che hanno due o più sottoposti si può scrivere:

```
SELECT COGNOME, NOME_BATT
  FROM IMPiegato
 WHERE (SELECT COUNT (*)
        FROM PERSONA_A_CARICO
       WHERE SSN=SSN_I) >= 2;
```

L'interrogazione nidificata correlata calcola il numero di persone a carico che ha ciascun dipendente; se è maggiore o uguale a 2, è selezionata la tupla del dipendente.

In molti casi si vogliono applicare le funzioni di aggregazione per creare *sottogruppi di tuple in una relazione*, sulla base dei valori di un attributo. Ad esempio, si può voler trovare lo stipendio medio dei dipendenti di ciascun dipartimento o il numero di dipendenti che lavorano su ogni progetto. In questi casi è necessario raggruppare le tuple che hanno lo stesso valore di uno o più attributi chiamati **attributi di raggruppamento** e applicare la funzione a ognuno di questi gruppi indipendentemente. SQL ha, per questo scopo, una clausola **GROUP BY** la quale specifica gli attributi di raggruppamento, che devono *apparire anche nella clausola SELECT*, in modo che il valore risultante dall'applicazione di una funzione a un gruppo di tuple compaia insieme al valore dell'attributo o degli attributi di raggruppamento.

¹¹ A meno che alcune tuple abbiano il valore NULL per l'attributo STIPENDIO, nel qual caso non sono conteggiate.

sia per il programmatore sia per il sistema generalmente è preferibile scrivere un'interrogazione evitando il più possibile la nidificazione e l'ordinamento conseguente.

Lo svantaggio dell'esistenza di molti modi per specificare la stessa interrogazione è che può causare confusione nell'utente, il quale può non sapere quale tecnica usare per particolari tipi di interrogazione. Un altro problema è che può essere più efficace eseguire un'interrogazione specificata in un modo piuttosto che in un metodo alternativo. In teoria questo non si dovrebbe verificare: il DBMS dovrebbe elaborare la stessa interrogazione nel medesimo modo, indipendentemente da come è specificata l'interrogazione. Questo è piuttosto difficile in pratica, poiché ogni DBMS usa metodi differenti per eseguire le interrogazioni specificate in modi diversi. Perciò un'ulteriore responsabilità dell'utente è stabilire qual è la più efficace tra varie specificazioni alternative della stessa interrogazione. Idealmente l'utente dovrebbe preoccuparsi solo di indicare l'interrogazione correttamente e dovrebbe essere responsabilità del DBMS eseguire l'interrogazione in modo efficace. In pratica, tuttavia, è meglio se l'utente sa quali tipi di costrutti in un'interrogazione sono più costosi da eseguire rispetto ad altri.

8.4 Le istruzioni Insert, Delete e Update in SQL

Per modificare la base di dati si possono usare tre comandi in SQL – INSERT, DELETE e UPDATE – qui di seguito presentati.

8.4.1 Il comando INSERT

Nella sua forma più semplice INSERT viene usato per aggiungere una singola tupla a una relazione. Si devono specificare il nome della relazione e un elenco di valori per la tupla. I valori dovrebbero essere elencati *nello stesso ordine* in cui gli attributi corrispondenti sono stati specificati dal comando CREATE TABLE. Ad esempio, per aggiungere una nuova tupla alla relazione IMPIEGATO mostrata in Figura 7.5 e specificata nel comando CREATE TABLE IMPIEGATO... in Figura 8.1 si può usare l'Istruzione U1:

```
U1:   INSERT INTO  IMPIEGATO
      VALUES      ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                    Oak Forest,Katy,TX','M', 37000, '987654321', 4);
```

Una seconda forma dell'istruzione INSERT consente all'utente di specificare esplicitamente i nomi degli attributi che corrispondono ai valori forniti nel comando INSERT. Ciò è utile se una relazione ha molti attributi, ma solo ad alcuni di questi vanno assegnati i valori nella nuova tupla. Questi attributi devono includere tutti quelli con la specificazione NOT NULL e nessun valore predefinito; gli attributi che consentono il NULL o hanno valori DEFAULT possono essere *tralasciati*. Ad esempio, per inserire una tupla per un nuovo impiegato di cui si conoscono solo gli attributi NOME_BATT, COGNOME, N_D e SSN si può usare U1A:

```
U1A:  INSERT INTO  IMPIEGATO (NOME_BATT, COGNOME, N_D, SSN)
      VALUES      ('Richard', 'Marini', 4, '653298653');
```

Gli attributi non specificati in U1A sono impostati ai loro valori DEFAULT o a NULL e i valori vengono elencati nello stesso ordine in cui gli *attributi sono elencati nel comando INSERT stesso*. È possibile anche inserire in una relazione *tuple multiple*, separate da virgolette, usando un singolo comando INSERT. I valori degli attributi che formano *ciascuna tupla* vanno posti tra parentesi.

Un DBMS che implementa completamente SQL2 dovrebbe supportare e imporre tutti i tipi di vincoli di integrità che possono essere specificati nel DDL. Tuttavia, in alcuni DBMS ciò non avviene a causa della perdita di efficienza che può derivare dalla complessità del far rispettare tutti i vincoli. Se un sistema non supporta alcuni vincoli, ad esempio quello dell'integrità referenziale, sono gli utenti o i programmati a doverli far rispettare. Ad esempio, se si immette il comando U2 nella base di dati mostrata in Figura 7.6, un DBMS che non supporta l'integrità referenziale eseguirà l'inserimento anche se non esiste alcuna tupla DIPARTIMENTO nella base di dati con NUMERO_D = 2. Spetta all'utente verificare che non sia violato alcun vincolo *il cui controllo non è implementato dal DBMS*. Tuttavia, il DBMS deve implementare i controlli per far rispettare tutti i vincoli d'integrità di SQL che *supporta*. Un DBMS che è in grado di far rispettare il vincolo NOT NULL rifiuterà un comando INSERT in cui un attributo dichiarato come NOT NULL non ha un valore; ad esempio U2A dovrebbe essere *rifiutato* perché non è fornito alcun valore di SSN.

```
U2:   INSERT INTO  IMPIEGATO (NOME_BATT, COGNOME, SSN, N_D)
      VALUES      ('Robert', 'Hatcher', '980760540', 2);
(* U2 è rifiutato se il controllo dell'integrità referenziale è fornito da DBMS *)
U2A:  INSERT INTO  IMPIEGATO (NOME_BATT, COGNOME, N_D)
      VALUES      ('Robert', 'Hatcher', 5);
(* U2A è rifiutato se il controllo di NOT NULL è fornito da DBMS *)
```

Una variante del comando INSERT inserisce tuple multiple in una relazione in corrispondenza della creazione della relazione, usando per l'inserimento il *risultato di un'interrogazione* ad altre relazioni. Ad esempio, per creare una tabella temporanea che contiene il nome, il numero degli impiegati e gli stipendi totali per ciascun dipartimento, si possono scrivere le istruzioni U3A e U3B:

```
U3A:  CREATE TABLE  INFO_DIP
      (NOME_DIP    VARCHAR(15),
       NUM_IMP     INTEGER,
       TOTAL_SAL   INTEGER);
U3B:  INSERT INTO  INFO_DIP (NOME_DIP, NUMERO_IMPIEGATI,
                           STIPENDIO_TOTALE)
      SELECT COUNT (*), SUM (STIPENDIO)
      FROM (DIPARTIMENTO JOIN IMPIEGATO ON
             NUMERO_D=N_D)
      GROUP BY NOME_D;
```

Una tabella INFO_DIP è creata da U3A ed è caricata con le informazioni di sintesi recuperate dalla base di dati dall'interrogazione in U3B. Ora è possibile interrogare INFO_DIP come si farebbe con qualsiasi altra relazione, e quando non serve più, si può eliminarla usando il comando DROP TABLE. Si noti che la tabella INFO_DIP può contenere dati non aggiornati; cioè se si aggiornano la relazione DIPARTIMENTO o la relazione IMPIEGATO dopo l'immersione di U3B, le informazioni in INFO_DIP non sono più aggiornate. Se si desidera l'aggiornamento automatico si deve creare una vista (si veda il Paragrafo 8.5).

8.4.2 Il comando DELETE

Il comando DELETE elimina le tuple da una relazione. Include una clausola WHERE simile a quella usata nelle interrogazioni SQL, per selezionare le tuple da cancellare. Le tuple sono rimosse esplicitamente da una tabella per volta. Tuttavia, l'eliminazione può propagarsi a tuple di altre relazioni se vi sono azioni specificate da vincoli d'integrità referenziale del DDL che vengono innescate (si veda il Sottoparagrafo 8.1.2). A seconda del numero di tuple selezionate dalla condizione nella clausola WHERE, nessuna, una o più tuple possono venir cancellate da un singolo comando DELETE. Se la clausola WHERE non è presente, tutte le tuple nella relazione vengono eliminate; tuttavia la tabella rimane nella base di dati come tabella vuota.¹² I comandi DELETE in U4A e fino a U4D, se applicati indipendentemente alla base di dati di Figura 7.6, cancelleranno rispettivamente nessuna, una, quattro e tutte le tuple dalla relazione IMPIEGATO:

```

U4A: DELETE FROM IMPIEGATO
      WHERE COGNOME='Brown';
U4B: DELETE FROM IMPIEGATO
      WHERE SSN='123456789';
U4C: DELETE FROM IMPIEGATO
      WHERE N_D IN (SELECT NUMERO_D
                    FROM DIPARTIMENTO
                   WHERE NOME_D='Ricerca');
U4D: DELETE FROM IMPIEGATO;
    
```

8.4.3 Il comando UPDATE

Il comando UPDATE è usato per modificare i valori degli attributi di una o più tuple selezionate. Come nel comando DELETE, la clausola WHERE del comando UPDATE seleziona le tuple da modificare in una singola relazione. Tuttavia, l'aggiornamento del valore di una chiave primaria può propagarsi ai valori della chiave esterna di tuple di altre relazioni se vi sono

¹² Si deve usare il comando DROP TABLE per eliminare completamente la tabella.

azioni referenziali che vengono innescate perché specificate in vincoli d'integrità referenziale del DDL (si veda il Sottoparagrafo 8.1.2). Un'ulteriore clausola SET specifica gli attributi da modificare e i loro nuovi valori. Ad esempio, per cambiare la posizione e il numero del dipartimento che controlla il progetto numero 10 rispettivamente in 'Bellaire' e 5, si usa U5:

```

U5: UPDATE PROGETTO
      SET SEDE_P = 'Bellaire', NUM_D = 5
      WHERE NUMERO_P=10;
    
```

Più tuple possono essere modificate con un unico comando UPDATE. Un esempio consiste nella modifica necessaria per dare a tutti gli impiegati nel dipartimento 'Ricerca' un aumento di stipendio del 10 per cento, come mostrato in U6. In questa richiesta il valore modificato STIPENDIO dipende dal valore originale STIPENDIO in ogni tupla, quindi sono necessari due riferimenti all'attributo STIPENDIO. Nella clausola SET, il riferimento all'attributo STIPENDIO a destra si riferisce al vecchio valore di STIPENDIO prima della modifica e quello a sinistra si riferisce al nuovo valore di STIPENDIO dopo la modifica:

```

U6: UPDATE IMPIEGATO
      SET STIPENDIO = STIPENDIO *1.1
      WHERE N_D IN (SELECT NUMERO_D
                    FROM DIPARTIMENTO
                   WHERE NOME_D='Ricerca');
    
```

È possibile anche specificare NULL o DEFAULT come nuovo valore di un attributo. Si noti che ogni comando UPDATE specifica esplicitamente le modifiche a una singola relazione. Per modificare più relazioni si devono immettere più comandi UPDATE. Questi (e altri comandi SQL) potrebbero essere incorporati in un programma di uso generale, come si vedrà nel Capitolo 12.

8.5 Viste (tabelle virtuali) in SQL

Verrà qui introdotto il concetto di vista SQL, illustrando come viene specificata, aggiornata e implementata una vista dal DBMS.

8.5.1 Il concetto di vista

Una vista nella terminologia di SQL è una singola tabella che deriva da altre tabelle,¹³ che possono essere tabelle base o altre viste definite precedentemente. Una vista non esiste necessa-

¹³ Come è usato qui, il termine *vista* è più limitato rispetto all'espressione *viste utente* illustrata nei Capitoli 1 e 2, poiché una vista utente può includere più relazioni.

riamente in forma fisica; è considerata una **tabella virtuale**, in contrapposizione alle tabelle base le cui tuple sono effettivamente memorizzate nella base di dati. Ciò limita le operazioni di aggiornamento possibili che possono essere applicate ad essa, ma non pone alcuna limitazione alla sua interrogazione.

Si può considerare una vista come un modo per specificare una tabella che si deve utilizzare di frequente anche se può non esistere fisicamente. Ad esempio, in Figura 7.5 si possono eseguire facilmente delle interrogazioni che recuperano il nome dell'impiegato e i nomi dei progetti in cui egli lavora. Piuttosto di dover specificare il join delle tabelle IMPiegato, Lavora_su e Progetto ogni volta che si esegue l'interrogazione, si può definire una vista che è un risultato di questi join. È possibile così effettuare le interrogazioni direttamente sulla vista, specificandole come interrogazioni su una singola tabella invece di interrogazioni che coinvolgono due join su tre tabelle. Le tabelle IMPiegato, Lavora_su e Progetto sono chiamate **tabelle di definizione** della vista.

8.5.2 Specificazione delle viste

Il comando per specificare un vista è **CREATE VIEW**. Alla vista è assegnato un nome di una tabella (virtuale) (o nome di vista), un elenco di nomi di attributi e un'interrogazione per specificare i contenuti della vista. Se nessuno degli attributi della vista è il risultato dell'applicazione di funzioni o di operazioni aritmetiche, non è necessario specificare i nomi degli attributi, perché risultano uguali ai nomi degli attributi corrispondenti nelle tabelle di definizione. Le viste in V1 e V2 creano delle tabelle virtuali i cui schemi sono illustrati in Figura 8.5 quando sono applicate allo schema della base di dati di Figura 7.5.

```
V1: CREATE VIEW      LAVORA_SU1
     AS   SELECT    NOME_BATT, COGNOME, NOME_P, ORE
           FROM    IMPiegato, PROGETTO, Lavora_SU
           WHERE   SSN=SSN_I AND N_P=NUMERO_P;
                    INFO_DIP(NOME_DIP, NUM_IMP,
                    STIP_TOTALE)
V2: CREATE VIEW
     AS   SELECT    NOME_D, COUNT (*), SUM (STIPENDIO)
           FROM    DIPARTIMENTO, IMPiegato
           WHERE   NUMERO_D=N_D
           GROUP BY NOME_D;
```

WORKS_ON			
NOME_BATT	COGNOME	NOME_P	ORE

DEPT_INFO		
NOME_DIP	NUM_IMP	STIP_TOTALE

Figura 8.5 Due viste specificate sullo schema della base di dati di Figura 7.5.

In V1 non è stato specificato nessun nuovo nome di attributi per la vista Lavora_su1 (anche se sarebbe possibile farlo); Lavora_su1 eredita i nomi degli attributi di vista dalle tabelle di definizione IMPiegato, PROGETTO e Lavora_su. La vista V2 specifica esplicitamente i nuovi nomi degli attributi della vista INFO_DIP, usando una corrispondenza uno a uno tra gli attributi specificati nella clausola CREATE VIEW e quelli specificati nella clausola SELECT dell'interrogazione che definisce la vista. Ora si possono specificare interrogazioni SQL su una vista, o tabella virtuale, esattamente nello stesso modo in cui si specificano le interrogazioni che coinvolgono le tabelle base.

Ad esempio, per recuperare il cognome e il nome di tutti gli impiegati che lavorano su 'ProgettoX', si può utilizzare la vista Lavora_su1 e specificare l'interrogazione come in QV1:

```
QV1: SELECT NOME_BATT, COGNOME
      FROM  LAVORA_SU1
      WHERE NOME_P='ProgettoX';
```

La stessa interrogazione se specificata sulle relazioni base richiederebbe la specificazione di due join; uno dei vantaggi principali di una vista è semplificare la specificazione di determinate interrogazioni. Le viste sono spesso usate anche come un meccanismo di protezione e autorizzazione.

Una vista è *sempre aggiornata*; se si modificano le tuple nelle tabelle base su cui è definita la vista, automaticamente la vista riflette questi cambiamenti. La vista, quindi, non è realizzata al momento della *definizione*, ma piuttosto quando si *specifica un'interrogazione* su di essa. È responsabilità del DBMS e non dell'utente assicurarsi che essa sia aggiornata.

Se una vista non serve più, si può usare il comando **DROP VIEW** per eliminarla. Ad esempio per eliminare la vista V1 si può utilizzare l'istruzione di SQL in V1A:

```
V1A: DROP VIEW  LAVORA_SU1;
```

8.5.3 Implementazione e aggiornamento delle viste

Il problema dell'implementazione efficace di una vista per l'interrogazione è complesso. Sono stati suggeriti due approcci principali. Una strategia, chiamata **modifica al momento dell'interrogazione**, richiede la modifica dell'interrogazione della vista in un'interrogazione sulle tabelle base sottostanti. Lo svantaggio di questo approccio è che non è efficace per le viste definite tramite interrogazioni complesse, che richiedono molto tempo di esecuzione, soprattutto se sono effettuate interrogazioni multiple alla vista in un breve periodo di tempo. L'altra strategia, detta **materializzazione della vista**, richiede la creazione fisica di una tabella temporanea della vista quando viene eseguita per la prima volta un'interrogazione su di essa e il mantenimento di tale tabella presupponendo che seguiranno altre interrogazioni. In questo secondo caso, per mantenere la tabella della vista aggiornata deve essere sviluppata una strategia efficiente che esegue il suo aggiornamento automatico quando sono aggiornate le tabelle di base. Per questo motivo sono state sviluppate delle tecniche che usano il concetto di **ag-**

giornamento incrementale, in cui si determina quali nuove tuple devono essere inserite, cancellate o modificate in una vista materializzata, quando avviene un cambiamento in una delle tabelle base. La vista in genere viene mantenuta purché su di essa si continuino a eseguire interrogazioni; se per un certo periodo di tempo non viene eseguita alcuna interrogazione, il sistema può rimuovere automaticamente la tabella fisica e ricalcolarla da zero quando successive interrogazioni fanno riferimento alla vista.

L'aggiornamento delle viste è complicato e può essere ambiguo. In generale un aggiornamento su una vista definita su una *singola tabella*, senza nessuna *funzione di aggregazione* può essere tradotto in un aggiornamento sulla tabella base sottostante. Per una vista che richiede dei join, un'operazione di aggiornamento può essere tradotta in operazioni di aggiornamento sulle relazioni base sottostanti in *vari modi*. Per illustrare i problemi potenziali che nascono dall'aggiornamento di una vista definita su più tabelle, si prenda in considerazione la vista LA-UV1: dall'aggiornamento di una vista definita su più tabelle, si prenda in considerazione la vista LA-UV1: si supponga di immettere il comando per aggiornare l'attributo NOME_P di 'John Smith' da 'ProdottoX' a 'ProdottoY'. Questo aggiornamento della vista è mostrato in UV1:

```
UV1: UPDATE LAVORA_SU1
      SET NOME_P = 'ProdottoY'
      WHERE COGNOME='Smith' AND NOME_BATT='John' AND
            NOME_P='ProdottoX';
```

Questa interrogazione può richiedere molti aggiornamenti delle relazioni base per ottenere l'effetto di aggiornamento desiderato sulla vista. Due possibili aggiornamenti (a) e (b) delle relazioni base che danno l'effetto desiderato su UV1 sono illustrati qui di seguito.

```
(a): UPDATE LAVORA_SU
      SET N_P = (SELECT NUMERO_P FROM PROGETTO
                  WHERE NOME_P='ProdottoY')
      WHERE SSN_I IN (SELECT SSN FROM IMPIEGATO
                      WHERE COGNOME='Smith' AND NOME_BATT='John')
            AND
            N_P IN (SELECT NUMERO_P FROM PROGETTO
                      WHERE NOME_P='ProdottoX');

(b): UPDATE PROGETTO
      SET NOME_P = 'ProdottoY'
      WHERE NOME_P = 'ProdottoX';
```

L'aggiornamento (a) collega 'John Smith' alla tupla PROGETTO di 'ProdottoY' al posto della tupla PROGETTO di 'ProdottoX' è la tecnica di aggiornamento migliore. Tuttavia, anche (b) farebbe raggiungere l'effetto desiderato sulla vista, anche se lo fa cambiando il nome della tupla 'ProdottoX' nella relazione PROGETTO in 'ProdottoY'. È improbabile che l'utente che ha specificato l'aggiornamento della vista UV1 desideri che l'aggiornamento sia interpretato come in (b), poiché esso ha l'effetto di cambiare tutte le tuple della vista con NOME_P = 'ProdottoX'.

Alcuni aggiornamenti della vista possono persino non avere molto senso; ad esempio modificare l'attributo STIP_TOTALE della vista INFO_DIP non ha senso perché STIP_TOTALE è definito come la somma dei stipendi personali degli impiegati. Questa richiesta è mostrata in UV2:

```
UV2: UPDATE INFO_DIP
      SET STIP_TOTALE=100000
      WHERE NOME_D='Ricerca';
```

Un numero elevato di aggiornamenti alle relazioni base sottostanti possono soddisfare tale aggiornamento della vista. L'aggiornamento di una vista è fattibile solamente quando *un solo possibile aggiornamento* delle relazioni base corrisponde all'effetto desiderato di aggiornamento sulla vista. Ogni volta che un aggiornamento della vista può essere eseguito attraverso *più di un aggiornamento* nelle relazioni base sottostanti, è necessario avere una procedura sicura per scegliere una delle possibilità. Alcuni ricercatori hanno sviluppato dei metodi per scegliere l'aggiornamento che è più probabile, mentre altri vogliono che sia l'utente a scegliere quello desiderato durante la definizione della vista.

Riepilogando si possono fare le seguenti osservazioni:

- una vista definita su una singola tabella è aggiornabile se i suoi attributi contengono la chiave primaria (o qualche altra chiave candidata) della relazione base, perché ciò collega ciascuna tupla (virtuale) della vista a una singola tupla base;
- le viste definite su tabelle multiple usando i join, in genere non sono aggiornabili;
- le viste definite usando le funzioni di raggruppamento e di aggregazione non sono aggiornabili.

In SQL2 la clausola WITH CHECK OPTION deve essere aggiunta alla fine della definizione della vista se questa deve essere aggiornata. Ciò consente al sistema di controllarne l'aggiornabilità e di progettare una strategia di esecuzione per i suoi aggiornamenti.

8.6 Specificazione di vincoli generali come asserzioni

In SQL2 gli utenti possono specificare vincoli più generali – che non rientrano in alcuna delle categorie descritte nel Sottoparagrafo 8.1.2 – attraverso **asserzioni dichiarative**, usando l'**istruzione del DDL CREATE ASSERTION**. A ogni asserzione viene assegnato un nome di vincolo, specificato attraverso una condizione simile alla clausola WHERE di un'interrogazione SQL. Ad esempio, per specificare il vincolo che "lo stipendio di un impiegato non deve essere superiore a quello di un dirigente del dipartimento per cui egli lavora" in SQL2, è possibile scrivere la seguente asserzione:

```
CREATE ASSERTION VINCOLO_STIPENDIO
CHECK (NOT EXISTS (SELECT * FROM IMPIEGATO I, IMPIEGATO M,
                    DIPARTIMENTO D
                   WHERE I.STIPENDIO>M.STIPENDIO AND
                         I.N_D=D.NUMERO_D AND
                         D.SSN_DIR=M.SSN));
```


- mento invece che tutti i dipendenti (come nell'Esercizio 8.14a). Si può specificare questa interrogazione in SQL? Perché? o perché no?
- 8.15 Si specifichino gli aggiornamenti dell'Esercizio 7.19, usando i comandi di aggiornamento di SQL.
- 8.16 Si specifichino le seguenti in interrogazioni SQL sullo schema della base di dati di Figura 1.2.
- Si recuperino i nomi di tutti gli studenti dell'ultimo anno che si specializzano in CS (Computer Science, Informatica).
 - Si recuperino i nomi di tutti i corsi tenuti dal professor King dal 1998 al 1999.
 - Per ciascun modulo di corso tenuto dal professor King, si recuperi il numero del corso, il semestre, l'anno e il numero di studenti che scelgono il modulo.
 - Si recuperi il nome e la trascrizione del libretto universitario di ogni studente dell'ultimo anno (AnnoCorso = 5) che si specializza in CS. Una trascrizione include il nome dell'insegnamento, il codice dell'insegnamento, le ore di frequenza, il semestre, l'anno e il voto per ciascun insegnamento completato dallo studente.
 - Si recuperino i nomi e i dipartimenti di specializzazione degli studenti più meritevoli (studenti che hanno conseguito il massimo dei voti in tutti i loro insegnamenti).
 - Si recuperino i nomi e i dipartimenti di specializzazione di tutti gli studenti che non hanno il massimo dei voti in tutti gli insegnamenti da loro frequentati.
- 8.17 Si scrivano le istruzioni SQL di aggiornamento per eseguire le seguenti operazioni sullo schema della base di dati mostrato in Figura 1.2.
- Si inserisca un nuovo studente <'Johnson', 25, 1, 'MATH'> nella base di dati.
 - Si cambi la classe dello studente 'Smith' in 2.
 - Si inserisca un nuovo corso <'Ingegneria della conoscenza', 'CS4390', 3, 'CS'>.
 - Si cancelli il record dello studente il cui nome è 'Smith' e il cui numero è 17.
- 8.18 Si specifichino le seguenti viste in SQL sullo schema della base di dati AZIENDA mostrato in Figura 7.5.
- Una vista che contiene il nome del dipartimento, il nome del dirigente e lo stipendio del dirigente di ogni dipartimento.
 - Una vista che contiene il nome dell'impiegato, il nome del supervisore e lo stipendio dell'impiegato per ogni impiegato che lavora nel dipartimento 'Ricerca'.
 - Una vista che contiene il nome del progetto, il nome del dipartimento che controlla il progetto, il numero degli impiegati e le ore totali lavorate per settimana sul progetto per ciascun progetto.
 - Una vista che contiene il nome del progetto, il nome del dipartimento che lo controlla, il numero degli impiegati e le ore totali lavorate per settimana sul progetto per ciascun progetto in cui lavora *più di un impiegato*.
- 8.19 Si prenda in considerazione la seguente vista SINTESI_DIP definita sulla base di dati AZIENDA di Figura 7.6:

```
CREATE VIEW SINTESI_DIP (D, C, S_TOTALE, S_MEDIO)
AS SELECT N_D, COUNT (*), SUM (STIPENDIO), AVG (STIPENDIO)
AS FROM IMPIEGATO
AS GROUP BY N_D;
```

Si specifichi quali delle seguenti interrogazioni e aggiornamenti sono permessi sulla vista. Se sono consentiti un'interrogazione o un aggiornamento, si mostri come sarebbero le corrispondenti interrogazioni o aggiornamenti sulle relazioni di base e si fornisca il risultato quando sono applicati alla base di dati di Figura 7.6.

- SELECT** *
FROM SINTESI_DIP;
 - SELECT** D, C
FROM SINTESI_DIP
WHERE S_TOTALE > 100000;
 - SELECT** D, S_MEDIO
FROM SINTESI_DIP
WHERE C > (SELECT C FROM SINTESI_DIP WHERE D=4);
 - UPDATE** SINTESI_DIP
SET D=3
WHERE D=4;
 - DELETE FROM** SINTESI_DIP
WHERE C > 4;
- 8.20 Si prenda in considerazione lo schema della relazione CONTAINS (Parent_part#, Sub_part#); il fatto che una tupla $\langle P_i, P_j \rangle$ si trovi in CONTAINS significa che la parte P_i contiene la parte P_j come componente diretta. Si supponga di scegliere una parte P_k che non include altre parti e di voler trovare i numeri delle parti che di tutte le parti contenute P_k direttamente o indirettamente, a un qualsiasi livello; questa è un'*interrogazione ricorsiva* che richiede il calcolo della *chiusura transitiva* di CONTAINS. Si mostri che questa interrogazione non può essere specificata direttamente con una singola interrogazione SQL. Si possono suggerire delle estensioni a SQL per consentire la specificazione di tali interrogazioni?
- 8.21 Se specifichino le interrogazioni e gli aggiornamenti degli Esercizi 7.20 e 7.21 che si riferiscono alla base di dati COMPAGNIA_AEREA.
- 8.22 Si scelga un'applicazione di base di dati con cui si ha familiarità.
- Si progetti uno schema relazionale per la propria applicazione di basi di dati.
 - Si dichiarino le proprie relazioni usando il DDL di SQL.
 - Si specifichino alcune interrogazioni in SQL necessarie per la propria applicazione di base di dati.
 - In base all'utilizzo preventivo per la base di dati, si scelgano alcuni attributi che dovrebbero avere degli indici specificati su di loro.
 - Si implementi la base di dati se si ha un DBMS che supporta SQL.
- 8.23 Si specifichino le risposte degli Esercizi da 7.24 fino a 7.28 in SQL.

Bibliografia selezionata

Il linguaggio SQL, chiamato originariamente SEQUEL, si basava sul linguaggio SQUARE (Specifying Queries as Relational Expressions, Specificazione di Interrogazioni come Espres-

struendo la relazione LAVORA_SU di Figura 7.5.² Si inseriscono le chiavi primarie delle relazioni PROGETTO e IMPIEGATO come chiavi esterne di LAVORA_SU e le si ridenomina rispettivamente N_P e SSN_I. Si inserisce anche un attributo ORE in LAVORA_SU, per rappresentare l'attributo Ore del tipo di associazione. La chiave primaria della relazione LAVORA_SU è data dalla combinazione degli attributi di chiave esterna {SSN_I, N_P}.

Sulle chiavi esterne della relazione corrispondente all'associazione R deve essere specificata l'opzione di propagazione (CASCADE) per l'azione referenziale innescata (si veda il Paragrafo 8.1), dal momento che ogni istanza di associazione presenta una dipendenza di esistenza da ognuna delle entità che collega. Ciò può essere usato sia per ON UPDATE sia per ON DELETE.

Si noti che è sempre possibile tradurre associazioni 1:1 o 1:N in modo analogo a quanto fatto per le associazioni M:N. Questa possibilità è particolarmente utile quando esistono poche istanze di associazione, per evitare la presenza di valori nulli nelle chiavi esterne. In questo caso la chiave primaria della relazione "associazione" sarà *solo una* delle chiavi esterne che riferiscono le relazioni "entità" partecipanti. Per un'associazione 1:N questa sarà la chiave esterna che riferisce la relazione entità al lato-N. Per un'associazione 1:1 è scelta come chiave primaria la chiave esterna che riferisce la relazione entità con partecipazione totale (se esiste).

PASSO 6. Per ogni attributo multivale A, si costruisca una nuova relazione R. Questa relazione R comprenderà un attributo corrispondente ad A, più l'attributo di chiave primaria K – come chiave esterna di R – della relazione che rappresenta il tipo di entità o il tipo di associazione che ha A come attributo. La chiave primaria di R è data dalla combinazione di A e K. Se l'attributo multivale è composto, si considerano le sue componenti semplici.³

Nel nostro esempio, si costruisce una relazione SEDI_DIP. L'attributo SEDE_D rappresenta l'attributo multivale Sedi di DIPARTIMENTO, mentre NUMERO_D – come chiave esterna – rappresenta la chiave primaria della relazione DIPARTIMENTO. La chiave primaria di SEDI_DIP è data dalla combinazione di {NUMERO_D, SEDE_D}. In SEDI_DIP esisterà una tupla separata per ogni sede di un dipartimento.

Per l'azione referenziale innescata (si veda il Paragrafo 8.1) dovrebbe essere specificata l'opzione di propagazione (CASCADE) sulla chiave esterna della relazione corrispondente all'attributo multivale, sia per ON UPDATE sia per ON DELETE.

In Figura 7.5 (e Figura 7.7) è mostrato lo schema di base di dati relazionale ottenuto seguendo i passi precedenti, mentre in Figura 7.6 è rappresentato uno stato esemplificativo della base di dati. Si noti che non è stata finora trattata la traduzione di tipi di associazione n -ari ($n > 2$), perché in Figura 3.2 non ce ne sono; essi vengono tradotti in modo simile ai tipi di associazione M:N inserendo il seguente passo aggiuntivo nell'algoritmo di traduzione.

PASSO 7. Per ogni tipo di associazione n -ario R , dove $n > 2$, si costruisca una nuova relazione S per rappresentare R. Si inseriscano in S come attributi di chiave esterna le chiavi primarie delle relazioni che rappresentano i tipi di entità partecipanti. Si inseriscano anche come attributi di S tutti gli attributi semplici del tipo di associazione n -ario (o componenti semplici degli attributi composti). La chiave primaria di S è di solito una combinazione di tutte le chiavi esterne che riferiscono le relazioni rappresentanti i tipi di entità partecipanti. Però, se uno qualsiasi dei tipi di entità partecipanti E in R ha 1 come vincolo di cardinalità, allora la chiave primaria di S non deve comprendere l'attributo di chiave esterna che riferisce la relazione E' corrispondente a E (si veda il Paragrafo 4.7). Con ciò si conclude la procedura di traduzione.

Ad esempio, si consideri il tipo di associazione FORNITURA di Figura 4.13(a). Esso può essere tradotto nella relazione FORNITURA mostrata in Figura 9.1, la cui chiave primaria è data dalla combinazione di chiavi primarie {NOME_F, NUM_PARTE, NOME_PROG}.

Il punto fondamentale da notare in uno schema relazionale è che, al contrario di quanto avviene in uno schema ER, i tipi di associazione non sono rappresentati esplicitamente; essi sono piuttosto rappresentati con l'uso di due attributi A e B, uno chiave primaria e l'altro chiave esterna – sullo stesso dominio – inseriti in due relazioni S e T. Due tuple di S e T sono collegate quando hanno lo stesso valore per A e B. Usando l'operazione di EQUIJOIN (o JOIN NATURALE) su S.A e T.B, è possibile combinare tutte le coppie di tuple collegate da S e T e dar corpo all'associazione. Quando è interessato un tipo di associazione binario 1:1 o 1:N, è di solito necessaria una sola operazione di join. Per un tipo di associazione M:N binario sono necessarie due operazioni di join, mentre per tipi di associazione n -ari sono necessari n join.

Ad esempio, per formare una relazione comprendente il nome dell'impiegato, il nome del progetto e le ore lavorate dall'impiegato su ogni progetto, occorre collegare ogni tupla IMPIEGATO alle relative tuple PROGETTO, tramite la relazione LAVORA_SU di Figura 7.5. Perciò si deve eseguire l'operazione di EQUIJOIN sulle relazioni IMPIEGATO e LAVORA_SU con la condizione di join SSN = SSN_I, e quindi eseguire un'altra operazione di EQUIJOIN sulla relazione risultante e sulla relazione PROGETTO con condizione di join N_P = NUMERO_P. In gene-

FORNITORE	
NOME_F	...
PROGETTO	
NOME_PROG	...
PARTE	
NUM_PARTE	...
FORNITURA	
NOME_F	NOME_PROG
	NUM_PARTE
	QUANTITÀ

Figura 9.1 Traduzione del tipo di associazione n -ario FORNITURA di Figura 4.13(a).

² Queste sono talora dette *relazioni associazione* perché ogni tupla (riga) corrisponde a un'istanza di associazione.

³ In alcuni casi quando un attributo multivale è composto, nella chiave di R sono richiesti solo alcuni degli attributi componenti; questi attributi svolgono funzione analoga a quella di una chiave parziale di un tipo di entità de-

rale, quando è necessario attraversare più associazioni, devono essere specificate numerose operazioni di join. L'utente di una base di dati relazionale deve essere sempre informato su quali sono gli attributi di chiave esterna, in modo da usarli correttamente nel combinare tuple collegate provenienti da due o più relazioni. Se viene eseguito un EQUIJOIN tra attributi di due relazioni che non rappresentano un'associazione chiave esterna/chiave primaria, il risultato può essere spesso privo di significato e può portare a dati spuri (non validi). Ad esempio, il lettore può provare a unire tramite join le relazioni PROGETTO e SEDI_DIP sotto la condizione SEDE_D = SEDE_P ed esaminare quindi il risultato (si veda anche il Capitolo 10).

Un altro aspetto degno di nota nello schema relazionale è che viene costruita una relazione separata per *ogni* attributo multivalore. Per una specifica entità con un insieme di valori per l'attributo multivalore, il valore dell'attributo chiave dell'entità è ripetuto in una tupla separata per ogni valore dell'attributo multivalore. Ciò perché il modello relazionale di base *non* consente valori multipli (una lista, o un insieme di valori) per un attributo di una tupla singola. Ad esempio, dato che il dipartimento 5 ha tre sedi, nella relazione SEDI_DIP di Figura 7.6 esistono tre tuple; ogni tupla specifica una delle sedi. Nel nostro esempio, si applica l'EQUIJOIN a SEDI_DIP e DIPARTIMENTO sulla base dell'attributo NUMERO_D per ottenere i valori di tutte le sedi insieme con altri attributi di DIPARTIMENTO. Nella relazione risultante, i valori degli altri attributi di dipartimento vengono ripetuti in tuple separate per ogni sede del dipartimento.

L'algebra relazionale di base non ha un'operazione NEST (NIDIFICA) o COMPRESS (COMPRIMI), la quale produrrebbe a partire dalla relazione SEDI_DIP di Figura 7.6 un insieme di tuple della forma {<1, Houston>, <4, Stafford>, <5, {Bellaire, Sugarland, Houston}>}. Questo è un serio inconveniente della versione di base normalizzata o "piatta" ("flat") del modello relazionale. Sotto questo aspetto, i modelli a oggetti, gerarchico e reticolare dispongono di funzionalità migliori rispetto a quelle del modello relazionale. Il modello relazionale nidificato cerca di porre rimedio a questo problema.

9.1.2 Sommario della traduzione per costrutti e vincoli del modello ER

In Tabella 9.1 vengono riassunte le corrispondenze tra costrutti e vincoli dei modelli ER e relazionale.

Tabella 9.1 Corrispondenza tra i modelli ER e relazionale.

Modello ER	Modello relazionale
Tipo di entità	Relazione "entità"
Tipo di associazione 1:1 o 1:N	Chiave esterna (o relazione "associazione")
Tipo di associazione M:N	Relazione "associazione" e due chiavi esterne
Tipo di associazione n-ario	Relazione "associazione" e n chiavi esterne
Attributo semplice	Attributo
Attributo composto	Insieme di attributi componenti semplici
Attributo multivalore	Relazione e chiave esterna
Insieme di valori	Dominio
Attributo chiave	Chiave primaria (o secondaria)

9.2 Traduzione dei concetti del modello EER in relazioni

9.2.1 Associazioni superclasse/sottoclasse e specializzazione (o generalizzazione)

Sono possibili molte opzioni per tradurre un certo numero di sottoclassi che formano nel complesso una specializzazione (o, in alternativa, che sono generalizzate in una superclasse), come ad esempio le sottoclassi {SEGRETARIO, TECNICO, INGEGNERE} di IMPIEGATO in Figura 4.4. È possibile aggiungere un passo ulteriore all'algoritmo di traduzione da ER a relazionale del Paragrafo 9.1, che ha sette passi, per gestire la traduzione della specializzazione. Il passo 8, che segue, fornisce le opzioni più comuni; sono anche possibili altre traduzioni. Successivamente saranno esaminate le condizioni in cui dovrebbe essere usata ciascuna opzione. Verrà qui usata la notazione Attr(*R*) per indicare gli attributi della relazione *R* e PK(*R*) per indicare la chiave primaria di *R*.

PASSO 8. Si trasformi ogni specializzazione con *m* sottoclassi {*S*₁, *S*₂, ..., *S_m*} e superclasse (generalizzata) *C*, dove gli attributi di *C* sono {*k*, *a₁*, ..., *a_n*} e *k* è la chiave (primaria), in schemi di relazione, usando una delle quattro opzioni seguenti.

Opzione 8A: si costruisca una relazione *L* per *C* con attributi Attr(*L*) = {*k*, *a₁*, ..., *a_n*} e PK(*L*) = *k*. Si costruisca una relazione *L_i* per ogni sottoclasse *S_i*, $1 \leq i \leq m$, con gli attributi Attr(*L_i*) = {*k*} \cup {attributi di *S_i*} e PK(*L_i*) = *k*.

Opzione 8B: si costruisca una relazione *L_i* per ogni sottoclasse *S_i*, $1 \leq i \leq m$, con gli attributi Attr(*L_i*) = {attributi di *S_i*} \cup {*k*, *a₁*, ..., *a_n*} e PK(*L_i*) = *k*.

Opzione 8C: si costruisca una singola relazione *L* con attributi Attr(*L*) = {*k*, *a₁*, ..., *a_n*} \cup {attributi di *S₁*} \cup ... \cup {attributi di *S_m*} \cup {*t*} e PK(*L*) = *k*. Questa opzione va usata per una specializzazione le cui sottoclassi sono *disgiunte*, e *t* è un attributo di tipo (o discriminante) che indica la sottoclasse a cui appartiene ciascuna tupla, nel caso in cui ce ne sia una. Questa opzione può produrre un gran numero di valori nulli.

Opzione 8D: si costruisca un solo schema di relazione *L* con attributi Attr(*L*) = {*k*, *a₁*, ..., *a_n*} \cup {attributi di *S₁*} \cup ... \cup {attributi di *S_m*} \cup {*t₁*, *t₂*, ..., *t_m*} e PK(*L*) = *k*. Questa opzione è pensata per una specializzazione le cui sottoclassi sono *sovraposte* (ma funzionerà anche per una specializzazione disgiunta), e ogni *t_i*, $1 \leq i \leq m$, è un attributo booleano che indica se una tupla appartiene alla sottoclasse *S_i*.

Le opzioni 8A e 8B possono essere denominate *opzioni di relazioni multiple*, mentre le opzioni 8C e 8D possono essere denominate *opzioni di relazioni singole*. L'opzione 8A dà luogo a una relazione *L* per la superclasse *C* e i suoi attributi, più una relazione *L_i* per ogni sottoclasse *S_i*; ogni *L_i* comprende gli attributi specifici (o locali) di *S_i* più la chiave primaria della superclasse *C*, che è propagata a *L_i* e diventa la sua chiave primaria. Un'operazione di EQUIJOIN sulla chiave primaria tra un *L_i* qualsiasi e *L* produce tutti gli attributi specifici ed ereditati delle entità in *S_i*. Questa opzione è illustrata in Figura 9.2(a) per lo schema EER di

(a)	IMPIEGATO
	SSN NomeBatt InizInt Cognome DataNascita Indirizzo TipoLavoro
	SEGRETARIO
	SSN VelocitàDattilografia
	TECNICO
	SSN GradoT
	INGEGNERE
	SSN Tipolog
(b)	AUTOMOBILE
	IdVeicolo NumTarga Prezzo VelocitàMax NumPasseggeri
	CAMION
	IdVeicolo NumTarga Prezzo NumeroAssi Tonnellaggio
(c)	IMPIEGATO
	SSN NomeBatt InizInt Cognome DataNascita Indirizzo TipoLavoro VelocitàDattilografia GradoT Tipolog
(d)	PARTITE
	NumParte Descrizione FlagF NumDisegno DataCostruzione NumLotto FlagA NomeFornitore PrezzoDiListino

Figura 9.2 Opzioni per tradurre specializzazioni (o generalizzazioni) in relazioni. (a) Traduzione dello schema EER di Figura 4.4 in relazioni usando l'Opzione A. (b) Traduzione dello schema EER di Figura 4.4 usan-
gura 4.3(b) in relazioni usando l'Opzione B. (c) Traduzione dello schema EER di Figura 4.4 usan-
gura l'Opzione C, con TipoLavoro che svolge il ruolo di attributo di tipo. (d) Traduzione dello sche-
ma EER di Figura 4.5 usando l'Opzione D, con due campi booleani di tipo, FlagF e FlagA.

Figura 4.4. L'opzione 8A funziona qualsiasi siano i vincoli della specializzazione: disgiunta o sovrapposta, totale o parziale. Si noti che per ogni L_i deve sussistere il vincolo

$$\pi_{\leftarrow\rightarrow}(L_i) \subseteq \pi_{\leftarrow\rightarrow}(L)$$

Ciò specifica la cosiddetta *dipendenza d'inclusione*, indicata con $L_i \ll L \cdot k$.

Nell'opzione 8B l'operazione di EQUIJOIN è *incorporata* nello schema, eliminando così la relazione L , come illustrato in Figura 9.2(b) per la specializzazione EER di Figura 4.3(b). Questa opzione va bene solo quando sussiste sia il vincolo di specializzazione disgiunta sia quello di specializzazione totale. Se la specializzazione non è totale, un'entità che non appartiene a nessuna delle sottoclassi S_i verrà persa. Se la specializzazione non è disgiunta, un'entità che appartiene a più di una sottoclasse avrà gli attributi ereditati dalla sottoclasse C memorizzati in modo ridondante in più di una L_i . Con l'opzione 8B nessuna relazione contiene tutte le entità presenti nella superclasse C ; di conseguenza dobbiamo eseguire un'operazione di UNIONE ESTERNA (o JOIN ESTERNO COMPLETO) sulle relazioni L_i per recuperare tutte le entità presenti in C . Il risultato dell'unione esterna sarà simile alle relazioni definite dalle opzioni 8C e 8D, tranne per il fatto che in questo caso mancheranno i campi di tipo. Ogni volta che si cerca un'entità arbitraria in C , occorre ispezionare tutte le

Le opzioni 8C e 8D danno luogo a una sola relazione per rappresentare la superclasse C e tutte le sue sottoclassi. Un'entità che non appartiene a qualcuna delle sottoclassi presenterà valori nulli per gli attributi specifici di queste sottoclassi. Perciò queste opzioni non sono consigliate se per le sottoclassi sono definiti molti attributi specifici. Però, se esistono pochi attributi specifici, queste traduzioni sono preferibili alle opzioni 8A e 8B perché eliminano la necessità di specificare operazioni di EQUIJOIN e di UNIONE ESTERNA, e perciò possono dar luogo a una implementazione più efficiente. L'opzione 8C è usata per gestire sottoclassi disgiunte inserendo un singolo attributo di tipo (o d'immagine o discriminante) t per indicare la sottoclasse a cui appartiene ciascuna tupla; perciò, il dominio di t potrebbe essere $\{1, 2, \dots, m\}$. Se la specializzazione è parziale, t può assumere valori nulli nelle tuple che non appartengono a nessuna sottoclasse. Se la specializzazione è definita tramite un attributo, quell'attributo serve lo scopo di t e t non è necessario; questa opzione è illustrata in Figura 9.2(c) per la specializzazione EER di Figura 4.4. L'opzione 8D è usata per gestire sottoclassi sovrapposte inserendo m campi booleani di tipo, uno per ogni sottoclasse. Essa può essere utilizzata anche per classi disgiunte. Ogni attributo t_i di tipo può avere un dominio {sì, no}, dove un valore sì indica che la tupla è un membro della sottoclasse S_i . Questa opzione è illustrata in Figura 9.2(d) per la specializzazione EER di Figura 4.5, dove FlagF e FlagA (flag = bandiera, indicatore) sono i campi di tipo. Si noti che è anche possibile costruire un solo tipo di m bit anziché gli m campi di tipo.

Quando si ha una gerarchia o un reticolto di specializzazioni (o generalizzazioni) su più livelli, non è necessario seguire la stessa opzione di traduzione per tutte le specializzazioni. Si può invece usare un'opzione di traduzione per parte della gerarchia o reticolto e altre opzioni per altre parti. In Figura 9.3 è mostrata una possibile traduzione in relazioni per il reticolto di Figura 4.7. Qui è stata usata l'opzione 8A per PERSONA/{IMPIEGATO, EX_ALLIEVO, STUDENTE}, l'opzione 8C per IMPIEGATO/{PERSONALE_DI_SUPPORTO, CORPO_DOCENTE, ASSISTENTE_STUDENTI}, e l'opzione 8D per ASSISTENTE_STUDENTI/{ASSISTENTE_DI_RICERCA, ASSISTENTE_DI_DIDATTICA}, STUDENTE/ASSISTENTE_STUDENTI (in STUDENTE), e STUDENTE/{STUDENTE_LAUREATO, STUDENTE_NON_LAUREATO}. In Figura 9.3 tutti gli attributi i cui nomi contengono 'Tipo' e 'Flag' sono campi di tipo.

PERSONA
SSN Nome DataNascita Sesso Indirizzo
IMPIEGATO
SSN Stipendio Tipolmpiegato Posizione Fascia TempoPercentuale FlagAR FlagAD Progetto Insegnamento
EX_ALLIEVO
TITOLI_EX_ALLIEVO
SSN Anno Titolo DisciplinaDiSpecializzazione
STUDENTE
SSN DipSpecializzazione FlagLaureato FlagNonLaureato ProgrammaDiStudi AnnoCorso FlagAssistStudi

Figura 9.3 Traduzione del reticolto di specializzazione EER di Figura 4.7 con l'uso di opzioni multiple.

dalla base di dati AZIENDA. In Tabella 9.1 sono riassunte le corrispondenze tra i costrutti e i vincoli dei modelli ER e relazionale. Abbiamo quindi illustrato passi aggiuntivi che ci consentono di tradurre i costrutti del modello EER nel modello relazionale.

Questionario di verifica

- 9.1. Si discutano le corrispondenze tra i costrutti del modello ER e quelli del modello relazionale. Si mostri come ogni costrutto del modello ER possa essere tradotto nel modello relazionale, e si studino tutte le traduzioni alternative. Si discutano le opzioni per la traduzione dei costrutti del modello EER.

Esercizi

- 9.2. Si provi a tradurre lo schema relazionale di Figura 7.20 in uno schema ER. Ciò fa parte di un processo noto come *reverse engineering* (ingegnerizzazione al contrario), in cui si costruisce uno schema concettuale per una base di dati implementata esistente. Si enunci esplicitamente ogni assunzione fatta.
- 9.3. In Figura 9.5 è mostrato uno schema ER per una base di dati che può essere usata per tener traccia delle navi da trasporto e delle loro posizioni, per le autorità marittime. Si traduca questo schema in uno schema relazionale, e si specifichino tutte le chiavi primarie nonché le chiavi esterne.
- 9.4. Si traduca lo schema ER BANCA dell'Esercizio 3.23 (mostrato in Figura 3.17) in uno schema relazionale. Si specifichino tutte le chiavi primarie e le chiavi esterne. Si ripeta la cosa per lo schema COMPAGNIA_AEREA (Figura 3.16) dell'Esercizio 3.19 e per gli altri schemi presenti negli Esercizi 3.16-3.24.

Capitolo 10

Dipendenze funzionali e normalizzazione per basi di dati relazionali

Nei Capitoli 7-9 abbiamo presentato vari aspetti del modello relazionale. Ogni *schemma di relazione* è composto da un certo numero di attributi, e lo *schemma di base di dati relazionale* è composto da un certo numero di schemi di relazione. Finora abbiamo supposto che gli attributi vengano raggruppati per formare uno schema di relazione usando il buon senso del progettista di basi di dati o traducendo uno schema specificato nel modello Entità-Associazione (ER) o ER-esteso (EER) (o altri modelli di dati concettuali simili) in uno schema relazionale. Il modello EER fa sì che il progettista individui i tipi di entità e i tipi di associazione e i loro rispettivi attributi, il che porta a un raggruppamento naturale e logico degli attributi in relazioni quando sono seguite le procedure di traduzione illustrate nei Paragrafi 9.1 e 9.2. Abbiamo però ancora bisogno di misurare formalmente il perché un raggruppamento di attributi in uno schema di relazione possa essere meglio di un altro. Finora, nella nostra discussione sulla progettazione concettuale nei Capitoli 3 e 4 e sulla sua traduzione nel modello relazionale nel Capitolo 9, non abbiamo sviluppato nessuna misura di adeguatezza, "bontà" o qualità del progetto, se non l'intuizione del progettista.

In questo capitolo esamineremo parte della teoria che è stata sviluppata nel tentativo di scegliere "buoni" schemi di relazione. Ci sono due livelli ai quali possiamo esaminare la loro "bontà". Il primo è il *livello logico* (o *concettuale*) – come gli utenti interpretano gli schemi di relazione e il significato dei loro attributi. L'avere buoni schemi di relazione a questo livello consente agli utenti di capire chiaramente il significato dei dati nelle relazioni, e perciò di formulare correttamente le loro interrogazioni. Il secondo è il *livello d'implementazione* (o *fisico*) – come le tuple in una relazione di base sono memorizzate e aggiornate. Questo livello si applica solo a schemi di relazione di base – che saranno memorizzati fisicamente come file – mentre al livello logico noi siamo interessati a schemi sia di relazioni di base sia di viste (relazioni virtuali). La teoria della progettazione di basi di dati relazionali sviluppata in questo capitolo si applica principalmente a *relazioni di base*, anche se alcuni criteri di adeguatezza si applicano anche alle viste, come verrà mostrato nel Paragrafo 10.1.

possia essere interpretata come un insieme di fatti o asserzioni. Questo significato, o **semantica**, specifica come interpretare i valori degli attributi memorizzati in una tupla della relazione – in altre parole, come i valori degli attributi in una tupla sono in relazione tra loro. Se è stata eseguita correttamente la progettazione concettuale, seguita da una traduzione nelle relazioni, la maggior parte della semantica dovrebbe essere spiegata e il progetto risultante dovrebbe avere un significato chiaro.

In generale il progetto di uno schema di relazione sarà tanto migliore quanto più facile è spiegare la semantica della relazione. Per illustrare ciò si considerino la Figura 10.1, una versione semplificata dello schema di base di dati relazionale AZIENDA di Figura 7.5, e la Figura 10.2, che presenta un esempio di relazioni popolate di questo schema. Il significato dello sche-

IMPIEGATO				
NOME_I	SSN	DATA_N	INDIRIZZO	NUMERO_D
Smith,John B.	123456789	1965-01-09	731 Fondren,Houston,TX	5
Wong,Franklin T.	333445555	1955-12-08	638 Voss,Houston,TX	5
Zelaya,Alicia J.	999887777	1968-07-19	3321 Castle,Spring,TX	4
Wallace,Jennifer S.	987654321	1941-06-20	291 Berry,Bellaire,TX	4
Narayan,Rmesh K.	666884444	1962-09-15	975 Fire Oak,Humble,TX	5
English,Joyce A.	453453453	1972-07-31	5631 Rice,Houston,TX	5
Jabbar,Ahmad V.	987987987	1969-03-29	980 Dallas,Houston,TX	4
Borg,James E.	888665555	1937-11-10	450 Stone,Houston,TX	1

DIPARTIMENTO			SEDI_DIP	
NOME_D	NUMERO_D	SSN_DIR_DIP	NUMERO_D	SEDE_D
Ricerca	5	333445555	1	Houston
Amministrazione	4	987654321	4	Stafford
Sede centrale	1	888665555	5	Bellaire
			5	Sugarland
			5	Houston

LAVORA_SU			PROGETTO		
SSN	NUMERO_P	ORE	NOME_P	NUMERO_P	SEDE_P
123456789	1	32.5	ProdottoX	1	Bellaire
123456789	2	7.5	ProdottoY	2	Sugarland
666884444	3	40.0	ProdottoZ	3	Houston
453453453	1	20.0	Computerizzazione	10	Stafford
453453453	2	20.0	Riorganizzazione	20	Houston
333445555	2	10.0	Nuove opportunità	30	Stafford
333445555	3	10.0			
333445555	10	10.0			
333445555	20	10.0			
999887777	30	30.0			
999887777	10	10.0			
987987987	10	35.0			
987987987	30	5.0			
987654321	30	20.0			
987654321	20	15.0			
888665555	20	null			

Figura 10.2 Relazioni d'esempio per lo schema di Figura 10.1.

ma di relazione IMPIEGATO è piuttosto semplice: ogni tupla rappresenta un impiegato, con i valori per il nome dell'impiegato (NOME_I), il numero di previdenza sociale (SSN), la data di nascita (DATA_N) e l'indirizzo (INDIRIZZO), nonché il numero del dipartimento per cui l'impiegato lavora (NUMERO_D). L'attributo NUMERO_D è una chiave esterna che rappresenta un'*associazione implicita* tra IMPIEGATO e DIPARTIMENTO. Anche la semantica degli schemi DIPARTIMENTO e PROGETTO è chiara; ogni tupla DIPARTIMENTO rappresenta un'entità dipartimento, e ogni tupla PROGETTO rappresenta un'entità progetto. L'attributo SSN_DIR_DIP di DIPARTIMENTO collega un dipartimento all'impiegato che ne è il direttore, mentre NUM_D di PROGETTO collega un progetto al dipartimento che lo controlla; entrambi gli attributi sono chiavi esterne.

La semantica degli altri due schemi di relazione in Figura 10.1 è un po' più complessa. Ogni tupla in SEDI_DIP fornisce un numero di dipartimento (NUMERO_D) e una delle sedi del dipartimento (SEDE_D). Ogni tupla in LAVORA_SU dà il numero di previdenza sociale di un impiegato (SSN), il numero di progetto di *uno dei* progetti su cui lavora l'impiegato (NUMERO_P) e il numero di ore settimanali lavorate dall'impiegato su quel progetto (ORE). Entrambi gli schemi hanno comunque un'interpretazione ben definita e non ambigua. Lo schema SEDI_DIP rappresenta un attributo multivaleure di DIPARTIMENTO, mentre LAVORA_SU rappresenta un'associazione M:N tra IMPIEGATO e PROGETTO. Tutti gli schemi di relazione in Figura 10.1 possono perciò essere considerati buoni dal punto di vista di una semantica chiara. La seguente linea guida informale sviluppa ulteriormente la progettazione di uno schema di relazione.

LINEA GUIDATA 1. Si progetti ogni schema di relazione in modo tale che sia semplice spiegarne il significato. Non si uniscano attributi provenienti da più tipi di entità e tipi di associazione in un'unica relazione. Intuitivamente, se uno schema di relazione corrisponde a un solo tipo di entità o a un solo tipo di associazione, il suo significato tende a essere chiaro. Altrimenti la relazione corrisponde a un miscuglio di più entità e associazioni e perciò diviene semanticamente oscura.

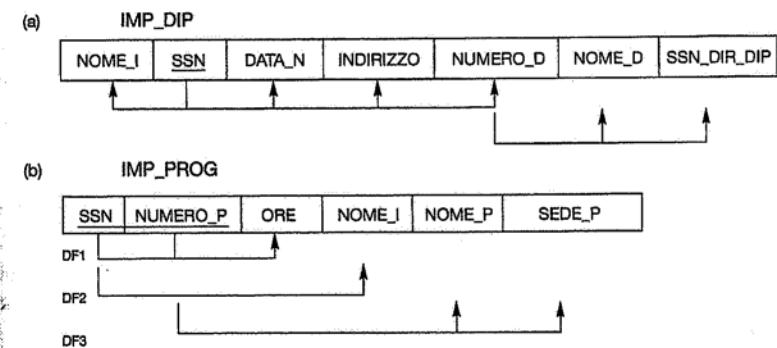


Figura 10.3 Due schemi di relazione e le rispettive dipendenze funzionali. Entrambi gli schemi presentano anomalie di aggiornamento. (a) Lo schema di relazione IMP_DIP. (b) Lo schema di relazione IMP_PROG.

Anche gli schemi di relazione nella Figura 10.3(a) e (b) presentano una semantica chiara (il lettore può per il momento ignorare le linee sotto le relazioni, dal momento che esse sono usate per illustrare la notazione della dipendenza funzionale che verrà presentata nel Paragrafo 10.2). Una tupla nello schema di relazione IMP_DIP di Figura 10.3(a) rappresenta un singolo impiegato ma comprende informazioni aggiuntive – vale a dire il nome (NOME_D) del dipartimento per cui l'impiegato lavora e il numero di previdenza sociale (SSN_DIR_DIP) del direttore del dipartimento. Per la relazione IMP_PROG di Figura 10.3(b), ogni tupla collega un impiegato a un progetto, ma comprende anche il nome dell'impiegato (NOME_I), il nome del progetto (NOME_P) e la sede del progetto (SEDE_P). Anche se non c'è alcunché di sbagliato dal punto di vista logico in queste due relazioni, esse sono considerate frutto di progetti mediocri, dato che violano la linea guida 1, mischiando attributi provenienti da distinte entità del mondo reale; IMP_DIP mescola attributi di impiegati e dipartimenti, e IMP_PROG mescola attributi di impiegati e progetti. Queste relazioni possono essere usate come viste, ma causano problemi quando vengono usate come relazioni di base, come si discuterà nel prossimo paragrafo.

10.1.2 Informazioni ridondanti nelle tuple e anomalie di aggiornamento

Uno scopo della progettazione di schemi è quello di ridurre al minimo lo spazio di memoria occupato dalle relazioni di base (file). Il raggruppamento di attributi in schemi di relazione ha un effetto significativo sullo spazio di memoria. Ad esempio, si confronti lo spazio occupato dalle due relazioni di base IMPIEGATO e DIPARTIMENTO in Figura 10.2 con lo spazio della relazione di base IMP_DIP in Figura 10.4, che è il risultato dell'applicazione dell'operazione di JOIN NATURALE a IMPIEGATO e DIPARTIMENTO. In IMP_DIP, i valori degli attributi che riguardano uno specifico dipartimento (NUMERO_D, NOME_D, SSN_DIR_DIP) sono ripetuti per *ogni impiegato che lavora per quel dipartimento*. Al contrario, l'informazione su ogni dipartimento appare solo una volta nella relazione DIPARTIMENTO di Figura 10.2. Solo il numero di dipartimento (NUMERO_D) è ripetuto nella relazione IMPIEGATO per ogni impiegato che lavora in quel dipartimento. Considerazioni simili si applicano alla relazione IMP_PROG (Figura 10.4), che aumenta la relazione LAVORA_SU con ulteriori attributi di IMPIEGATO e PROGETTO.

Un altro grave problema che si presenta usando le relazioni in Figura 10.4 come relazioni di base è il problema delle **anomalie di aggiornamento**. Esse possono essere classificate in anomalie di inserimento, anomalie di cancellazione e anomalie di modifica.²

Anomalie di inserimento. Possono essere distinte in due tipi, illustrati dai seguenti esempi basati sulla relazione IMP_DIP.

- Per inserire una nuova tupla impiegato in IMP_DIP occorre inserire i valori degli attributi del dipartimento per cui l'impiegato lavora, o valori nulli (se l'impiegato ancora non lavora per un dipartimento). Ad esempio, per inserire una nuova tupla per un impiegato che lavora nel dipartimento numero 5, bisogna inserire correttamente i valori degli attributi del dipartimento 5 in modo tale che siano *consistenti* con i valori del dipartimento 5 in altre tuple di IMP_DIP. Nel progetto di Figura 10.2 non ci si deve preoccupare di questo problema di consistenza, dato che nella tupla impiegato viene inserito solo il numero del dipartimento; tutti gli altri valori di attributi di dipartimento 5 sono registrati solo una volta nella base di dati, come tupla singola nella relazione DIPARTIMENTO.

IMP_DIP							
NOME_I	SSN	DATA_N.	INDIRIZZO	NUMERO_D	NOME_D	SSN_DIR_DIP	
Smith,John B.	123456789	1965-01-09	731 Fondren,Houston,TX	5	Ricerca	333445555	
Wong,Franklin T.	333445555	1955-12-08	638 Voss,Houston,TX	5	Ricerca	333445555	
Zelaya,Alicia J.	999887777	1968-07-19	3321 Castle Spring,TX	4	Amministrazione	987654321	
Wallace,Jennifer S.	987654321	1941-06-20	291 Berry,Bellaire,TX	4	Amministrazione	987654321	
Narayan,Ramesh K.	666884444	1962-09-15	975 FireOak,Humble,TX	5	Ricerca	333445555	
English,Joyce A.	453463483	1972-07-31	5631 Rice Houston,TX	5	Ricerca	333445555	
Jabbar,Ahmad V.	987987987	1969-03-29	980 Dallas Houston,TX	4	Amministrazione	987654321	
Borg,James E.	888665555	1937-11-10	450 Stone Houston,TX	1	Sede centrale	888665555	

IMP_PROG

SSN	NUMERO_P	ORE	NOME_I	NOME_P	SEDE_P
123456789	1	32.5	Smith,John B.	ProdottoX	Bellaire
123456789	2	7.5	Smith,John B.	ProdottoY	Sugarland
666884444	3	40.0	Narayan,Ramesh K.	ProdottoZ	Houston
453453453	1	20.0	English,Joyce A.	ProdottoX	Bellaire
453453453	2	20.0	English,Joyce A.	ProdottoY	Sugarland
333445555	2	10.0	Wong,Franklin T.	ProdottoY	Sugarland
333445555	3	10.0	Wong,Franklin T.	ProdottoZ	Houston
333445555	10	10.0	Wong,Franklin T.	Computerizzazione	Stafford
333445555	20	10.0	Wong,Franklin T.	Riorganizzazione	Houston
999887777	20	30.0	Zelaya,Alicia J.	Nuove opportunità	Stafford
999887777	30	10.0	Zelaya,Alicia J.	Computerizzazione	Stafford
987987987	10	35.0	Jabbar,Ahmad V.	Computerizzazione	Stafford
987987987	30	5.0	Jabbar,Ahmad V.	Nuove opportunità	Stafford
987654321	30	20.0	Wallace,Jennifer S.	Nuove opportunità	Stafford
987654321	20	15.0	Wallace,Jennifer S.	Riorganizzazione	Houston
888665555	20	null	Borg,James E.	Riorganizzazione	Houston

Figura 10.4 Relazioni esemplificative per gli schemi in Figura 10.3, ottenute applicando il JOIN NATURALE alle relazioni in Figura 10.2. Queste nuove relazioni possono essere memorizzate come relazioni di base per incrementare le prestazioni.

² Queste anomalie sono state messe in evidenza da Codd (1972b) per giustificare la necessità della normalizzazione delle relazioni, come si vedrà nel Paragrafo 10.3.

Anomalie di cancellazione. Questo problema è legato alla seconda situazione di anomalia di inserimento discussa sopra. Se si cancella da IMP_DIP una tupla impiegato che è quella che rappresenta l'ultimo impiegato che lavora per un particolare dipartimento, l'informazione riguardante quel dipartimento non è più presente nella base di dati. Il problema non si presenta nella base di dati di Figura 10.2 perché le tuple di DIPARTIMENTO sono memorizzate separatamente.

Anomalie di modifica. In IMP_DIP, se si cambia il valore di uno degli attributi di un particolare dipartimento – ad esempio il direttore del dipartimento 5 – occorre aggiornare le tuple di tutti gli impiegati che lavorano in quel dipartimento, altrimenti la base di dati diverrà inconsistente. Se ci si dimentica di aggiornare alcune tuple, per lo stesso dipartimento verranno mostrati due diversi valori di direttore in diverse tuple impiegato, il che non dovrebbe verificarsi.

Basandosi sulle tre anomalie precedenti, è possibile fornire la seguente linea guida.

LINEA GUIDA 2. Si progettino gli schemi di relazione di base in modo che nelle relazioni non siano presenti anomalie di inserimento, cancellazione o modifica. Se sono presenti delle anomalie, le si rilevi chiaramente e ci si assicuri che i programmi che aggiornano la base di dati operino correttamente.

La seconda linea guida è consistente con la prima e, in un certo senso, ne è una raffigurazione. Si può anche ravvisare la necessità di un approccio più formale per valutare se un progetto soddisfa queste linee guida (si vedano i Paragrafi 10.2-10.4). È importante notare che queste linee guida possono talora *dover essere violate* al fine di *incrementare le prestazioni* di certe interrogazioni. Ad esempio, se un'interrogazione importante recupera informazioni riguardanti il dipartimento di un impiegato, insieme con gli attributi di impiegato, lo schema IMP_DIP può essere usato come relazione di base. Le anomalie in IMP_DIP devono però essere notate e ben comprese, in modo tale che non si finisca col produrre inconsistenze ogni volta che viene aggiornata la relazione di base. In generale è consigliabile usare relazioni di base prive di anomalie, e specificare viste che utilizzino i JOIN per raggruppare gli attributi frequentemente richiamati in interrogazioni importanti. Ciò riduce il numero di termini JOIN specificati nell'interrogazione, rendendo più semplice scrivere l'interrogazione correttamente, e in molti casi migliora le prestazioni.³

10.1.3 Valori nulli nelle tuple

È possibile che nei progetti di alcuni schemi vengano raggruppati numerosi attributi a formare una “grossa” relazione. Se molti di questi attributi non riguardano tutte le tuple della relazione, quelle tuple finiscono con l'avere molti valori nulli. Ciò può dar luogo a uno spreco di spazio

³ Le prestazioni di un'interrogazione specificata su una vista che è il JOIN di molte relazioni di base dipende da come il DBMS implementa la vista. Molti DBMS relazionali memorizzano una vista usata frequentemente, per non dover eseguire spesso i JOIN. Il DBMS è responsabile di aggiornare la vista memorizzata (o immediatamente o periodicamente) ogni volta che vengono aggiornate le relazioni di base.

di memoria e può anche portare a problemi di comprensione del significato degli attributi e di specificazione delle operazioni di JOIN a livello logico.⁴ Un altro problema con i valori nulli è come rispondere della loro presenza quando vengono applicate operazioni aggregate come COUNT o SUM. Inoltre i valori nulli possono avere più interpretazioni, tra cui le seguenti:

- l'attributo *non è pertinente* per questa tupla;
- il valore dell'attributo per questa tupla è *sconosciuto*;
- il valore è *noto ma assente*, cioè non è ancora stato memorizzato.

L'uso della stessa rappresentazione per tutti i valori nulli non consente di cogliere i diversi significati che essi possono assumere. Si può pertanto enunciare un'altra linea guida.

LINEA GUIDA 3. Per quanto possibile, si eviti di porre in una relazione di base attributi i cui valori possono essere frequentemente nulli. Se i valori nulli sono inevitabili, ci si assicuri che essi si presentino solo in casi eccezionali e che non riguardino una maggioranza di tuple nella relazione.

Ad esempio, se solo il 10 per cento degli impiegati ha un ufficio personale, non c'è ragione di includere un attributo NUMERO_UFFICIO nella relazione IMPIEGATO; piuttosto può essere progettata una relazione IMP_UFFICI(SSN_I, NUMERO_UFFICIO) che comprenda tuple solo per gli impiegati che hanno uffici privati.

10.1.4 Generazione di tuple spurie

Si considerino i due schemi di relazione IMP_SEDI e IMP_PROG1 di Figura 10.5(a), che possono essere usati al posto della relazione IMP_PROG di Figura 10.3(b). Una tupla in IMP_SEDI indica che l'impiegato il cui nome è NOME_I lavora su *qualche progetto* la cui sede è SEDE_P. Una tupla in IMP_PROG1 indica che l'impiegato il cui numero di previdenza sociale è SSN lavora un certo numero di ORE a settimana sul progetto il cui nome, numero e sede sono NOME_P, NUMERO_P e SEDE_P. In Figura 10.5(b) sono mostrate estensioni di relazione di IMP_SEDI e IMP_PROG1 corrispondenti alla relazione IMP_PROG di Figura 10.4, ottenute applicando l'operazione di PROIEZIONE (π) appropriata a IMP_PROG (per il momento si trascurino le linee tratteggiate in Figura 10.5b).

Si supponga di aver usato IMP_PROG1 e IMP_SEDI invece di IMP_PROG come relazioni di base. Ciò dà luogo a un progetto di schema particolarmente infelice, perché da IMP_PROG1 e IMP_SEDI non si può recuperare l'informazione originariamente presente in IMP_PROG. Se si tenta un'operazione di JOIN NATURALE su IMP_PROG1 e IMP_SEDI, il risultato produce molte più tuple rispetto all'originaria popolazione di tuple in IMP_PROG. In Figura 10.6 è mostrato il risultato dell'applicazione del join alle sole tuple *sopra* le linee tratteggiate di Figu-

⁴ Ciò si verifica perché il join interno e quello esterno danno risultati diversi quando sono coinvolti valori nulli. Gli utenti dovrebbero perciò essere consapevoli dei diversi significati assunti dai vari tipi di join. Anche se ciò appare ragionevole per utenti sofisticati, potrebbe essere invece difficile per gli altri tipi di utenti.

(a)

IMP_SEDI	
NOME_I	SEDE_P
c.p.	

IMP_PROG1

SSN	NUMERO_P	ORE	NOME_P	SEDE_P
c.p.				

(b)

IMP_SEDI	
NOME_I	SEDE_P
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Bong, James E.	Houston

IMP_PROG1				
SSN	NUMERO_P	ORE	NOME_P	SEDE_P
123456789	1	32.5	Prodotto X	Bellaire
123456789	2	7.5	Prodotto Y	Sugarland
666884444	3	40.0	Prodotto Z	Houston
453453453	1	20.0	Prodotto X	Bellaire
453453453	2	20.0	Prodotto Y	Sugarland
333445555	2	10.0	Prodotto Z	Houston
333445555	3	10.0	Computerizzazione	Stafford
333445555	10	10.0	Computerizzazione	Houston
333445555	20	10.0	Riorganizzazione	Stafford
999887777	30	30.0	Nuove opportunità	Stafford
999887777	10	10.0	Computerizzazione	Stafford
987987987	10	35.0	Computerizzazione	Stafford
987987987	30	5.0	Nuove opportunità	Stafford
987654321	30	20.0	Nuove opportunità	Stafford
987654321	20	15.0	Riorganizzazione	Houston
888665555	20	null	Riorganizzazione	Houston

Figura 10.5 Rappresentazione alternativa (di cattiva qualità) della relazione IMP_PROG. (a) Rappresentazione di IMP_PROG di Figura 10.3(b) tramite due schemi di relazione: IMP_SEDI e IMP_PROG1. (b) Risultato della prolezione della relazione popolata IMP_PROG di Figura 10.4 sugli attributi di TMP_SEDI e IMP_PROG1.

ra 10.5(b) (per ridurre la dimensione della relazione risultato). Tuple aggiuntive, non presenti in IMP_PROG, sono dette *tuple spurie* perché rappresentano un'informazione spuria o *sbagliata* che non è valida. In Figura 10.6 le tuple spurie sono contrassegnate con asterischi (*).

La decomposizione di IMP_PROG in IMP_SEDI e IMP_PROG1 produce effetti indesiderati perché, quando si riuniscono le due relazioni usando il JOIN NATURALE, non si ottiene l'informazione originale corretta. Ciò si verifica perché in questo caso SEDE_P è l'attributo che collega IMP_SEDI e IMP_PROG1, e SEDE_P non è né una chiave primaria né una chiave esterna in IMP_SEDI o in IMP_PROG1. È possibile ora enunciare informalmente un'altra linea guida di progettazione.

LINIA GUIDA 4. Si progettino schemi di relazione in modo tale che essi possano essere riuniti, tramite JOIN, con condizioni di uguaglianza su attributi che sono o chiavi primarie o chiavi esterne in modo da garantire che non vengano generate tuple spurie. Non si abbiano relazioni che contengono attributi di accoppiamento diversi dalle combinazioni chiave esterna-chiave primaria. Se relazioni di questo tipo sono inevitabili, non si effettui su di esse un'operazione di join sulla base di questi attributi, perché il join può produrre tuple spurie.

Questa linea guida informale dovrà naturalmente essere esposta più formalmente. Nel Capitolo 11 verrà presentata una condizione formale, detta proprietà di join non-additivo (o senza perdita), che garantisce che certi join non producano tuple spurie.

SSN	NUMERO_P	ORE	NOME_P	SEDE_P	NOME_I
123456789	1	32.5	Prodotto X	Bellaire	Smith,John B.
* 123456789	1	32.5	Prodotto X	Bellaire	English,Joyce A.
123456789	2	7.5	Prodotto Y	Sugarland	Smith,John B.
* 123456789	2	7.5	Prodotto Y	Sugarland	English,Joyce A.
* 123456789	2	7.5	Prodotto Z	Sugarland	Wong,Franklin T.
666884444	3	40.0	Prodotto Z	Houston	Narayan,Ramesh K.
* 666884444	3	40.0	Prodotto Z	Houston	Wong,Franklin T.
* 453453453	1	20.0	Prodotto X	Bellaire	Smith,John B.
453453453	1	20.0	Prodotto X	Bellaire	English,Joyce A.
* 453453453	2	20.0	Prodotto Y	Sugarland	Smith,John B.
453453453	2	20.0	Prodotto Y	Sugarland	English,Joyce A.
* 453453453	2	20.0	Prodotto Y	Sugarland	Wong,Franklin T.
* 333445555	2	10.0	Prodotto Y	Sugarland	Smith,John B.
333445555	2	10.0	Prodotto Y	Sugarland	English,Joyce A.
* 333445555	2	10.0	Prodotto Z	Sugarland	Wong,Franklin T.
333445555	2	10.0	Prodotto Z	Sugarland	Narayan,Ramesh K.
* 333445555	3	10.0	Prodotto Z	Houston	Wong,Franklin T.
333445555	3	10.0	Prodotto Z	Houston	Wong,Franklin T.
* 333445555	10	10.0	Computerizzazione	Stafford	Wong,Franklin T.
* 333445555	20	10.0	Riorganizzazione	Houston	Narayan,Ramesh K.
333445555	20	10.0	Riorganizzazione	Houston	Wong,Franklin T.
					⋮

Figura 10.6 Risultato dell'applicazione dell'operazione di JOIN NATURALE alle tuple che stanno sopra le linee tratteggiate in IMP_PROG1 e IMP_SEDI; le tuple spurie che sono state generate sono indicate con un asterisco (*).

10.1.5 Sommario ed esame delle linee guida di progettazione

Dal Paragrafo 10.1.1 al Paragrafo 10.1.4 sono state esaminate informalmente situazioni che portano a schemi di relazione problematici, e sono state proposte linee guida informali per un buon progetto relazionale. I problemi evidenziati, che possono essere rivelati senza strumenti aggiuntivi di analisi, sono i seguenti:

- anomalie che richiedono che venga fatto un lavoro aggiuntivo durante l'inserimento in una relazione e la modifica di una relazione, e che possono causare una perdita accidentale d'informazione durante una cancellazione da una relazione;
- perdita di spazio di memoria a causa di valori nulli e difficoltà di eseguire operazioni di aggregazione e operazioni di join a causa della presenza di valori nulli;
- generazione di dati non validi e spuri durante i join su relazioni di base unite in modo improprio.

Nel resto del capitolo verranno presentati concetti formali e una teoria che possono essere usati per definire più precisamente i concetti di "buona qualità" e "cattiva qualità" dei *singoli* schemi di relazione. Verrà esaminata prima di tutto la dipendenza funzionale come strumento d'analisi. Quindi si specificheranno le tre forme normali e la forma normale di Boyce e Codd (BCNF: Boyce-Codd normal form) per schemi di relazione. Nel Capitolo 11 verranno forniti criteri aggiuntivi per verificare se un insieme di schemi di relazione formi, nel complesso, un buono schema di base di dati relazionale. Verranno presentati anche degli algoritmi che fanno parte di questa teoria per progettare basi di dati relazionali. Le forme normali definite in questo capitolo sono basate sul concetto di dipendenza funzionale, descritte nel prossimo paragrafo.

10.2 Dipendenze funzionali

Il concetto più importante nella progettazione di schemi relazionali è quello di dipendenza funzionale. In questo paragrafo si definirà formalmente questo concetto, mentre nel Paragrafo 10.3 si vedrà come esso possa essere usato per definire forme normali per schemi di relazione.

10.2.1 Definizione di dipendenza funzionale

Una dipendenza funzionale è un vincolo tra due insiemi di attributi della base di dati. Si supponga che il nostro schema di base di dati relazionale abbia n attributi A_1, A_2, \dots, A_n ; si pensi all'intera base di dati come se fosse descritta da un solo schema di relazione universale $R =$

$\{A_1, A_2, \dots, A_n\}$.⁵ Ciò non significa che si memorizzerà effettivamente la base di dati con una sola tabella universale; questo concetto verrà usato solo per sviluppare la teoria formale delle dipendenze tra dati.⁶

Una **dipendenza funzionale**, indicata con $X \rightarrow Y$, tra due insiemi di attributi X e Y che siano sottoinsiemi di R specifica un *vincolo* sulle tuple che possono formare uno stato di relazione r di R . Il vincolo è che, per ogni coppia di tuple t_1 e t_2 in r per le quali è $t_1[X] = t_2[X]$, si deve avere anche $t_1[Y] = t_2[Y]$. Ciò significa che i valori della componente Y di una tupla in r dipendono da, o sono *determinati da*, i valori della componente X , o, in alternativa, che i valori della componente X di una tupla *determinano* univocamente (o **funzionalmente**) i valori della componente Y . Si dice anche che c'è una dipendenza funzionale da X a Y o che Y è **funzionalmente dipendente** da X . L'abbreviazione per dipendenza funzionale è **DF** o **d.f.** (in ingl. **FD** o **f.d.**: *functional dependency*). L'insieme di attributi X è detto **parte sinistra** della DF, e Y è **parte destra**.

Pertanto X determina funzionalmente Y in uno schema di relazione R se e solo se, ogni volta che due tuple di $r(R)$ concordano sul loro valore X , esse devono necessariamente concordare anche sul loro valore Y . Si noti quanto segue:

- se un vincolo su R stabilisce che non ci possa essere più di una tupla con un dato valore per X in una generica istanza di relazione $r(R)$ – cioè X è una **chiave candidata** di R – ciò implica che $X \rightarrow Y$ per qualsiasi sottoinsieme Y di attributi di R (perché il vincolo di chiave implica che nessuna coppia di tuple in un qualsiasi stato valido $r(R)$ avrà lo stesso valore di X);
- se $X \rightarrow Y$ in R , ciò non dice se in R $Y \rightarrow X$ oppure no.

La dipendenza funzionale è una proprietà della **semantica** o del **significato degli attributi**. I progettisti della base di dati useranno la loro conoscenza della semantica degli attributi di R – vale a dire come sono collegati tra di loro – per specificare le dipendenze funzionali che dovrebbero sussistere su *tutti* gli stati di relazione (estensioni) r di R . Ogni volta che la semantica di due insiemi di attributi in R indica che deve sussistere una dipendenza funzionale, la dipendenza viene specificata con un vincolo. Estensioni di relazione $r(R)$ che soddisfano i vincoli di dipendenza funzionale sono dette **estensioni valide** (o **stati validi di relazione**) di R , perché soddisfano i vincoli di dipendenza funzionale. L'uso principale delle dipendenze funzionali è perciò quello di descrivere ulteriormente uno schema di relazione R , tramite una specificazione dei vincoli sui suoi attributi che devono valere *sempre*. Certe dipendenze funzionali possono essere specificate senza riferirsi a una particolare relazione, ma come una proprietà degli attributi coinvolti. Ad esempio, $\{\text{Stato}, \text{Numero_patente}\} \rightarrow \text{SSN}$ deve valere per ogni adulto negli Stati Uniti. È anche possibile che certe dipendenze funzionali possano cessare di esistere nel mondo reale se l'associazione cambia. Ad esempio,

⁵ Questo concetto di relazione universale è importante nello studio degli algoritmi per la progettazione di una base di dati relazionale.

⁶ Questa assunzione implica che ogni attributo nella base di dati debba avere un *nome distinto*. Nel Capitolo 7 abbiamo fatto precedere i nomi di attributi da nomi di relazioni per ottenere l'unicità ogni volta che attributi in relazioni diverse avessero lo stesso nome.

INSEGNA		
DOCENTE	INSEGNAMENTO	TESTO
Smith	Strutture dati	Bartram
Smith	Gestione dati	Al-Nour
Hall	Compilatori	Hoffman
Brown	Strutture dati	Augenthaler

Figura 10.7 Lo stato della relazione INSEGNA con una dipendenza funzionale apparente $\text{TESTO} \rightarrow \text{INSEGNAMENTO}$. $\text{INSEGNAMENTO} \rightarrow \text{TESTO}$ è invece da escludere.

un tempo negli Stati Uniti esisteva un'associazione tra i codici di avviamento postale e i prefissi telefonici, che si poteva esprimere tramite la dipendenza funzionale $\text{Codice_avv_postale} \rightarrow \text{Prefisso_telefonico}$, ma con la proliferazione dei prefissi telefonici ciò non è più vero.

Si consideri lo schema di relazione IMP_PROG in Figura 10.3(b); dalla semantica degli attributi sappiamo che devono sussistere le seguenti dipendenze funzionali:

- a. $\text{SSN} \rightarrow \text{NOME_I}$
- b. $\text{NUMERO_P} \rightarrow \{\text{NOME_P}, \text{SEDE_P}\}$
- c. $\{\text{SSN}, \text{NUMERO_P}\} \rightarrow \text{ORE}$

Queste dipendenze funzionali specificano che (a) il valore del numero di previdenza sociale di un impiegato (SSN) determina univocamente il nome dell'impiegato (NOME_I), (b) il valore del numero di un progetto (NUMERO_P) determina univocamente il nome del progetto (NOME_P) e la sua sede (SEDE_P), e (c) una combinazione dei valori di SSN e NUMERO_P determina univocamente il numero di ore lavorate a settimana sul progetto dall'impiegato (ORE). In alternativa si dice che NOME_I è determinato funzionalmente da (o è funzionalmente dipendente da) SSN, o che "dato un valore di SSN, noi sappiamo il valore di NOME_I" e così via.

Quella di dipendenza funzionale è una proprietà dello schema di relazione (intensione) R , non di un particolare stato valido di relazione (estensione) r di R . Perciò una DF non può essere dedotta automaticamente da una data estensione di relazione r , ma deve essere definita esplicitamente da qualcuno che conosce la semantica degli attributi di R . Ad esempio, in Figura 10.7 viene mostrato uno stato particolare dello schema di relazione INSEGNA. Anche se a prima vista si può pensare che $\text{TESTO} \rightarrow \text{INSEGNAMENTO}$, non è possibile confermare questa ipotesi se non si sa che ciò è vero per tutti i possibili stati validi di INSEGNA. È però sufficiente descrivere un solo controesempio per escludere una dipendenza funzionale. Ad esempio, dato che 'Smith' insegna sia 'Strutture dati' che 'Gestione dati', è possibile concludere che DOCENTE non determina funzionalmente INSEGNAMENTO.

In Figura 10.3 è introdotta una notazione diagrammatica per rappresentare graficamente le dipendenze funzionali: ogni DF è rappresentata graficamente con una linea orizzontale. Gli attributi della parte sinistra della DF sono collegati tramite linee verticali alla linea che rappresenta la DF, mentre gli attributi della parte destra sono collegati tramite frecce che puntano verso gli attributi stessi.

10.2.2 Regole di inferenza per dipendenze funzionali

Si indichi con F l'insieme di dipendenze funzionali specificate sullo schema di relazione R . Tipicamente il progettista dello schema specifica le dipendenze funzionali *semanticamente evidenti*; di solito, però, sussistono molte altre dipendenze funzionali in *tutte* le istanze di relazione valide che soddisfano le dipendenze in F . Queste altre dipendenze possono essere *inferite* o *dedotte* dalle dipendenze funzionali presenti in F . Per esempi tratti dalla vita reale è praticamente impossibile specificare tutte le dipendenze funzionali che possono sussistere. L'insieme di tutte queste dipendenze è detto **chiusura** di F ed è indicato con F^* . Ad esempio, si supponga di specificare il seguente insieme F di ovvie dipendenze funzionali sullo schema di relazione di Figura 10.3(a):

$$F = \{\text{SSN} \rightarrow \{\text{NOME_I}, \text{DATA_N}, \text{INDIRIZZO}, \text{NUMERO_D}\}, \\ \text{NUMERO_D} \rightarrow \{\text{NOME_D}, \text{SSN_DIR_DIP}\}\}$$

È possibile *inferire* da F le seguenti dipendenze funzionali aggiuntive:

$$\begin{aligned} \text{SSN} &\rightarrow \{\text{NOME_D}, \text{SSN_DIR_DIP}\}, \\ \text{SSN} &\rightarrow \text{SSN}, \\ \text{NUMERO_D} &\rightarrow \text{NOME_D} \end{aligned}$$

Una DF $X \rightarrow Y$ è *inferita* da un insieme di dipendenze F specificate su R se $X \rightarrow Y$ sussiste in *ogni* stato di relazione r che sia un'estensione valida di R ; cioè, ogni volta che r soddisfa tutte le dipendenze in F , in r sussiste anche $X \rightarrow Y$. La chiusura F^* di F è l'insieme di tutte le dipendenze funzionali che possono essere dedotte da F . Per determinare un modo sistematico per inferire dipendenze, occorre trovare un insieme di **regole di inferenza** che possono essere usate per inferire nuove dipendenze a partire da un insieme dato di dipendenze. Ora si considereranno alcune di queste regole di inferenza. Verrà usata la notazione $F \vdash X \rightarrow Y$ per indicare che la dipendenza funzionale $X \rightarrow Y$ è inferita dall'insieme di dipendenze funzionali F .

Nella discussione seguente quando vengono esaminate le dipendenze funzionali si userà una notazione abbreviata. Le variabili di attributo verranno concatenate e le virgole omesse per comodità. Perciò la DF $\{X,Y\} \rightarrow Z$ è abbreviata in $XY \rightarrow Z$, e la DF $\{X,Y,Z\} \rightarrow \{U,V\}$ è abbreviata in $XYZ \rightarrow UV$. Le seguenti sei regole (da RI1 a RI6) sono note regole di inferenza per dipendenze funzionali:

RI1 (regola riflessiva):⁷ se $X \supseteq Y$, allora $X \rightarrow Y$;

RI2 (regola di arricchimento):⁸ $\{X \rightarrow Y\} \vdash XZ \rightarrow YZ$;

RI3 (regola transitiva): $\{X \rightarrow Y, Y \rightarrow Z\} \vdash X \rightarrow Z$;

RI4 (regola di decomposizione, o di proiezione): $\{X \rightarrow YZ\} \vdash X \rightarrow Y$;

⁷ La regola riflessiva può anche essere enunciata come $X \rightarrow X$, ossia, ogni insieme di attributi determina funzionalmente se stesso.

⁸ La regola di arricchimento può anche essere enunciata come $\{X \rightarrow Y\} \vdash XZ \rightarrow Y$, ossia, l'arricchimento degli attributi di parte sinistra di una DF produce un'altra DF valida.

RI5 (regola di unione, o additiva): $\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$;

RI6 (regola pseudo-transitiva): $(X \rightarrow Y, WY \rightarrow Z) \vdash WX \rightarrow Z$.

La regola riflessiva (RI1) afferma che un insieme di attributi determina sempre se stesso o uno qualsiasi dei suoi sottoinsiemi, il che è ovvio. Dato che RI1 genera dipendenze che sono sempre vere, queste dipendenze sono dette banali. Formalmente una dipendenza funzionale $X \rightarrow Y$ è banale se $X \supseteq Y$; altrimenti è non-banale. La regola di arricchimento (RI2) sostiene che aggiungendo lo stesso insieme di attributi alla parte sinistra e alla parte destra di una dipendenza si ottiene un'altra dipendenza valida. Secondo la RI3 le dipendenze funzionali sono transitive. La regola di decomposizione (RI4) sostiene che si possono rimuovere attributi dalla parte destra di una dipendenza; l'applicazione ripetuta di questa regola può decomporre la DF $X \rightarrow \{A_1, A_2, \dots, A_n\}$ nell'insieme di dipendenze $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$. La regola di unione (RI5) consente di fare l'opposto: è possibile combinare un insieme di dipendenze $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$ nella singola DF $X \rightarrow \{A_1, A_2, \dots, A_n\}$.

Ognuna delle precedenti regole di inferenza può essere dimostrata a partire dalla definizione di dipendenza funzionale, con prova diretta o per assurdo. Una prova per assurdo suppone che la regola non valga e mostra che ciò porta a una contraddizione. Verrà ora dimostrato che le prime tre regole (da RI1 a RI3) sono valide. La seconda prova è per assurdo.

PROVA DI RI1

Si supponga che $X \supseteq Y$ e che esistano due tuple t_1 e t_2 in una certa istanza di relazione r di R tali che $t_1[X] = t_2[X]$. Allora $t_1[Y] = t_2[Y]$ perché $X \supseteq Y$; perciò in r deve valere $X \rightarrow Y$.

PROVA DI RI2 (PER ASSURDO)

Si supponga che in un'istanza di relazione r di R valga la $X \rightarrow Y$, ma che non valga la $XZ \rightarrow YZ$. Devono perciò esistere due tuple t_1 e t_2 in r tali che (1) $t_1[X] = t_2[X]$, (2) $t_1[Y] = t_2[Y]$, (3) $t_1[XZ] = t_2[XZ]$, e (4) $t_1[YZ] \neq t_2[YZ]$. Ciò non è possibile, perché da (1) e (3) si deduce (5) $t_1[Z] = t_2[Z]$, e da (2) e (5) si deduce (6) $t_1[YZ] = t_2[YZ]$, contraddicendo (4).

PROVA DI RI3

Si supponga che in una relazione r sussistano sia (1) $X \rightarrow Y$ che (2) $Y \rightarrow Z$. Allora, per ogni coppia di tuple t_1 e t_2 in r tali che $t_1[X] = t_2[X]$, si deve avere (3) $t_1[Y] = t_2[Y]$, dall'assunzione (1); perciò occorre anche avere (4) $t_1[Z] = t_2[Z]$, dalla (3) e dall'assunzione (2); quindi in r deve valere la $X \rightarrow Z$.

Usando argomentazioni simili è possibile provare le regole di inferenza da RI4 a RI6 e ogni altra regola di inferenza valida. Però un modo più semplice per dimostrare che una regola di inferenza per dipendenze funzionali è valida consiste nel provarla usando regole di inferenza che sono già state dimostrate valide. Ad esempio, si può provare le regole da RI4 a RI6 usando le regole da RI1 a RI3 come segue:

PROVA DI RI4 (USANDO DA RI1 A RI3)

1. $X \rightarrow YZ$ (data).
2. $YZ \rightarrow Y$ (usando RI1 e sapendo che $YZ \supseteq Y$).
3. $Y \rightarrow Y$ (usando RI3 su 1 e 2).

PROVA DI RI5 (USANDO DA RI1 A RI3)

1. $X \rightarrow Y$ (data).
2. $X \rightarrow Z$ (data).
3. $X \rightarrow XY$ (usando RI2 su 1 arricchendo con X ; si noti che $XX = X$).
4. $XY \rightarrow YZ$ (usando RI2 su 2 arricchendo con Y).
5. $X \rightarrow YZ$ (usando RI3 su 3 e 4).

PROVA DI RI6 (USANDO DA RI1 A RI3)

1. $X \rightarrow Y$ (data).
2. $WY \rightarrow Z$ (data).
3. $WX \rightarrow WY$ (usando RI2 su 1 arricchendo con W).
4. $WX \rightarrow Z$ (usando RI3 su 3 e 2).

È stato dimostrato da Armstrong (1974) che le regole di inferenza da RI1 a RI3 sono corrette e complete. Per corrette si intende che, dato un insieme di dipendenze funzionali F specificate su uno schema di relazione R , tutte le dipendenze che è possibile inferire da F usando le regole da RI1 a RI3 sussistono in ogni stato di relazione r di R che soddisfa le dipendenze in F . Per complete si intende che, usando ripetutamente le regole da RI1 a RI3 per inferire dipendenze finché non se ne possono dedurre più, si ottiene come risultato l'insieme completo di tutte le possibili dipendenze che possono essere dedotte da F . In altre parole l'insieme di dipendenze F^* , che è stato detto chiusura di F , può essere determinato da F usando solo le regole di inferenza da RI1 a RI3. Le regole di inferenza da RI1 a RI3 sono note come regole di inferenza di Armstrong.⁹

Tipicamente i progettisti di basi di dati dapprima specificano l'insieme F delle dipendenze funzionali che possono essere facilmente determinate dalla semantica degli attributi di R ; poi usano RI1, RI2 e RI3 per inferire ulteriori dipendenze funzionali, le quali saranno pure valide su R . Un modo sistematico per determinare queste dipendenze funzionali aggiuntive è quello di determinare prima di tutto ogni insieme X di attributi che appare come parte sinistra di qualche dipendenza funzionale in F , e poi di determinare l'insieme di tutti gli attributi che sono dipendenti da X . Perciò per ogni insieme X di attributi di questo tipo, si calcola l'insieme X^* di attributi che sono determinati funzionalmente da X sulla base di F ; X^* è detto chiusura di X rispetto a F . Per calcolare X^* può essere usato l'Algoritmo 10.1.

ALGORITMO 10.1 Determinazione di X^* , chiusura di X rispetto a F .

```

 $X^* := X;$ 
repeat
    old $X^* := X^*$ ;
    for ogni dipendenza funzionale  $Y \rightarrow Z$  in  $F$  do
        if  $X^* \supseteq Y$  then  $X^* := X^* \cup Z$ ;
    until ( $X^* = oldX^*$ );

```

⁹ A dire il vero, esse sono note come assiomi di Armstrong. Però, in senso matematico stretto, gli assiomi (fatti dati) sono le dipendenze funzionali in F , dato che si suppone che esse siano corrette, mentre le regole da RI1 a RI3 sono le regole di inferenza per inferire nuove dipendenze funzionali (nuovi fatti).

Tale algoritmo inizia ponendo X^* uguale a tutti gli attributi in X . Da RI1 è noto che tutti questi attributi sono funzionalmente dipendenti da X . Servendosi delle regole di inferenza RI3 e RI4 si aggiungono attributi a X^* , usando tutte le dipendenze funzionali in F . Si continuano a considerare tutte le dipendenze in F (il ciclo repeat) finché non vengono più aggiunti attributi a X^* durante un ciclo completo (il ciclo for) sulle dipendenze in F . Ad esempio, si consideri lo schema di relazione IMP_PROG di Figura 10.3(b); dalla semantica degli attributi è possibile specificare il seguente insieme F di dipendenze funzionali che devono valere su IMP_PROG:

$$\begin{aligned} F = & \{ \text{SSN} \rightarrow \text{NOME_I}, \\ & \text{NUMERO_P} \rightarrow \{\text{NOME_P}, \text{SEDE_P}\}, \\ & \{\text{SSN}, \text{NUMERO_P}\} \rightarrow \text{ORE} \} \end{aligned}$$

Usando l'Algoritmo 10.1 si possono calcolare i seguenti insiemi chiusura rispetto a F :

$$\{\text{SSN}\}^* = \{\text{SSN}, \text{NOME_I}\}$$

$$\{\text{NUMERO_P}\}^* = \{\text{NUMERO_P}, \text{NOME_P}, \text{SEDE_P}\}$$

$$\{\text{SSN}, \text{NUMERO_P}\}^* = \{\text{SSN}, \text{NUMERO_P}, \text{NOME_I}, \text{NOME_P}, \text{SEDE_P}, \text{ORE}\}$$

10.2.3 Equivalenza di insiemi di dipendenze funzionali

Un insieme di dipendenze funzionali E è coperto da un insieme di dipendenze funzionali F – alternativamente si dice che F copre E – se ogni DF in E è presente anche in F^* , cioè se ogni dipendenza in E può essere inferita da F . Due insiemi E e F di dipendenze funzionali sono equivalenti se $E^* = F^*$. Perciò l'equivalenza implica che ogni DF in E possa essere inferita da F , e ogni DF in F possa essere inferita da E , ossia E è equivalente a F se sussistono entrambe le condizioni E copre F e F copre E .

Si può determinare se F copre E calcolando X^* rispetto a F per ogni DF $X \rightarrow Y$ in E , e quindi verificando se questo X^* comprende gli attributi presenti in Y . Se è così per ogni DF in E , allora F copre E . Si determina se E e F sono equivalenti verificando se E copre F e F copre E .

10.2.4 Insiemi minimali di dipendenze funzionali

Un insieme F di dipendenze funzionali è minima se soddisfa le seguenti condizioni:

1. ogni dipendenza presente in F ha come parte destra un solo attributo;
2. non è mai possibile sostituire una dipendenza $X \rightarrow A$ di F con una dipendenza $Y \rightarrow A$, dove Y è un sottoinsieme proprio di X , e avere ancora un insieme di dipendenze equivalenti a F ;
3. non è mai possibile rimuovere una dipendenza da F e avere ancora un insieme di dipendenze equivalenti a F .

Si può pensare a un insieme minima di dipendenze come a un insieme di dipendenze in una forma standard o canonica e senza ridondanze. La condizione 1 assicura che ogni dipendenza si presenta in una forma canonica con un solo attributo nella parte destra.¹⁰ Le condizioni 2 e 3 assicurano che non ci siano ridondanze nelle dipendenze, o per la presenza di attributi ridondanti nella parte sinistra di una dipendenza (condizione 2), o per la presenza di una dipendenza che può essere inferita dalle altre DF in F (condizione 3). Una copertura minima di un insieme F di dipendenze funzionali è un insieme minima di dipendenze F_{\min} equivalenti a F . Purtroppo per uno stesso insieme di dipendenze funzionali ci possono essere molte coperture minimali. Usando l'Algoritmo 10.2 si può sempre trovare almeno una copertura minima G per ogni insieme F di dipendenze.

ALGORITMO 10.2 Ricerca di una copertura minima G di F .

1. Poni $G := F$.
2. Sostituisci ogni dipendenza funzionale $X \rightarrow \{A_1, A_2, \dots, A_n\}$ in G con le n dipendenze funzionali $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$.
3. Per ogni dipendenza funzionale $X \rightarrow A$ in G
 - per ogni attributo B che sia un elemento di X
se $((G - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\})$ è equivalente a G ,
allora sostituisci $X \rightarrow A$ con $(X - \{B\}) \rightarrow A$ in G .
4. Per ogni dipendenza funzionale rimanente $X \rightarrow A$ in G
 - se $(G - \{X \rightarrow A\})$ è equivalente a G ,
allora rimuovi $X \rightarrow A$ da G .

10.3 Forme normali basate su chiavi primarie

Dopo aver studiato le dipendenze funzionali e alcune delle loro proprietà, si è ora pronti a usarle come fonti d'informazione sulla semantica degli schemi di relazione. Si supporrà qui che per ogni relazione venga dato un insieme di dipendenze funzionali, e che ogni relazione abbia una chiave primaria designata; queste informazioni, combinate con i test (condizioni) per forme normali, guidano il processo di normalizzazione. Si rivolgerà l'attenzione alle prime tre forme normali per schemi di relazione e all'intuizione che sta dietro di esse, e si studierà come sono state sviluppate storicamente. Definizioni più generali di queste forme normali, che tengono conto di tutte le chiavi candidate di una relazione anziché solo della chiave primaria, sono rinviate al Paragrafo 10.4, mentre nel Paragrafo 10.5 verrà definita la forma normale di Boyce e Codd (BCNF).

¹⁰ Questa è una forma standard, non una necessità, ed è richiesta per semplificare le condizioni e gli algoritmi che garantiscono che non esista nessuna ridondanza in F . Usando le regole di inferenza RI4 e RI5, si può convertire una singola dipendenza con più attributi nella parte destra in un insieme di dipendenze, e viceversa.

10.3.1 Introduzione alla normalizzazione

Il processo di normalizzazione, così come è stato inizialmente proposto da Codd (1972b), sottopone uno schema di relazione a una serie di test per "certificare" se soddisfa a una certa **forma normale**. Il processo, che procede in modo top-down valutando ogni relazione con i criteri per le forme normali e decomponendo le relazioni quando necessario, può pertanto essere considerato come una *progettazione relazionale per analisi*. Inizialmente Codd ha proposto tre forme normali, che ha chiamato prima, seconda e terza forma normale. Una definizione più restrittiva di 3NF – detta forma normale di Boyce e Codd (BCNF) – è stata proposta in seguito da Boyce e Codd. Tutte queste forme normali sono basate sulle dipendenze funzionali tra gli attributi di una relazione. Più tardi sono state proposte una quarta forma normale (4NF) e una quinta forma normale (5NF), basate sui concetti di dipendenza multivaleore e dipendenza di join, rispettivamente. Nel Capitolo 15 si studierà come le relazioni in 3NF possano essere sintetizzate a partire da un insieme dato di DF. Quest'approccio è detto *progettazione relazionale per sintesi*.

La **normalizzazione dei dati** può perciò essere considerata come un processo di analisi degli schemi di relazione forniti, basato sulle loro dipendenze funzionali e chiavi primarie, per raggiungere le proprietà desiderate di (1) minimizzazione della ridondanza e (2) minimizzazione delle anomalie di inserimento, cancellazione e modifica discusse nel Sottoparagrafo 10.1.2. Schemi di relazione inadeguati, che non soddisfano certe condizioni – i **test di forma normale** – vengono decomposti in schemi di relazione più piccoli che superano i test e pertanto possiedono le proprietà desiderate. Quindi la procedura di normalizzazione fornisce ai progettisti di basi di dati:

- una struttura formale di analisi degli schemi di relazione basata sulle loro chiavi e sulle dipendenze funzionali tra i loro attributi;
- una serie di test di forma normale che possono essere effettuati sui singoli schemi di relazione, in modo che la base di dati relazionale possa essere **normalizzata** fino al livello desiderato, qualunque esso sia.

La **forma normale** di una relazione fa riferimento alla più alta condizione di forma normale soddisfatta, e perciò indica il livello al quale è stata normalizzata. Quando vengono considerate *separatamente* da altri fattori, le forme normali non garantiscono una buona progettazione di basi di dati. Di solito non è sufficiente verificare separatamente che ogni schema di relazione nella base di dati sia, ad esempio, in BCNF o in 3NF. Il processo di normalizzazione tramite decomposizione deve piuttosto confermare anche l'esistenza di proprietà aggiuntive che gli schemi di relazione, considerati nel loro insieme, devono possedere, come:

- la **proprietà di join senza perdita (lossless join)** o di **join non-additivo (nonadditive join)**, che garantisce che negli schemi di relazione creati dopo una decomposizione non si presenti il problema di generazione di tuple spurie discusso nel Sottoparagrafo 10.1.4;
- la **proprietà di conservazione delle dipendenze (dependency preservation)**, che assicura che ogni dipendenza funzionale sia rappresentata in una qualche relazione risultante.

La proprietà di join non-additivo è estremamente critica e deve essere raggiunta a ogni costo, mentre la proprietà di conservazione delle dipendenze, sebbene desiderabile, viene qualche volta sacrificata (si veda il Sottoparagrafo 15.1.2). Si rimanda la presentazione dei concetti e delle tecniche formali che garantiscono le due proprietà sopra esposte al Capitolo 15.

Per soddisfare altri criteri desiderabili possono essere definite altre forme normali, basate su altri tipi di vincoli. L'utilità pratica delle forme normali diventa però discutibile quando i vincoli su cui sono basate sono difficili da capire o da rilevare dai progettisti della base di dati e dagli utenti che devono scoprirli. Perciò oggi giorno la progettazione di basi di dati nell'industria presta particolare attenzione alla normalizzazione fino alla BCNF o alla 4NF.

Un altro punto degno di nota è che i progettisti di basi di dati *non sono obbligati* a normalizzare fino alla più alta forma normale possibile. Le relazioni possono essere lasciate a un livello più basso di normalizzazione per ragioni di prestazioni, del tipo di quelle esaminate alla fine del Sottoparagrafo 10.1.2. Il processo di memorizzazione del join di relazioni in forma normale più alta come una relazione di base – in una forma normale più bassa – è noto come **denormalizzazione**.

Prima di procedere oltre, occorre riprendere le definizioni di chiavi di uno schema di relazione viste nel Capitolo 7. Una **superchiave** di uno schema di relazione $R = \{A_1, A_2, \dots, A_n\}$ è un insieme di attributi $S \subseteq R$ con la proprietà che nessuna coppia di tuple t_1 e t_2 in un generico stato valido di relazione r di R avrà $t_1[S] = t_2[S]$. Una **chiave** K è una superchiave con la **proprietà aggiuntiva** che la rimozione di un qualsiasi attributo da K fa sì che K cessi di essere superchiave. La differenza tra una chiave e una superchiave sta nel fatto che una chiave deve essere **minimale**; cioè, se si ha una chiave $K = \{A_1, A_2, \dots, A_k\}$ di R , allora $K - \{A_i\}$ non è una chiave di R , qualsiasi sia i , $1 \leq i \leq k$. In Figura 10.1 {SSN} è una chiave per IMPIEGATO, mentre {SSN}, {SSN, NOME_I}, {SSN, NOME_I, DATA_N} ecc. sono tutte superchiavi.

Se uno schema di relazione ha più di una chiave, ognuna di esse è detta **chiave candidata**. Una delle chiavi candidate è *arbitrariamente* nominata **chiave primaria**, e le altre sono dette chiavi secondarie. Ogni schema di relazione deve avere una chiave primaria. In Figura 10.1 {SSN} è la sola chiave candidata per IMPIEGATO, e pertanto essa è anche la chiave primaria.

Un attributo di uno schema di relazione R è detto **attributo primo** di R se esso è membro di una *qualche chiave candidata* di R . Un attributo è detto **non-primo** se non è un attributo primo – cioè se non è membro di nessuna chiave candidata. In Figura 10.1 sia SSN sia NUMERO_P sono attributi primi di LAVORA_SU, mentre gli altri attributi di LAVORA_SU sono non-primi.

Verranno qui ora presentate le prime tre forme normali: 1NF, 2NF e 3NF, proposte da Codd (1972b) come una sequenza per raggiungere lo stato di relazioni in 3NF attraversando, se necessario, gli stati intermedi di 1NF e 2NF.

10.3.2 Prima forma normale

La **prima forma normale (1NF)** è ora considerata parte integrante della definizione formale di relazione nel modello relazionale di base (detto anche, in questo contesto, *flat*: piano, piatto). Storicamente è stata definita per non permettere l'uso di attributi multivaleore, di attributi

(a)

DIPARTIMENTO			
NOME_D	NUMERO_D	SSN_DIR_DIP	SEDI_D
			↑

(b)

DIPARTIMENTO			
NOME_D	NUMERO_D	SSN_DIR_DIP	SEDI_D
Ricerca	5	333445555	{Bellaire, Sugarland, Houston}
Amministrazione	4	987654321	{Stafford}
Sede centrale	1	888665555	{Houston}

(c)

DIPARTIMENTO			
NOME_D	NUMERO_D	SSN_DIR_DIP	SEDE_D
Ricerca	5	333445555	Bellaire
Ricerca	5	333445555	Sugarland
Ricerca	5	333445555	Houston
Amministrazione	4	987654321	Stafford
Sede centrale	1	888665555	Houston

Figura 10.8 Normalizzazione in 1NF. (a) Schema di relazione che non è in 1NF. (b) Istanza di relazione d'esempio. (c) Relazione in 1NF con ridondanza.

composti e delle loro combinazioni.¹¹ Essa richiede che il dominio di un attributo comprenda solo *valori atomici* (semplici, indivisibili) e che il valore di qualsiasi attributo in una tupla sia un *valore singolo* del dominio di quell'attributo. Perciò la 1NF non consente di avere un insieme di valori, una tupla di valori, o una combinazione di entrambi come valore di un attributo per una *tupla singola*. In altre parole, la 1NF non permette "relazioni entro relazioni" o "relazioni come attributi di tuple". I soli valori di attributi consentiti dalla 1NF sono i singoli *valori atomici* (o *indivisibili*).

Si consideri lo schema di relazione DIPARTIMENTO mostrato in Figura 10.1, la cui chiave primaria è NUMERO_D, e si supponga di estenderlo inserendo l'attributo SEDI_D come mostrato in Figura 10.8(a). Si supponga che ogni dipartimento possa avere *un certo numero di sedi*. Lo schema DIPARTIMENTO è un'estensione d'esempio sono mostrati in Figura 10.8. Come si può vedere, essa non è in 1NF perché SEDI_D non è un attributo atomico, come illustrato dalla prima tupla in Figura 10.8(b). Ci sono due modi di vedere l'attributo SEDI_D:

- il dominio di SEDI_D contiene valori atomici, ma alcune tuple possono avere un insieme di questi valori; in questo caso SEDI_D non è funzionalmente dipendente da NUMERO_D;

- il dominio di SEDI_D contiene insiemi di valori e perciò è non-atomico; in questo caso NUMERO_D → SEDI_D, perché ogni insieme è considerato un unico membro del dominio dell'attributo.¹²

In entrambi i casi la relazione DIPARTIMENTO di Figura 10.8 non è in 1NF; di fatto essa non si potrebbe nemmeno definire relazione, considerando la definizione di relazione data nel Paragrafo 7.1. Ci sono tre tecniche principali per raggiungere la prima forma normale per una relazione di questo tipo.

- Si rimuova l'attributo SEDI_D che viola la 1NF e lo si ponga in una relazione separata SEIDI_DIP insieme con la chiave primaria NUMERO_D di DIPARTIMENTO. La chiave primaria di questa relazione è la combinazione {NUMERO_D, SEDE_D}, come mostrato in Figura 10.2. In SEIDI_DIP esiste una tupla distinta per *ogni sede* di un dipartimento. Ciò decomponete la relazione che non è in 1NF in due relazioni in 1NF.
- Si espanda la chiave in modo tale che ci sia una tupla separata nella relazione DIPARTIMENTO originaria per ogni sede di un DIPARTIMENTO, come mostrato in Figura 10.8(c). In questo caso chiave primaria diventa la combinazione {NUMERO_D, SEDE_D}. Questa soluzione ha lo svantaggio di introdurre *ridondanza* nella relazione.
- Se è noto il *numero di massimo di valori* dell'attributo – ad esempio, se è noto che per un dipartimento possono esistere *al più tre sedi* – si sostituisca l'attributo SEDI_D con tre attributi atomici: SEDE_D1, SEDE_D2, e SEDE_D3. Questa soluzione ha lo svantaggio di introdurre *valori nulli* se la maggior parte dei dipartimenti ha meno di tre sedi.

Delle tre soluzioni sopra presentate, la prima è la migliore perché non presenta ridondanze ed è completamente generale, non presentando limiti sul numero massimo di valori. Infatti, se si sceglie la seconda soluzione, essa verrà ulteriormente decomposta, durante i successivi passi di normalizzazione, nella prima soluzione.

La prima forma normale non permette neppure attributi multivalore che siano essi stessi composti. Queste relazioni sono dette *relazioni nidificate* perché ogni tupla può avere *al suo interno* una relazione. In Figura 10.9 viene mostrato come potrebbe apparire la relazione IMP_PROG se fosse concessa la nidificazione. Ogni tupla rappresenta un'entità impiegato, e una relazione PROGG(NUMERO_P, ORE) *all'interno di ogni tupla* rappresenta i progetti su cui lavora l'impiegato e le ore settimanali lavorate dall'impiegato su ogni progetto. Lo schema di questa relazione IMP_PROG può essere rappresentato come segue:

IMP_PROG(SSN, NOME_I, {PROGG(NUMERO_P, ORE)})

Le parentesi graffe {} indicano che l'attributo PROGG è multivale, mentre gli attributi componenti che formano PROGG sono posti tra parentesi (). È interessante notare che una recente ricerca sul modello relazionale sta tentando di consentire e formalizzare le relazioni nidificate, che erano state inizialmente proibite dalla 1NF.

Si noti che SSN è la chiave primaria della relazione IMP_PROG nelle Figure 10.9(a) e (b), mentre NUMERO_P è la chiave primaria *parziale* della relazione nidificata; cioè la relazione ni-

¹¹ Il modello relazionale *nidificato* (*nested relational model*) e sistemi relazionali a oggetti (*object-relational systems*) (ORDBMS), consentono invece relazioni non normalizzate.

¹² In questo caso è possibile considerare il dominio di SEDI_D come l'*insieme potenza* dell'insieme delle singole sedi; vale a dire che il dominio è costituito da *tutti i sottoinsiemi possibili* dell'insieme delle singole sedi.

(a) IMP_PROG

SSN	NOME_I	PROGG	
		NUMERO_P	ORE
123456789	Smith,John B.	1	32.5
		2	7.5
666884444	Narayan,Ramesh K.	3	40.0
453453453	English,Joyce A.	1	20.0
		2	20.0
333445555	Wong,Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya,Alicia J.	30	30.0
		10	10.0
987987987	Jabbar,Ahmad V.	10	35.0
		30	5.0
987654321	Wallace,Jennifer S.	30	20.0
		20	15.0
888665555	Borg,James E.	20	null

(b) IMP_PROG

SSN	NOME_I	NUMERO_P	ORE
123456789	Smith,John B.	1	32.5
		2	7.5
666884444	Narayan,Ramesh K.	3	40.0
453453453	English,Joyce A.	1	20.0
		2	20.0
333445555	Wong,Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya,Alicia J.	30	30.0
		10	10.0
987987987	Jabbar,Ahmad V.	10	35.0
		30	5.0
987654321	Wallace,Jennifer S.	30	20.0
		20	15.0
888665555	Borg,James E.	20	null

(c) IMP_PROG1

SSN	NOME_I	
IMP_PROG2		
SSN	NUMERO_P	ORE

Figura 10.9 Normalizzazione in 1NF di relazioni nidificate. (a) Schema della relazione IMP_PROG con una "relazione nidificata" PROGG. (b) Estensione d'esempio della relazione IMP_PROG che mostra relazioni nidificate all'interno di ciascuna tupla. (c) Decomposizione di IMP_PROG nelle relazioni in 1NF IMP_PROG1 e IMP_PROG2 tramite propagazione della chiave primaria.

dificata deve avere valori univoci per NUMERO_P, all'interno di ogni tupla. Per normalizzare questa relazione in 1NF si spostano gli attributi della relazione nidificata in una nuova relazione e *si propaga la chiave primaria* al suo interno; la chiave primaria della nuova relazione combinerà la chiave parziale con la chiave primaria della relazione originaria. La decomposizione e la propagazione della chiave primaria producono gli schemi IMP_PROG1 e IMP_PROG2 di Figura 10.9(c).

Questa procedura può essere applicata ricorsivamente a una relazione con nidificazione a più livelli per **eliminare la nidificazione (unnest)** dalla relazione, portandola a un insieme di relazioni in 1NF. Ciò è utile per convertire uno schema di relazione non normalizzato con mol-

ti livelli di nidificazione in relazioni in 1NF. L'operatore unnest fa parte del modello relazionale nidificato. Limitare le relazioni alla 1NF porta ai problemi associati alle dipendenze multivalore e alla 4NF.

10.3.3 Seconda forma normale

La **seconda forma normale (2NF)** si basa sul concetto di *dipendenza funzionale completa*. Una dipendenza funzionale $X \rightarrow Y$ è una **dipendenza funzionale completa** se la rimozione di qualsiasi attributo A da X comporta che la dipendenza non sussista più; cioè, per ogni attribu-

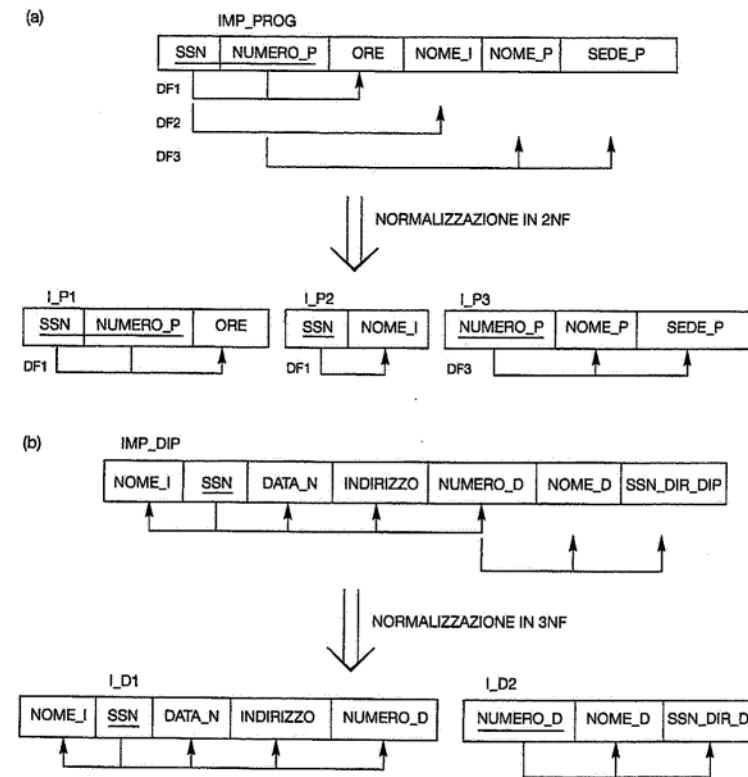


Figura 10.10 Il processo di normalizzazione. (a) Normalizzazione di IMP_PROG in relazioni in 2NF. (b) Normalizzazione di IMP_DIP in relazioni in 3NF.

to $A \in X$, $(X - \{A\})$ non determina funzionalmente Y . Una dipendenza funzionale $X \rightarrow Y$ è una **dipendenza parziale** se si possono rimuovere da X certi attributi $A \in X$ e la dipendenza continua a sussistere; cioè, per qualche $A \in X$, $(X - \{A\}) \rightarrow Y$. In Figura 10.3(b) $\{\text{SSN}, \text{NUMERO_P}\} \rightarrow \text{ORE}$ è una dipendenza completa (non sussiste né $\text{SSN} \rightarrow \text{ORE}$ né $\text{NUMERO_P} \rightarrow \text{ORE}$). La dipendenza $\{\text{SSN}, \text{NUMERO_P}\} \rightarrow \text{NOME_I}$ è invece parziale perché sussiste la $\text{SSN} \rightarrow \text{NOME_I}$.

Il test per la 2NF comporta l'esame di dipendenze funzionali i cui attributi di parte sinistra fanno parte della chiave primaria. Se la chiave primaria contiene un solo attributo è inutile effettuare il test. Uno schema di relazione R è in 2NF se ogni attributo non-primo A di R dipende funzionalmente in modo completo dalla chiave primaria di R . La relazione IMP_PROG di Figura 10.3(b) è in 1NF ma non è in 2NF. L'attributo non-primo NOME_I viola la 2NF a causa di DF2, come fanno gli attributi non-primi NOME_P e SEDE_P a causa di DF3. Le dipendenze funzionali DF2 e DF3 rendono NOME_I, NOME_P e SEDE_P parzialmente dipendenti dalla chiave primaria $\{\text{SSN}, \text{NUMERO_P}\}$ di IMP_PROG, violando così il test di 2NF.

Se uno schema di relazione non è in 2NF, esso può essere "normalizzato in 2NF", scomponendolo in un certo numero di relazioni in 2NF, nelle quali gli attributi non-primi sono associati solo alla parte della chiave primaria da cui sono funzionalmente dipendenti in modo completo. Le dipendenze funzionali DF1, DF2 e DF3 di Figura 10.3(b) portano perciò alla decomposizione di IMP_PROG nei tre schemi di relazione I_P1, I_P2 e I_P3 di Figura 10.10(a), ognuno dei quali è in 2NF.

10.3.4 Terza forma normale

La **terza forma normale (3NF)** è basata sul concetto di **dipendenza transitiva**. Una dipendenza funzionale $X \rightarrow Y$ in uno schema di relazione R è una **dipendenza transitiva** se esiste un insieme di attributi Z , che non è né una chiave candidata né un sottoinsieme di una chiave di R ,¹³ per cui valgono contemporaneamente $X \rightarrow Z$ e $Z \rightarrow Y$. La dipendenza $\text{SSN} \rightarrow \text{SSN_DIR_DIP}$ in IMP_DIP di Figura 10.3(a) è transitiva attraverso NUMERO_D perché sussistono entrambe le dipendenze $\text{SSN} \rightarrow \text{NUMERO_D}$ e $\text{NUMERO_D} \rightarrow \text{SSN_DIR_DIP}$, e NUMERO_D non è né una chiave di per sé né un sottoinsieme della chiave di IMP_DIP. Intuitivamente si può vedere che la dipendenza di SSN_DIR_DIP da NUMERO_D in IMP_DIP è sgradita, dal momento che NUMERO_D non è una chiave di IMP_DIP.

Stando alla definizione originaria di Codd uno schema di relazione R è in 3NF se soddisfa la 2NF e nessun attributo non-primo di R dipende in modo transitivo dalla chiave primaria. Lo schema di relazione IMP_DIP in Figura 10.3(a) è in 2NF, dal momento che non esistono dipendenze funzionali parziali da una chiave. IMP_DIP non è però in 3NF a causa della dipendenza transitiva di SSN_DIR_DIP (e anche NOME_D) da SSN attraverso NUMERO_D.

¹³ Questa è la definizione generale di dipendenza transitiva. Dato che in questo paragrafo ci interessiamo solo delle chiavi primarie, concediamo le dipendenze transitive in cui X è la chiave primaria, ma Z può essere (sottoinsieme di) una chiave candidata.

Tabella 10.1 Sommario delle forme normali basate su chiavi primarie e corrispondente normalizzazione.

Forma Normale	Test	Rimedio (Normalizzazione)
Prima (1NF)	La relazione non deve avere attributi non-atomici o relazioni ridificate.	Formare nuove relazioni per ogni attributo non-atomico o relazione ridificata.
Seconda (2NF)	Per relazioni in cui la chiave primaria contiene più attributi, nessun attributo non-chiave deve essere funzionalmente dipendente da una parte della chiave primaria.	Decomporre e preparare una nuova relazione per ogni chiave parziale con i suoi attributi dipendenti. Assicurarsi di mantenere una relazione con la chiave primaria originale e tutti gli attributi funzionalmente dipendenti in modo completo da essa.
Terza (3NF)	La relazione non deve avere un attributo non-chiave determinato funzionalmente da un altro attributo non-chiave (o da un insieme di attributi non-chiave). Ciò non deve esserci nessuna dipendenza transitiva di un attributo non-chiave dalla chiave primaria.	Decomporre e preparare una relazione che comprenda gli attributi non-chiave che determinano funzionalmente altri attributi non-chiave.

Si può normalizzare IMP_DIP decomponendolo nei due schemi di relazione in 3NF I_D1 e I_D2 mostrati in Figura 10.10(b). Intuitivamente si vede che I_D1 e I_D2 rappresentano fatti di entità indipendenti sugli impiegati e sui dipartimenti. Un'operazione di JOIN NATURALE su I_D1 e I_D2 recupererà la relazione originale IMP_DIP senza generare tuple spurious.

La Tabella 10.1 riassume informalmente le tre forme normali basate sulle chiavi primarie, i test usati nei vari casi e il corrispondente "rimedio" o normalizzazione per ottenere la forma normale.

10.4 Definizioni generali di seconda e terza forma normale

In generale, si vogliono progettare schemi di relazione in modo tale che non presentino né dipendenze parziali né dipendenze transitive, perché questi tipi di dipendenze provocano le anomalie di aggiornamento discusse nel Sottoparagrafo 10.1.2. I passi per la normalizzazione in relazioni in 3NF trattati finora non consentono dipendenze parziali e transitive sulla **chiave primaria**. Queste definizioni, però, non tengono conto delle altre chiavi candidate di una relazione, ammesso che ve ne siano. In questo Paragrafo si forniranno le definizioni più generali di 2NF e 3NF, che tengono conto di *tutte* le chiavi candidate di una relazione. Si noti che ciò non riguarda la definizione di 1NF, dal momento che essa non dipende da chiavi e dipendenze funzionali. Come definizione generale di **attributo primo**, verrà considerato primo un attributo che faccia parte di *una qualsiasi chiave candidata*. Le dipendenze funzionali parziali e totali nonché le dipendenze transitive saranno ora *relative a tutte le chiavi candidate* di una relazione.

che provoca la dipendenza transitiva) in un'altra relazione LOTTI1B. Sia LOTTI1A sia LOTTI1B sono in 3NF. Ci sono due aspetti degni di nota al riguardo della definizione generale di 3NF:

- LOTTI1 viola la 3NF perché PREZZO è transitivamente dipendente da ognuna delle chiavi candidate di LOTTI1 tramite l'attributo non-primo AREA;
- questa definizione può essere applicata *direttamente* per verificare se uno schema di relazione è in 3NF; *non* è necessario passare prima attraverso la 2NF. Se si applica la definizione di 3NF sopra vista a LOTTI con le dipendenze da DF1 a DF4, si trova che DF3 e DF4 violano *entrambe* la 3NF. Sarebbe perciò possibile decomporre direttamente LOTTI in LOTTI1A, LOTTI1B e LOTTI12. Quindi le dipendenze transitive e parziali che violano la 3NF possono essere rimosse *in qualsiasi ordine*.

10.4.3 Interpretazione della definizione generale di 3NF

Uno schema di relazione R viola la definizione generale di 3NF se sussiste in R una dipendenza funzionale $X \rightarrow A$ che viola *entrambe* le condizioni (a) e (b) di 3NF. La violazione di (b) comporta che A sia un attributo non-primo. La violazione di (a) comporta che X non sia un sovrainsieme di qualche chiave di R ; perciò X potrebbe essere non-primo o potrebbe essere un sottoinsieme proprio di una chiave di R . Se X è non-primo si ha tipicamente una dipendenza transitiva che viola la 3NF, mentre se X è un sottoinsieme proprio di una chiave di R si ha una dipendenza parziale che viola la 3NF (e anche la 2NF). È perciò possibile enunciare una **definizione generale alternativa di 3NF** come segue: uno schema di relazione R è in 3NF se ogni attributo non-primo di R soddisfa entrambe le condizioni seguenti:

- è funzionalmente dipendente in modo completo da ogni chiave di R ;
- non è dipendente in modo transitivo da nessuna chiave di R .

10.5 Forma normale di Boyce e Codd

La **forma normale di Boyce e Codd (BCNF)** è stata proposta come una forma più semplice di 3NF, ma si è rivelata più restrittiva della 3NF; infatti, ogni relazione in BCNF è anche in 3NF, mentre una relazione in 3NF *non è necessariamente* in BCNF. Si può intuitivamente ravvisare la necessità di una forma normale più forte della 3NF tornando allo schema di relazioni LOTTI di Figura 10.11(a), con le sue quattro dipendenze funzionali, da DF1 a DF4. Si supponga di avere migliaia di lotti nella relazione, ma che i lotti vengano solo da due contee: Dekalb e Fulton. Si supponga inoltre che le dimensioni possibili dei lotti nella contea di Dekalb siano ristrette a 0,5, 0,6, 0,7, 0,8, 0,9, e 1,0 acri, mentre le dimensioni possibili dei lotti nella contea di Fulton si limitino a 1,1, 1,2, ..., 1,9 e 2,0 acri. In questa situazione si avrà la dipendenza funzionale aggiuntiva FD5: AREA \rightarrow NOME_CONTEA. Se la si aggiunge alle altre dipendenze, lo schema di relazione LOTTI1A rimane in 3NF perché NOME_CONTEA è un attributo primo.

L'area di un lotto che determina la contea, come specificato da DF5, può essere rappresentata da 16 tuple in una relazione separata $R(\text{AREA}, \text{NOME_CONTEA})$, dal momento che ci sono solo 16 possibili valori per AREA. Questa rappresentazione riduce la ridondanza insita nel ripetere la stessa informazione in migliaia di tuple di LOTTI1A. La BCNF è una *forma normale più restrittiva*, che non permette LOTTI1A e fornisce indicazioni per decomporla.

La definizione formale di BCNF differisce solo leggermente dalla definizione di 3NF. Uno schema di relazione R è in BCNF se, ogni volta che sussiste in R una dipendenza funzionale *non-banale* $X \rightarrow A$, X è una superchiave di R . La sola differenza tra le definizioni di BCNF e di 3NF è che la condizione (b) della 3NF, che consente ad A di essere primo, è assente dalla BCNF.

Nel nostro esempio DF5 viola la BCNF in LOTTI1A, perché AREA non è una superchiave di LOTTI1A. Si noti che DF5 soddisfa la 3NF in LOTTI1A perché NOME_CONTEA è un attributo primo (condizione b), ma questa condizione non esiste nella definizione di BCNF. Si può decomporre LOTTI1A nelle due relazioni in BCNF LOTTI1AX e LOTTI1AY di Figura 10.12(a). Questa decomposizione perde la dipendenza funzionale DF2 perché i suoi attributi non coesistono più nella stessa relazione.

In pratica, la maggior parte degli schemi di relazione in 3NF è anche in BCNF. Solo se in uno schema di relazione R sussiste $X \rightarrow A$, con X che non è superchiave e A attributo primo, R sarà in 3NF ma non in BCNF. Lo schema di relazione R di Figura 10.12(b) illustra il caso generale di una relazione di questo tipo. Idealmente la progettazione di una base di dati relazionale dovrebbe sforzarsi di raggiungere la BCNF o la 3NF per ogni schema di relazione. Il

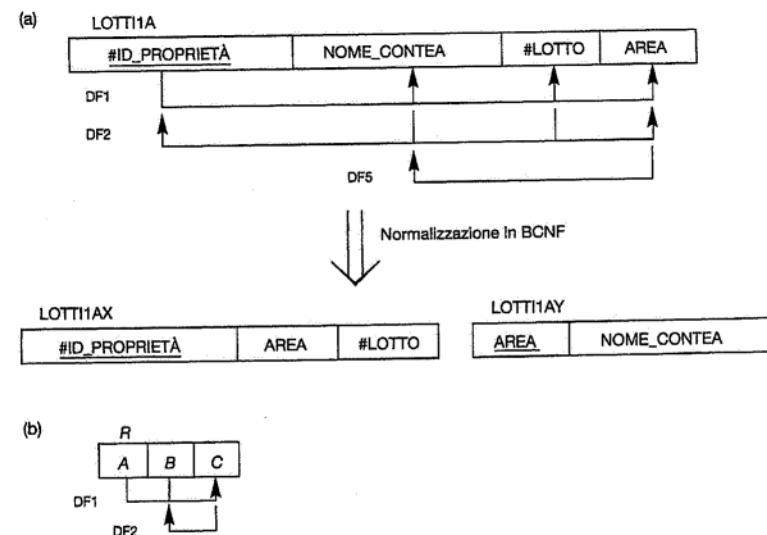


Figura 10.12 Forma normale di Boyce e Codd. (a) Normalizzazione in BCNF con la dipendenza DF2 che viene "persa" nella decomposizione. (b) Una relazione R in 3NF ma non in BCNF.

INSEGNA		
STUDENTE	INSEGNAMENTO	DOCENTE
Narayan	Basi di dati	Mark
Smith	Basi di dati	Navathe
Smith	Sistemi operativi	Ammar
Smith	Informatica teorica	Schulman
Wallace	Basi di dati	Mark
Wallace	Sistemi operativi	Ahamad
Wong	Basi di dati	Omieciński
Zelaya	Basi di dati	Navathe

Figura 10.13 Una relazione INSEGNA che è in 3NF ma non in BCNF.

raggiungimento dello stato di normalizzazione a livello della sola 1NF o 2NF non è considerato sufficiente, dal momento che esse sono state storicamente sviluppate come punti di partenza verso la 3NF e la BCNF. In Figura 10.13 è mostrata una relazione INSEGNA con le seguenti dipendenze:

DF1: {STUDENTE, INSEGNAMENTO} → DOCENTE

DF2:¹⁵ DOCENTE → INSEGNAMENTO

Si noti che {STUDENTE, INSEGNAMENTO} è una chiave candidata per questa relazione e che le dipendenze viste seguono il modello illustrato in Figura 10.12(b). Pertanto questa relazione è in 3NF ma non in BCNF. La decomposizione di questo schema di relazione in due schemi distinti non è immediata, dal momento che esso può essere decomposto in una delle tre coppie possibili:

1. {STUDENTE, DOCENTE} e {STUDENTE, INSEGNAMENTO};
2. {INSEGNAMENTO, DOCENTE} e {INSEGNAMENTO, STUDENTE};
3. {DOCENTE, INSEGNAMENTO} e {DOCENTE, STUDENTE}.

Tutte e tre le decomposizioni "perdonano" la dipendenza funzionale DF1. La decomposizione preferibile tra le tre sopra viste è la terza, perché non genererà tuple spurie dopo un join. Un test per determinare se una decomposizione è non-additiva (senza perdita) sarà esaminato nel Sottoparagrafo 11.1.3 nella proprietà LJ1. In generale una relazione non in BCNF dovrebbe essere decomposta senza perdita, rinunciando eventualmente a preservare tutte le dipendenze funzionali nelle relazioni decomposte, come accade in questo esempio. L'Algoritmo 11.3 fa esattamente ciò, e avrebbe potuto essere usato sopra per fornire la stessa decomposizione per INSEGNA.

Sommario

In questo capitolo abbiamo studiato su base intuitiva molte insidie presenti nella progettazione di una base di dati relazionale, abbiamo individuato informalmente alcune delle misure usate per indicare se uno schema di relazione è "buono" o "cattivo", e fornito linee guida informali per una buona progettazione. Abbiamo quindi presentato alcuni concetti formali che ci consentono di effettuare la progettazione relazionale in modo top-down, analizzando le relazioni una ad una. Si è definito questo procedimento di progettazione per analisi e decomposizione introducendo il processo di normalizzazione. Gli argomenti studiati in questo capitolo saranno ripresi nel prossimo capitolo, dove esamineremo concetti più avanzati di teoria di progettazione relazionale.

Abbiamo esaminato i problemi delle anomalie di aggiornamento, che si presentano quando si verificano ridondanze nelle relazioni. Misure informali di buoni schemi di relazione prevedono una semantica degli attributi semplice e chiara e pochi valori nulli nelle estensioni delle relazioni. Una buona decomposizione, inoltre, deve evitare il problema della generazione di tuple spurie come risultato dell'operazione di join.

Abbiamo definito il concetto di dipendenza funzionale e discusso alcune delle sue proprietà. Le dipendenze funzionali costituiscono la sorgente fondamentale di informazione semantica sugli attributi di uno schema di relazione. Abbiamo mostrato come da un insieme dato di dipendenze funzionali possano essere inferite altre dipendenze usando un insieme di regole di inferenza. Sono stati definiti i concetti di chiusura e copertura minimale di un insieme di dipendenze ed è stato fornito un algoritmo per calcolare una copertura minimale. Si è anche mostrato come verificare se due insiemi di dipendenze funzionali sono equivalenti.

Abbiamo quindi descritto il processo di normalizzazione, che permette di ottenere progetti di buona qualità esaminando relazioni alla ricerca di dipendenze funzionali indesiderate. È stato fornito un trattamento di normalizzazione successiva basato su una chiave primaria predefinita in ogni relazione, quindi si è rilassato questo requisito e sono state fornite definizioni più generali di seconda forma normale (2NF) e terza forma normale (3NF) che prendono in considerazione tutte le chiavi candidate di una relazione. Sono stati presentati poi degli esempi che permettono di illustrare come, usando la definizione generale di 3NF, una data relazione possa essere analizzata e decomposta per produrre alla fine un insieme di relazioni in 3NF.

Infine abbiamo illustrato la forma normale di Boyce e Codd (BCNF) e si è visto come essa sia una forma più forte di 3NF. Abbiamo anche mostrato come debba essere effettuata la decomposizione di una relazione non-BCNF considerando il requisito di decomposizione non-additiva.

Il Capitolo 11 presenterà algoritmi di sintesi e di decomposizione per la progettazione di basi di dati relazionali basata su dipendenze funzionali. In relazione alla decomposizione esamineremo i concetti di *join senza perdita (non-additivo)* e *conservazione delle dipendenze*, che sono imposti da alcuni di questi algoritmi.

¹⁵ Ciò presume che un vincolo di questa applicazione sia "ogni docente tiene un solo insegnamento".

Questionario di verifica

- 10.1. Si discuta sulla semantica degli attributi vista come misura informale di qualità di uno schema di relazione.
- 10.2. Si esaminino le anomalie di inserimento, cancellazione e modifica. Perché sono considerate pericolose? Si illustri la risposta con esempi.
- 10.3. Perché non è consigliabile avere molti valori nulli in una relazione?
- 10.4. Si tratti il problema delle tuple spurie e si esaminino le relative modalità di prevenzione.
- 10.5. Si enuncino le linee guida informali per la progettazione di schemi di relazione discussi nel corso del capitolo. Si illustri come una violazione di queste linee guida possa essere dannosa.
- 10.6. Cos'è una dipendenza funzionale? Chi specifica le dipendenze funzionali esistenti fra gli attributi di uno schema di relazione?
- 10.7. Perché non si può dedurre una dipendenza funzionale da uno specifico stato di relazione?
- 10.8. Perché sono importanti le regole di inferenza di Armstrong – le tre regole di inferenza dalla RI1 alla RI3?
- 10.9. Cosa si intende per completezza e correttezza delle regole di inferenza di Armstrong?
- 10.10. Cosa si intende per chiusura di un insieme di dipendenze funzionali?
- 10.11. Quando si dice che due insiemi di dipendenze funzionali sono equivalenti? Come possiamo determinare la loro equivalenza?
- 10.12. Cos'è un insieme minimale di dipendenze funzionali? Per ogni insieme di dipendenze esiste un insieme minimale equivalente?
- 10.13. A cosa fa riferimento la locuzione *relazione non normalizzata*? Come si sono sviluppate storicamente le forme normali?
- 10.14. Si definiscano prima, seconda e terza forma normale considerando solo le chiavi primarie. Come differiscono le definizioni generali di 2NF e 3NF, che considerano tutte le chiavi di una relazione, da quelle che considerano solo le chiavi primarie?
- 10.15. Quali dipendenze indesiderabili sono evitate quando una relazione è in 3NF?
- 10.16. Si definisca la forma normale di Boyce e Codd. In cosa differisce dalla 3NF? Perché è considerata una forma più forte di 3NF?

Esercizi

- 10.17. Si supponga di disporre dei seguenti requisiti per una base di dati università usata per tener traccia delle trascrizioni dei libretti universitari degli studenti.
 - a. L'università tiene traccia del nome di ogni studente (NOME_STUD), numero dello studente (NUM_STUD), numero di previdenza sociale (SSN), indirizzo corrente (IND_CORR_STUD) e telefono corrente (TELEFONO_CORR_STUD), indirizzo permanente (IND_PERM_STUD) e telefono permanente (TELEFONO_PERM_STUD), data di nascita (DATA_N), sesso (SESSO), anno di corso (ANNO_CORSO) (primo, secondo, ..., laureato), dipartimento di specializzazione principale (CODICE_DIP_SPEC_PRINC),

dipartimento di specializzazione secondaria (CODICE_DIP_SPEC_SEC) (se esiste) e programma di studi (PROG) (B.A., B.S., ..., PH.D.). Sia SSN sia numero studente assumono valori univoci per ogni studente.

- b. Ogni dipartimento è descritto da nome (NOME_D), codice dipartimento (CODICE_D), numero di ufficio (UFFICO_D), telefono di ufficio (TELEFONO_D) e college (COLLEGEGE_D). Sia nome sia codice assumono valori univoci per ogni dipartimento.
- c. Ogni insegnamento ha un nome (NOME_I), una descrizione (DESC_I), un codice (CODICE_I), un numero di ore per semestre (CREDITI), un livello (LIVELLO) e un dipartimento che lo offre (DIP_I). Il codice dell'insegnamento è univoco per ogni insegnamento.
- d. Ogni modulo ha un assistente (NOME_A), un semestre (SEMESTRE), un anno (ANNO), un insegnamento (INSEGNAMENTO_MOD) e un codice di modulo (CODICE_MOD). Il codice del modulo permette di distinguere moduli diversi dello stesso insegnamento, tenuti durante lo stesso semestre/anno; i suoi valori sono 1, 2, 3, ..., fino al numero totale di moduli tenuti durante ogni semestre.
- e. Una registrazione di voto fa riferimento a uno studente (SSN), a un modulo specifico e a un voto (VOTO).

Si progetti uno schema di base di dati relazionale per quest'applicazione di basi di dati. Prima di tutto si illustrino tutte le dipendenze funzionali che devono sussistere fra gli attributi. Si progettino quindi schemi di relazione per la base di dati, ciascuno dei quali sia in 3NF o in BCNF. Si specifichino gli attributi chiave di ogni relazione. Si prenda nota di ogni requisito non specificato, e si facciano assunzioni appropriate per rendere completo l'insieme di specifiche.

- 10.18. Si dimostri la correttezza o la falsità delle seguenti regole di inferenza per dipendenze funzionali. Una dimostrazione può essere svolta o tramite un'argomentazione specifica o tramite l'uso delle regole di inferenza da RI1 a RI3. Una confutazione deve essere svolta presentando un'istanza di relazione che soddisfi le condizioni e le dipendenze funzionali a primo membro della regola di inferenza ma non soddisfi le dipendenze a secondo membro.
 - a. $\{W \rightarrow Y, X \rightarrow Z\} \vdash \{WX \rightarrow Y\}$.
 - b. $\{X \rightarrow Y\} \text{ e } Y \supseteq Z \vdash \{X \rightarrow Z\}$.
 - c. $\{X \rightarrow Y, X \rightarrow W, WY \rightarrow Z\} \vdash \{X \rightarrow Z\}$.
 - d. $\{XY \rightarrow Z, Y \rightarrow W\} \vdash \{XW \rightarrow Z\}$.
 - e. $\{X \rightarrow Z, Y \rightarrow Z\} \vdash \{X \rightarrow Y\}$.
 - f. $\{X \rightarrow Y, XY \rightarrow Z\} \vdash \{X \rightarrow Z\}$.
 - g. $\{X \rightarrow Y, Z \rightarrow W\} \vdash \{XZ \rightarrow YW\}$.
 - h. $\{XY \rightarrow Z, Z \rightarrow X\} \vdash \{Z \rightarrow Y\}$.
 - i. $\{X \rightarrow Y, Y \rightarrow Z\} \vdash \{X \rightarrow YZ\}$.
 - j. $\{XY \rightarrow Z, Z \rightarrow W\} \vdash \{X \rightarrow W\}$.
- 10.19. Si considerino i due insiemi seguenti di dipendenze funzionali: $F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$ e $G = \{A \rightarrow CD, E \rightarrow AH\}$. Si controlli se sono equivalenti.
- 10.20. Si consideri lo schema di relazione IMP_DIP di Figura 10.3(a) e il seguente insieme G di dipendenze funzionali su IMP_DIP: $G = \{\text{SSN} \rightarrow \{\text{NOME_I}, \text{DATA_N}, \text{INDIRIZZO}, \text{NUMERO_D}\}, \text{NUMERO_D} \rightarrow \{\text{NOME_D}, \text{SSN_DIR_DIP}\}\}$. Si calcolino le chiusure $(\text{SSN})^+$ e $(\text{NUMERO_D})^+$ rispetto a G .



Capitolo 11

Algoritmi per la progettazione di basi di dati relazionali

Come abbiamo visto nel Capitolo 10, esistono fondamentalmente due approcci per la progettazione di basi di dati relazionali. Il primo consiste nella **progettazione top-down**, una tecnica ampiamente usata per le applicazioni commerciali di basi di dati che prevede la progettazione di uno schema concettuale in un modello di dati di alto livello, come il modello EER, e quindi una sua traduzione in un insieme di relazioni, attraverso procedure quali quelle studiate nel Capitolo 9. Seguendo tale approccio ogni relazione viene analizzata sulla base delle dipendenze funzionali e delle chiavi primarie assegnate, applicando la procedura di normalizzazione descritta nel Paragrafo 10.3 per rimuovere dipendenze parziali e transitive, se ne rimangono. L'analisi di dipendenze indesiderate può anche essere svolta durante la progettazione concettuale, tramite un esame delle dipendenze funzionali presenti fra gli attributi di tipi di entità e tipi di associazione, ovviando così alla necessità di un'ulteriore normalizzazione successiva alla traduzione.

Il secondo approccio consiste nella **progettazione bottom-up**, una tecnica più purista che vede la progettazione di uno schema di base di dati relazionale rigorosamente in termini di dipendenze, funzionali e di altro tipo, specificate sugli attributi della base di dati. Dopo che i progettisti della base di dati hanno specificato le dipendenze, viene applicato un **algoritmo di normalizzazione** per sintetizzare gli schemi di relazione. Ogni singolo schema di relazione dovrebbe raggiungere il livello di qualità associato alla 3NF o alla BCNF, se non a qualche forma normale superiore. In questo Capitolo descriveremo alcuni di questi algoritmi. Descriveremo anche in maggior dettaglio le due proprietà desiderate di join non-additivo (senza perdita) e di conservazione delle dipendenze. Gli algoritmi di normalizzazione cominciano tipicamente sintetizzando uno schema di relazione gigante, detto **relazione universale**, che comprende tutti gli attributi della base di dati. Si eseguono quindi decomposizioni ripetute, basandosi sulle dipendenze funzionali specificate dal progettista di basi di dati, finché non diventino non più possibili o non più desiderabili.

Il Paragrafo 11.1 presenta diversi algoritmi di normalizzazione, basati sulle sole dipen-

denze funzionali, che possono essere usati per sintetizzare schemi in 3NF e in BCNF. Cominceremo col descrivere le due proprietà desiderabili delle decomposizioni – vale a dire la proprietà di conservazione delle dipendenze e la proprietà di join senza perdita (o non-additivo), entrambe usate dagli algoritmi di progettazione per ottenere decomposizioni desiderabili. Mostreremo anche come le forme normali siano *di per sé insufficienti* a garantire una buona progettazione di schemi di basi di dati relazionali: per poter essere considerate frutto di buona progettazione, infatti, le relazioni devono nell'insieme soddisfare queste due ulteriori proprietà.

11.1 Algoritmi per la progettazione di schemi di basi di dati relazionali

11.1.1 Decomposizione delle relazioni e insufficienza delle forme normali

Gli algoritmi di progettazione di basi di dati relazionali qui presentati prendono inizio da un unico **schema di relazione universale** $R = \{A_1, A_2, \dots, A_n\}$ contenente *tutti* gli attributi della base di dati. Si farà implicitamente l'**assunzione di relazione universale**, la quale stabilisce che ogni nome di attributo sia unico. L'insieme F di dipendenze funzionali che deve sussistere sugli attributi di R è specificato dai progettisti della base di dati ed è fornito agli algoritmi di progettazione. Basandosi sulle dipendenze funzionali, gli algoritmi decompongono lo schema di relazione universale R in un insieme di schemi di relazione $D = \{R_1, R_2, \dots, R_m\}$ che diventerà lo schema della base di dati relazionale; D è detto **decomposizione di R** .

È necessario assicurarsi che ogni attributo presente in R sia anche presente in almeno uno schema di relazione R_i nella decomposizione, così che non ci siano attributi “persi”; formalmente si ha

$$\bigcup_{i=1}^m R_i = R$$

Questa condizione è detta **condizione di conservazione degli attributi** di una decomposizione.

Un altro obiettivo è quello che ogni singola relazione R_i nella decomposizione D sia in BCNF (o in 3NF). Questa condizione non è però sufficiente a garantire di per sé una buona progettazione di basi di dati. Occorre infatti considerare la decomposizione nel complesso, oltre a esaminare le singole relazioni. Per illustrare questo punto, si consideri la relazione $\text{IMP_SEDI}(\text{NOME_I}, \text{SEDE_P})$ di Figura 10.5, che è sia in 3NF sia in BCNF. In realtà ogni schema di relazione con due soli attributi è automaticamente in BCNF.¹ Anche se IMP_SEDI è in BCNF, essa dà comunque origine a tuple spurie quando viene unita tramite join con $\text{IMP_PROG1}(\text{SSN}, \text{NUMERO_P}, \text{ORE}, \text{NOME_P}, \text{SEDE_P})$, che non è in BCNF (si veda il ri-

sultato del join naturale in Figura 10.6). IMP_SEDI rappresenta perciò uno schema di relazione particolarmente mal definito, a causa della sua semantica contorta in cui SEDE_P fornisce la sede di *uno dei progetti* sui quali lavora un impiegato. L'unione tramite join di IMP_SEDI con $\text{PROGETTO}(\text{NOME_P}, \text{NUMERO_P}, \text{SEDE_P}, \text{NUM_D})$ di Figura 10.2 – che è in BCNF – dà pure origine a tuple spurie. Si ha quindi bisogno di altri criteri che, insieme alle condizioni di 3NF o BCNF, evitino progetti di cattiva qualità di questo tipo. Nei tre sottoparagrafi seguenti verranno esaminate le condizioni aggiuntive che devono sussistere su una decomposizione D considerata nel suo insieme.

11.1.2 Decomposizione e conservazione delle dipendenze

Sarebbe utile che ogni dipendenza funzionale $X \rightarrow Y$ specificata in F apparisse direttamente in uno degli schemi di relazione R_i della decomposizione D , oppure potesse essere inferita dalle dipendenze presenti in qualche R_i . Informalmente, quella enunciata è la **condizione di conservazione delle dipendenze**: si desidera conservare le dipendenze perché ogni dipendenza in F rappresenta un vincolo sulla base di dati. Se una delle dipendenze non è rappresentata in una singola relazione R_i della decomposizione, non è possibile imporre questo vincolo considerando una sola relazione; occorre piuttosto unire tramite join due o più relazioni della decomposizione e quindi verificare che nel risultato dell'operazione di join sussistano le dipendenze funzionali. Chiaramente questa è una procedura inefficiente e poco pratica.

Non è necessario che le dipendenze specificate in F si presentino esattamente nelle singole relazioni della decomposizione D . È sufficiente che l'unione delle dipendenze che sussistono sulle singole relazioni in D siano equivalenti a F . Si darà ora definizione formale di questi concetti.

Dato un insieme di dipendenze F su R , la **proiezione** di F su R_i , denotata con $\pi_{R_i}(F)$, dove R_i è un sottoinsieme di R ,² è l'insieme di dipendenze $X \rightarrow Y$ di F^+ tali che gli attributi di $X \cup Y$ siano tutti contenuti in R_i . Perciò la proiezione di F su ogni schema di relazione R_i della decomposizione D è l'insieme delle dipendenze funzionali in F^+ , chiusura di F , tali che tutti i loro attributi di parte sinistra e di parte destra siano in R_i . Si dirà che una decomposizione $D = \{R_1, R_2, \dots, R_m\}$ di R **conserva le dipendenze** rispetto a F se l'unione delle proiezioni di F su ogni R_i in D è equivalente a F ; cioè

$$((\pi_{R_1}(F)) \cup \dots \cup (\pi_{R_m}(F)))^+ = F^+$$

Se una decomposizione non conserva le dipendenze, qualche dipendenza viene persa nella decomposizione. Come detto prima, per verificare se una dipendenza persa sussiste comunque, occorre effettuare il JOIN di due o più relazioni presenti nella decomposizione, fino a ottenere una relazione che presenti tutti gli attributi di parte sinistra e di parte destra della dipendenza persa, e quindi verificare se nel risultato del JOIN sussiste la dipendenza – un'operazione poco pratica.

¹ Nella seconda edizione (in lingua inglese) di questo libro, si è usata la notazione $\pi_{R_i}(F)$ (invece che $\pi_{R_i}(F)$) per indicare la proiezione di F su R_i .

Un esempio di decomposizione che non conserva le dipendenze è presentato in Figura 10.12(a), in cui la dipendenza funzionale DF2 è persa quando LOTTI1A viene decomposta in {LOTTI1AX, LOTTI1AY}. Le decomposizioni di Figura 10.11, invece, conservano le dipendenze. Analogamente, per l'esempio in Figura 10.13, indipendentemente da quale delle tre decomposizioni mostrate venga scelta per la relazione INSEGNA (STUDENTE, INSEGNAMENTO, DOCENTE), una o entrambe le dipendenze originariamente presenti vengono perse. Si enuncia qui una proposizione relativa a tale proprietà senza darne la dimostrazione.

Proposizione 1: È sempre possibile trovare una decomposizione D che conserva le dipendenze rispetto a F e tale che ogni relazione R_i di D sia in 3NF.

L'Algoritmo 11.1 crea, a partire da un insieme di dipendenze funzionali F , una decomposizione $D = \{R_1, R_2, \dots, R_m\}$ di una relazione universale R , che conserva le dipendenze e tale che ogni R_i di D sia in 3NF. Esso garantisce solo la proprietà di conservazione delle dipendenze, *non* la proprietà di join senza perdita. Il primo passo dell'algoritmo 11.1 consiste nel trovare una copertura minimale G di F ; per questo passo può essere usato l'Algoritmo 10.2.

ALGORITMO 11.1 Algoritmo di sintesi relazionale con conservazione delle dipendenze.

Input: Una relazione universale R e un insieme di dipendenze funzionali F sugli attributi di R .

1. Si trovi una copertura minimale G per F (si usi l'Algoritmo 10.2);
2. Per ogni parte sinistra X di una dipendenza funzionale presente in G , si crei uno schema di relazione in D con attributi $\{X \cup A_1 \cup A_2 \cup \dots \cup A_k\}$, dove $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_k$ sono le sole dipendenze in G con X come parte sinistra (X è la CHIAVE di questa relazione);
3. Si pongano tutti gli attributi rimanenti (che non sono stati posti in nessuna relazione) in un solo schema di relazione per garantire la proprietà di conservazione degli attributi.

Proposizione 1A: Ogni schema di relazione creato dall'Algoritmo 11.1 è in 3NF (non se ne dà qui una dimostrazione formale;³ la dimostrazione utilizza il fatto che G è un insieme minima di dipendenze).

È ovvio che l'algoritmo conserva tutte le dipendenze presenti in G , perché ogni dipendenza compare in una delle relazioni R_i della decomposizione D . Dal momento che G è equivalente a F , tutte le dipendenze in F sono direttamente conservate nella decomposizione oppure sono derivabili da quelle presenti nelle relazioni risultanti, assicurando così la proprietà di conservazione delle dipendenze. L'Algoritmo 10.1 è detto **algoritmo di sintesi relazionale**, perché ogni schema di relazione R_i nella decomposizione è *sintetizzato* (costruito) a partire dall'insieme di dipendenze funzionali in G con la stessa parte sinistra X .

³ Per una dimostrazione si veda Maier (1983) o Ullman (1982).

11.1.3 Decomposizione e join senza perdita (non-additivo)

Un'altra proprietà che una decomposizione D deve soddisfare è quella di join senza perdita o join non-additivo, che assicura che non vengano generate tuple spurious quando alle relazioni della decomposizione viene applicata un'operazione di JOIN NATURALE. Il problema è già stato illustrato nel Sottoparagrafo 10.1.4 con l'esempio delle Figure 10.5 e 10.6. Dato che si tratta della proprietà di una decomposizione di *schemi* di relazione, la condizione di assenza di tuple spurious deve sussistere per *ogni stato valido di relazione* – cioè per ogni stato di relazione che soddisfa le dipendenze funzionali in F . La proprietà di join senza perdita è perciò sempre definita rispetto a un insieme specifico F di dipendenze. Formalmente, una decomposizione $D = \{R_1, R_2, \dots, R_m\}$ di R soddisfa la proprietà di join senza perdita (non-additivo) rispetto all'insieme di dipendenze F di R se, per *ogni* stato di relazione r di R che soddisfa F , sussiste quanto segue, dove $*$ è il JOIN NATURALE di tutte le relazioni in D :

$$*(\pi_{R_1}(r), \dots, \pi_{R_m}(r)) = r$$

La parola perdita in *senza perdita* si riferisce a *perdita di informazione*, non a perdita di tuple. Se una decomposizione non soddisfa la proprietà di join senza perdita, è possibile che si presentino tuple spurious aggiuntive, dopo che sono state eseguite le operazioni di PROIEZIONE (π) e di JOIN NATURALE (*); queste tuple aggiuntive rappresentano un'informazione errata. Si preferisce qui la locuzione **join non-additivo** perché essa descrive più fedelmente la situazione; se per una decomposizione sussiste questa proprietà è sicuro che, dopo che sono state eseguite le operazioni di PROIEZIONE e JOIN NATURALE, non verranno *aggiunte* al risultato tuple spurious, che portano informazione sbagliata.

Ovviamente la decomposizione di IMP_PROG(SSN, NUMERO_P, ORE, NOME_I, NOME_P, SEDE_P), di Figura 10.3, in IMP_SEDI(NOME_I, SEDE_P) e IMP_PROG1(SSN, NUMERO_P, ORE, NOME_P, SEDE_P), presente in Figura 10.5, non soddisfa la proprietà di join senza perdita, come illustrato in Figura 10.6. Per verificare se una data decomposizione D soddisfa la proprietà di join senza perdita, rispetto a un insieme F di dipendenze funzionali, si può usare l'Algoritmo 11.2.

ALGORITMO 11.2 Verifica della proprietà di join senza perdita (non-additivo).

Input: Una relazione universale R , una decomposizione $D = \{R_1, R_2, \dots, R_m\}$ di R , e un insieme F di dipendenze funzionali.

1. Si costruisca una matrice iniziale S con una riga i per ogni relazione R_i di D , e una colonna j per ogni attributo A_j di R .
2. Si ponga $S(i, j) := B_{ij}$ per ogni elemento della matrice.
(* ogni B_{ij} è un simbolo distinto associato agli indici (i, j) *)
3. Per ogni riga i che rappresenta lo schema di relazione R_i (per ogni colonna j che rappresenta l'attributo A_j , se la relazione R_i contiene l'attributo A_j) allora si ponga $S(i, j) := A_j$;});
(* ogni A_j è un simbolo distinto associato all'indice (j) *)

- ..., $X \rightarrow A_k$ sono le sole dipendenze di G con X come parte sinistra (X è la CHIAVE di questa relazione).
- 3 Se nessuno degli schemi di relazione in D contiene una chiave di R , allora si costruisca un altro schema di relazione in D che contenga attributi che formano una chiave di R .

Si può dimostrare che la decomposizione formata dall'insieme di schemi di relazione costruiti dall'algoritmo precedente conserva le dipendenze e gode della proprietà di join senza perdita. Inoltre ogni schema di relazione nella decomposizione è in 3NF. Questo algoritmo è un miglioramento dell'Algoritmo 11.1, dato che quest'ultimo garantisce solo la conservazione delle dipendenze.⁴

Il passo 3 dell'Algoritmo 11.4 prevede l'individuazione di una chiave K di R . Per individuare una chiave K di R sulla base dell'insieme dato F di dipendenze funzionali può essere usato l'Algoritmo 11.4a. Si comincia ponendo K uguale a tutti gli attributi di R ; quindi si rimuove un attributo alla volta e si verifica se i restanti attributi formano ancora una superchiave. Si noti che l'insieme di dipendenze funzionali usato per determinare una chiave nell'Algoritmo 11.4a può essere F o G , dal momento che essi sono equivalenti. Si noti, inoltre, che l'Algoritmo 11.4a determina solo *una chiave* fra le possibili chiavi candidate di R ; la chiave fornita dipende dall'ordine con cui gli attributi sono rimossi da R al passo 2.

ALGORITMO 11.4a Ricerca di una chiave K per lo schema di relazione R sulla base di un insieme F di dipendenze funzionali.

1. Si ponga $K := R$.
2. Per ogni attributo A in K
 - {si calcoli $(K - A)^*$ rispetto a F ;
 - Se $(K - A)^*$ contiene tutti gli attributi di R , allora si ponga $K := K - \{A\}$ };

Non è sempre possibile trovare una decomposizione in schemi di relazione che conservi le dipendenze e che consenta a ogni schema di relazione della decomposizione di essere in BCNF (anziché in 3NF come nell'Algoritmo 11.4). Si può pensare di controllare gli schemi di relazione in 3NF della decomposizione uno a uno per vedere se soddisfano la BCNF. Se qualche schema di relazione R_i non è in BCNF si può scegliere di decomporlo ulteriormente o di lasciarlo così com'è in 3NF (con qualche possibile anomalia di aggiornamento). Il fatto che non si possa sempre trovare una decomposizione in schemi di relazione in BCNF che conservi le dipendenze può essere evidenziato dagli esempi in Figura 10.12. Le relazioni LOTTI1A (Figura 10.12a) e INSEGNA (Figura 10.13) non sono in BCNF, ma sono in 3NF. Ogni tentativo di decomporre ulteriormente queste relazioni in relazioni in BCNF ha come risultato una perdita della dipendenza DF2: $\{\text{NOME_CONTEA}, \#LOTTO\} \rightarrow \{\#\text{ID_PROPRIETÀ}, \text{AREA}\}$ in LOTTI1A o una perdita di DF1: $\{\text{STUDENTE}, \text{INSEGNAMENTO}\} \rightarrow \text{DOCENTE}$ in INSEGNA.

È importante sottolineare che la teoria delle decomposizioni con join senza perdita si basa sull'assunzione che *non sono consentiti valori nulli per gli attributi di join*. Il prossimo Paragrafo affronta alcuni problemi che nelle decomposizioni relazionali possono essere causati dai valori nulli.

11.1.4 Problemi con valori nulli e tuple dangling

Quando si progetta uno schema di base di dati relazionale bisogna valutare attentamente i problemi associati alla presenza di valori nulli. A tutt'oggi non è stata presentata nessuna teoria di progettazione relazionale pienamente soddisfacente che comprenda i valori nulli. Un problema si verifica quando alcune tuple presentano valori nulli per attributi che saranno usati per effettuare il JOIN di singole relazioni della decomposizione. Per illustrare ciò si consideri la base di dati mostrata in Figura 11.2(a), in cui sono presenti le due relazioni IMPIEGATO e DIPARTIMENTO. Le ultime due tuple impiegato – Berger e Benitez – rappresentano impiegati appena assunti che non sono stati ancora assegnati a un dipartimento (si dia per scontato che ciò non violi alcun vincolo di integrità). Si supponga ora di voler recuperare un elenco di valori di $(\text{NOME_I}, \text{NOME_D})$ per tutti gli impiegati. Se si esegue l'operazione di JOIN NATURALE su IMPIEGATO e DIPARTIMENTO (Figura 11.2b), le due tuple *non* appariranno nel risultato. Con l'operazione di JOIN ESTERNO, studiata nel Capitolo 7, si può affrontare questo problema. Si ricordi che, se si considera il JOIN ESTERNO SINISTRO di IMPIEGATO con DIPARTIMENTO, le tuple di IMPIEGATO che presentano un valore nullo per l'attributo di join appariranno comunque nel risultato, congiunte (*joined*) con una tupla "immaginaria" di DIPARTIMENTO che presenta valori nulli per tutti i suoi attributi. La Figura 11.2(c) mostra il risultato.

In generale, ogni volta che viene progettato uno schema di base di dati relazionale in cui due o più relazioni sono collegate tramite chiavi esterne, deve essere dedicata particolare attenzione a ricercare potenziali valori nulli nelle chiavi esterne. Ciò può infatti causare un'inaspettata perdita d'informazione nelle interrogazioni che prevedono join su quella chiave esterna. Inoltre, se si presentano valori nulli in altri attributi, come STIPENDIO, il loro effetto su funzioni built-in (integrate) come SUM e AVERAGE deve essere attentamente valutato.

Un problema collegato è quello delle **tuple dangling** (dondolanti), che possono presentarsi se si spinge troppo oltre una decomposizione. Si supponga di decomporre ulteriormente la relazione IMPIEGATO di Figura 11.2(a) nelle relazioni IMPIEGATO_1 e IMPIEGATO_2, mostrate in Figura 11.3(a) e 11.3(b).⁵ Se si applica l'operazione di JOIN NATURALE a IMPIEGATO_1 e IMPIEGATO_2, si ottiene la relazione IMPIEGATO originale. Si può però usare la rappresentazione alternativa mostrata in Figura 11.3(c), in cui *non si inserisce una tupla* in IMPIEGATO_3 se all'impiegato non è stato assegnato un dipartimento (anziché inserire una tupla con un valore nullo per NUM_D come in IMPIEGATO_2). Se si usa IMPIEGATO_3 anziché IMPIEGATO_2 e si esegue un JOIN NATURALE su IMPIEGATO_1 e IMPIEGATO_3, le tuple re-

⁴ Il passo 3 dell'Algoritmo 11.1 non è necessario nell'Algoritmo 11.4 per conservare gli attributi, dato che la chiave comprendrà tutti gli attributi non ancora considerati; questi sono gli attributi che non partecipano a nessuna dipendenza funzionale.

⁵ Ciò accade talora quando si applica una frammentazione verticale a una relazione nel contesto di una base di dati distribuita.

(a) **IMPIEGATO**

NOME_I	SSN	DATA_N	INDIRIZZO	NUM_D
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	null
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	null

DIPARTIMENTO

NOME_D	NUM_D	SSN_DIR_DIP
Ricerca	5	333445555
Amministrazione	4	987654321
Sede centrale	1	888665555

(b)

NOME_I	SSN	DATA_N	INDIRIZZO	NUM_D	NOME_D	SSN_DIR_DIP
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Ricerca	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Ricerca	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Amministrazione	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Amministrazione	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Ricerca	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Ricerca	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Amministrazione	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Sede centrale	888665555

(c)

NOME_I	SSN	DATA_N	INDIRIZZO	NUM_D	NOME_D	SSN_DIR_DIP
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Ricerca	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Ricerca	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Amministrazione	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Amministrazione	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Ricerca	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Ricerca	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Amministrazione	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Sede centrale	888665555
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	null	null	null
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	null	null	null

Figura 11.2 Illustrazione dei problemi relativi al join con valori nulli. (a) Una base di dati con valori nulli per alcuni attributi di join. (b) Risultato dell'applicazione del JOIN NATURALE alle relazioni IMPIEGATO e DIPARTIMENTO. (c) Risultato dell'applicazione del JOIN ESTERNO a IMPIEGATO e DIPARTIMENTO.

lative a Berger e Benitez non saranno presenti nel risultato; queste sono dette *tuple dangling* perché sono rappresentate in una sola delle due relazioni che rappresentano impiegati e pertanto vengono perse se si esegue un'operazione di join (interno).⁶

⁶ Il termine "dangling" viene di solito tradotto con "dondolanti"; più precisamente si dovrebbe parlare di tuple che rimangono senza un collegamento ad altri dati. (N.d.T.)

(a) **IMPIEGATO_1**

NOME_I	SSN	DATA_N	INDIRIZZO
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX

(b) **IMPIEGATO_2**

SSN	NUM_D
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	null
888664444	null

(c) **IMPIEGATO_3**

SSN	NUM_D
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

Figura 11.3 Il problema della "tuple dangling". (a) La relazione IMPIEGATO_1 (comprende tutti gli attributi di IMPIEGATO a eccezione di NUMERO_D). (b) La relazione IMPIEGATO_2 (comprende l'attributo NUMERO_D con valori nulli). (c) La relazione IMPIEGATO_3 (comprende l'attributo NUMERO_D ma non comprende tuple per le quali NUMERO_D assume valori nulli).

11.1.5 Analisi degli algoritmi di normalizzazione

Uno dei problemi con gli algoritmi di normalizzazione descritti è che il progettista della base di dati deve prima di tutto specificare *tutte* le dipendenze funzionali rilevanti fra gli attributi della base di dati. Questo non è un compito agevole per una grande base di dati con centinaia di attributi. Una dimenticanza nello specificare una o due dipendenze importanti può avere come risultato un progetto inadeguato. Un altro problema è che questi algoritmi non sono in generale deterministici. Ad esempio, gli *algoritmi di sintesi* (Algoritmi 11.1 e 11.4) richiedono la specificazione di una copertura minimale G per l'insieme di dipendenze funzionali F . Dato che in generale ci possono essere molte coperture minimali corrispondenti a F , l'algoritmo può fornire diversi progetti a seconda della specifica copertura minimale usata. Alcuni di questi progetti possono non essere desiderabili. L'*algoritmo di decomposizione* (Algoritmo 11.3) dipende dall'ordine con cui le dipendenze funzionali sono fornite all'algoritmo; anche qui è possibile che si derivino molti progetti diversi relativi allo stesso insieme di dipendenze funzionali, a seconda dell'ordine con cui queste dipendenze sono considerate per la violazione della BCNF. E ancora, alcuni progetti possono essere significativamente superiori mentre altri possono essere inadeguati.

Sommario

In questo capitolo abbiamo presentato diversi algoritmi di normalizzazione. Gli *algoritmi di sintesi relazionale* costruiscono relazioni in 3NF, a partire da uno schema di relazione universale, sulla base di un insieme dato di dipendenze funzionali specificato dal progettista della base di dati. Gli algoritmi di decomposizione relazionale creano relazioni in BCNF tramite successive decomposizioni senza perdita di relazioni non normalizzate in una coppia di relazioni componenti. Abbiamo esaminato due importanti proprietà delle decomposizioni: la proprietà di join senza perdita (non-additivo) e la proprietà di conservazione delle dipendenze. Si è quindi descritto un algoritmo per effettuare il test di decomposizione senza perdita, e un più semplice test per verificare l'assenza di perdita di decomposizioni binarie. Abbiamo visto che è possibile sintetizzare schemi di relazione in 3NF che soddisfano entrambe le proprietà sudette; invece, nel caso di BCNF, è possibile aspirare solo all'assenza di perdita; la conservazione delle dipendenze *non* è necessariamente assicurata.

Questionario di verifica

- 11.1. Cosa si intende per condizione di conservazione degli attributi in una decomposizione?
- 11.2. Perché le forme normali da sole non sono sufficienti come condizione di buona progettazione di schemi?
- 11.3. Cos'è la proprietà di conservazione delle dipendenze per una decomposizione? Perché è importante?
- 11.4. Perché non è possibile garantire che gli schemi di relazione in BCNF vengano prodotti da decomposizioni che conservano le dipendenze di schemi di relazione non-BCNF? Si fornisca un controesempio per illustrare questo punto.
- 11.5. Cos'è la proprietà di join senza perdita (o non-additivo) di una decomposizione? Perché è importante?
- 11.6. Tra le due proprietà di conservazione delle dipendenze e assenza di perdita, quale deve essere assolutamente soddisfatta? Perché?
- 11.7. Si discutano i problemi relativi ai valori nulli e alle tuple dangling.

Esercizi

- 11.8. Si dimostri che gli schemi di relazione prodotti dall'Algoritmo 11.1 sono in 3NF.
- 11.9. Si dimostri che, se la matrice S risultante dall'Algoritmo 11.2 non ha una riga con tutti simboli "a", la proiezione di S sulla decomposizione e la successiva riunione tramite join produrrà sempre almeno una tupla spuria.

- 11.11. Si dimostri che gli schemi di relazione prodotti dall'Algoritmo 11.4 sono in 3NF.
- 11.12. Si consideri l'esempio di normalizzazione della relazione LOTTI del Paragrafo 10.4. Si determini se la decomposizione di LOTTI in {LOTTI1AX, LOTTI1AY, LOTTI1B, LOTTI2} gode della proprietà di join senza perdita, applicando l'Algoritmo 11.2 e anche usando il test presente nella proprietà LJ1.
- 11.13. Si applichi l'Algoritmo 11.4a alla relazione dell'Esercizio 10.26 per determinare una chiave per R . Si costruisca un insieme minimale G di dipendenze equivalenti a F , e si applichi l'algoritmo di sintesi (Algoritmo 11.4) per decomporre R in relazioni in 3NF.
- 11.14. Si ripeta l'Esercizio 11.13 per le dipendenze funzionali dell'Esercizio 10.27.
- 11.15. Si applichi l'algoritmo di decomposizione (Algoritmo 11.3) alla relazione R e all'insieme di dipendenze F dell'Esercizio 10.26. Si ripeta il tutto per le dipendenze G dell'Esercizio 10.27.
- 11.16. Si applichi l'Algoritmo 11.4a alle relazioni presenti negli Esercizi 10.29 e 10.30 per determinare una chiave per R . Si applichi l'algoritmo di sintesi (Algoritmo 11.4) per decomporre R in relazioni in 3NF e l'algoritmo di decomposizione (Algoritmo 11.3) per decomporre R in relazioni in BCNF.
- 11.17. Si scrivano programmi che implementano gli Algoritmi 11.3 e 11.4.
- 11.18. Si considerino le decomposizioni seguenti per lo schema di relazione R dell'esercizio 10.26. Si determini se ogni decomposizione gode (i) della proprietà di conservazione delle dipendenze e (ii) della proprietà di join senza perdita, rispetto a F . Si determini anche in quale forma normale si trova ogni relazione della decomposizione.
 - a. $D_1 = \{R_1, R_2, R_3, R_4, R_5\}; R_1 = \{A, B, C\}, R_2 = \{A, D, E\}, R_3 = \{B, F\}, R_4 = \{F, G, H\}, R_5 = \{D, I, J\}$.
 - b. $D_2 = \{R_1, R_2, R_3\}; R_1 = \{A, B, C, D, E\}, R_2 = \{B, F, G, H\}, R_3 = \{D, I, J\}$.
 - c. $D_3 = \{R_1, R_2, R_3, R_4, R_5\}; R_1 = \{A, B, C, D\}, R_2 = \{D, E\}, R_3 = \{B, F\}, R_4 = \{F, G, H\}, R_5 = \{D, I, J\}$.
- 11.19. Si consideri la seguente relazione FRIGORIFERO(#Modello, Anno, Prezzo, Stab_Prod, Colore), abbreviata con FRIGORIFERO(M, A, P, SP, C) e che ha il seguente insieme F di dipendenze funzionali: $F = \{M \rightarrow SP, \{M, A\} \rightarrow P, SP \rightarrow C\}$.
 - a. Si valuti ciascuno dei seguenti insiemi come possibile chiave candidata per FRIGORIFERO, giustificando perché può essere o non può essere una chiave: {M}, {M, A}, {M, C}.
 - b. Sulla base di questa determinazione della chiave, si stabilisca se la relazione FRIGORIFERO è in 3NF e in BCNF, fornendone le opportune giustificazioni.
 - c. Si consideri la decomposizione di FRIGORIFERO in $D = \{R1(M, A, P), R2(M, SP, C)\}$. Questa decomposizione è senza perdita? Si dica perché. (Si può consultare il test presente nella proprietà LJ1 del Sottoparagrafo 11.1.3).

Bibliografia selezionata

I libri di Maier (1983) e Atzeni e De Antonellis (1993) contengono un'esauriente trattazione della teoria della dipendenza relazionale. L'algoritmo di decomposizione (Algoritmo 11.1) è

dovuto a Bernstein (1976). L'Algoritmo 11.4 si basa sull'algoritmo di normalizzazione presentato in Biskup e altri (1979). In Tsou e Fischer (1982) è fornito un algoritmo polinomiale per una decomposizione in BCNF.

La teoria della conservazione delle dipendenze e dei join senza perdita è fornita in Ullman (1988), dove sono presenti prove per alcuni degli algoritmi qui discussi. La proprietà di join senza perdita è analizzata in Aho e altri (1979). Algoritmi per determinare le chiavi di una relazione a partire dalle dipendenze funzionali sono forniti in Osborn (1976); il test per la BCNF è analizzato in Osborn (1979). Il test per la 3NF è trattato in Tsou e Fischer (1982). Algoritmi per progettare relazioni in BCNF sono forniti in Wang (1990) e Hernandez e Chan (1991).

In Abiteboul e altri (1995) è presente una trattazione teorica di molte delle idee illustrate in questo capitolo e nel Capitolo 10.

Capitolo 12

Esempi di sistemi di gestione di basi di dati relazionali: Oracle e Microsoft Access

In questo capitolo analizzeremo l'implementazione del modello relazionale di dati in alcuni sistemi commerciali. Poiché la famiglia dei sistemi relazionali di gestione di basi dati (RDBMS: relational database management system) comprende moltissimi prodotti, non è possibile in questa sede confrontare o valutare le caratteristiche di tutti; si esamineranno in dettaglio, invece, due sistemi particolarmente rappresentativi: Oracle, che fa parte dei prodotti di fascia alta che ebbero origine dai computer mainframe, e Microsoft Access, che si rivolge all'utente di personal computer. L'obiettivo è mostrare come questi prodotti pur offrendo funzionalità e caratteristiche relativamente simili, siano caratterizzati da architetture e interfacce molto diverse.

Nel Paragrafo 12.1 forniremo una panoramica storica dello sviluppo del RDBMS, mentre dal Paragrafo 12.2 al Paragrafo 12.5 descriveremo il RDBMS Oracle: l'architettura e le funzioni principali del sistema (Paragrafo 12.2); la modellazione a oggetti dei dati, i linguaggi di programmazione e i trigger (Paragrafo 12.3); come Oracle organizza il suo spazio di memorizzazione interno (Paragrafo 12.4); alcuni esempi di programmazione in ambiente Oracle (Paragrafo 12.5); strumenti disponibili in Oracle per la progettazione delle basi di dati e lo sviluppo delle applicazioni (Paragrafo 12.6).

Il pacchetto Microsoft Access attualmente viene fornito insieme agli altri prodotti di Office 2000 e può essere usato su macchine Windows 95/98, Windows NT e Windows 2000. Nel Paragrafo 12.7 daremo una panoramica su Microsoft Access comprendente le tecniche di definizione e gestione dei dati, nonché le funzioni grafiche interattive per facilitare le interrogazioni; nel Paragrafo 12.8 presenteremo un riassunto delle caratteristiche e delle funzionalità di Access relative alle maschere, ai report e alle macro e considereremo brevemente alcune funzioni aggiuntive del prodotto Microsoft.

12.1 Sistemi di gestione di basi di dati relazionali: una prospettiva storica

L'introduzione del modello relazionale nel 1970 segnò l'inizio di un'intensa attività di ricerca e sperimentazione sulle idee relazionali. I principali sforzi di ricerca furono iniziati presso il Centro di Ricerche di San Josè (ora chiamato Almaden) dell'IBM. Negli anni ottanta si arrivò così all'annuncio di due prodotti DBMS relazionali dell'IBM: SQL/DS per ambienti DOS/VSE (disk operating system/virtual storage extended) e VM/CMS (virtual machine/conversational monitoring system), introdotti nel 1981, e DB2 per il sistema operativo MVS, introdotto nel 1983. Un altro DBMS relazionale, INGRES, fu sviluppato presso l'Università della California, Berkeley, nei primi anni settanta e messo in commercio dalla Relational Technology alla fine di quel decennio. Da prototipo di ricerca, INGRES diventò in breve un RDBMS commerciale per opera della Ingres, una consociata della ASK; attualmente viene venduto da Computer Associates. Altri notissimi RDBMS commerciali sono: Oracle della Oracle; Sybase della Sybase; RDB della Digital Equipment, ora di proprietà della Compaq; INFORMIX della Informix e UNIFY della Unify.

Oltre ai RDBMS appena menzionati negli anni ottanta apparvero molte implementazioni del modello di dati relazionale per personal computer. Tra queste vi sono RIM, RBASE 5000, PARADOX, OS/2 Database Manager, DBase IV, XDB, WATCOM SQL, il Server SQL della Sybase, il Server SQL della Microsoft e, più recentemente, Access (anch'esso della Microsoft). Inizialmente si trattava di sistemi monoutente; ora, però, i produttori hanno iniziato a offrire su personal computer l'architettura delle basi di dati client/server rendendo i loro RDBMS compatibili con lo standard *ODBC* (*Open Database Connectivity*) che permette di eseguire interrogazioni SQL a questi sistemi.

Anche se i prodotti per personal computer hanno subito una notevole evoluzione, occorre notare che il termine *relazionale* viene ancora usato da molti fornitori in modo inappropriate. Perché una base di dati possa essere definita relazionale, deve avere almeno le seguenti proprietà:¹

- memorizzare i dati come relazioni in modo che ciascuna colonna sia identificata indipendentemente dal suo nome e l'ordine delle righe sia irrilevante;
- le operazioni disponibili per l'utente così come quelle usate internamente dal sistema sono vere operazioni relazionali in grado di creare nuove relazioni partendo da quelle esistenti;
- il sistema deve supportare almeno una variante dell'operazione JOIN.

Anche se sarebbe possibile aggiungere altri elementi all'elenco appena fornito, i criteri appena considerati sono quelli minimi per controllare se un sistema è relazionale: è facile notare come alcuni dei DBMS presentati commercialmente come relazionali non li soddisfino affatto.

12.2 Struttura base del sistema Oracle

Tradizionalmente i fornitori di RDBMS hanno scelto di usare una terminologia specifica per la documentazione dei loro prodotti. L'organizzazione del sistema Oracle verrà descritta cercando di mettere in relazione questa terminologia con quella di uso più generale. È interessante vedere come i vendori di RDBMS hanno progettato pacchetti software che seguono fondamentalmente il modello relazionale, ma offrono anche diverse funzioni aggiuntive per eseguire il progetto e l'implementazione delle basi di dati di grandi dimensioni e delle applicazioni che ne fanno uso.

Un server Oracle consiste in una **base di dati Oracle**, contenente i dati memorizzati (compresi i file di registro o log file e di controllo), e nell'**istanza di Oracle**, cioè i processi esecutivi. Questi ultimi si suddividono nei processi di Oracle (o processi di sistema) e quelli degli utenti, relativi a una specifica istanza della base di dati. Il server Oracle usa il linguaggio SQL per definire e gestire i dati. Dispone, inoltre, di un linguaggio procedurale, chiamato PL/SQL, per controllare il flusso esecutivo di SQL, per trasferire i risultati delle interrogazioni in variabili e per gestire eventuali errori. A Oracle si può accedere anche attraverso linguaggi di programmazione di uso generale come C o Java.

12.2.1 Struttura della base di dati

La base di dati Oracle ha due strutture principali: (1) una *struttura fisica* che si riferisce ai dati effettivamente memorizzati e (2) una *struttura logica* che corrisponde alla rappresentazione astratta dei dati. Quest'ultima è approssimativamente equivalente allo schema concettuale della base di dati.² La base di dati contiene i seguenti tipi di file:

- uno o più *file di dati* contenenti i dati effettivi;
- due o più file di registro chiamati *file di registro delle azioni di ripristino* (*redo log files*) che registrano tutti i cambiamenti fatti ai dati e sono usati nel processo di ripristino della base di dati nel caso di modifiche che non devono essere scritte nella memoria permanente, ad esempio per il fallimento di una transazione;
- uno o più *file di controllo* (*control files*) contenenti informazioni di controllo come il nome della base di dati, i nomi e le localizzazioni degli altri file e un timestamp di creazione delle basi di dati;
- i *file di traccia* (*trace files*) e un *registro di allarme* (*alert log*); i processi del server hanno un file di traccia che ne registra le azioni, mentre il registro di allarme contiene i principali eventi relativi alla base di dati.

Sia il file di registro sia i file di controllo possono essere mantenuti in copie multiple, scrivendone più copie su dispositivi multipli.

¹ Codd (1985) indica 12 regole per determinare se un DBMS è relazionale. Codd (1990) si occupa di modelli relazionali estesi, identificando più di trecentotrenta funzioni dei sistemi relazionali, suddivise in diciotto categorie.

² In questo paragrafo vengono utilizzati alcuni termini che non sono ancora stati definiti formalmente. Sono tuttavia essenziali per una discussione completa sull'architettura di Oracle.

- *Punto di controllo (CKPT: checkpoint)*. Si tratta di un evento che fa sì che tutti i buffer modificati dopo l'ultimo processo di CKPT e presenti nell'area SGA vengano trascritti nei file di dati. Il processo CKPT si coordina con il processo DBWR per stabilire il momento opportuno per fissare un punto di controllo.
- *Monitor di sistema (SMON: system monitor)*. Esegue il ripristino dell'istanza della base di dati, gestisce le aree di memoria andando a occupare aree di memoria contigue condivise e recupera le transazioni eventualmente tralasciate durante le operazioni di ripristino.
- *Monitor di processo (PMON: process monitor)*. Esegue il recupero di un processo utente quando si interrompe a causa di un errore. È responsabile anche della gestione della cache e delle altre risorse usate da un processo utente.
- *Memorizzatore (ARCH: archiver)*. Archivia periodicamente i file di registro in linea nella memoria di secondo livello (ad esempio nastro magnetico) se configurato per farlo.
- *Processo di ripristino (RECO: recover process)*. Risolve le transazioni distribuite che sono in attesa a causa di un malfunzionamento di rete o di sistema in una base dati distribuita.
- *Smistatori (Dnnn: dispatcher)*. Nelle configurazioni a flussi esecutivi multipli, smista le richieste d'instradamento provenienti dai processi utente ai processi server condivisi disponibili. Vi è un processo smistatore per ciascun protocollo standard di comunicazione.
- *Processi di blocco (LCKn: lock process)*. Assicurano il blocco degli accessi alle istanze quando Oracle viene eseguito in modalità server parallela.

12.2.3 Avvio

Una base di dati Oracle non è disponibile per gli utenti finché il server Oracle non è stato avviato e la base di dati aperta. Per attivare una base di dati e renderla disponibile agli utenti sono necessarie le operazioni seguenti.

1. *Avvio di un'istanza della base di dati*. In questa fase viene allocata l'area SGA e sono creati i processi di background. Per gestire l'inizializzazione dell'istanza viene preparato un file di parametri che contiene dati di controllo, fra cui le dimensioni dell'area SGA, il nome della base di dati alla quale l'istanza può connettersi e così via.
2. *Montaggio (mounting) di una base di dati*. Questa fase associa una base di dati a un'istanza di Oracle appena avviata. Prima che quest'operazione venga eseguita, l'istanza di Oracle è disponibile solo per gli amministratori. Il montaggio è un'operazione uno a molti, nel senso che più istanze di Oracle possono montare contemporaneamente la medesima base di dati. È compito dell'amministratore della base di dati scegliere se attivarla in modalità parallela o esclusiva. Quando un'istanza di Oracle monta una base di dati in modalità esclusiva, solo quell'istanza può accedere alla base di dati. D'altra parte se l'istanza è attivata in modalità parallela o condivisa, anche altre istanze che sono avviate in modalità parallela possono montare la base di dati.
3. *Apertura di una base di dati*. Si tratta di un'attività di amministrazione della base di dati. L'apertura di una base di dati montata, la rende disponibile per le normali operazioni di gestione rendendo accessibili i file di dati in linea e i file di registro.

Ovviamenete vi è una serie di operazioni opposte per chiudere un'istanza di Oracle:

1. chiusura della base di dati;
2. smontaggio (*dismount*) della base di dati;
3. chiusura (*shut down*) dell'istanza di Oracle.

Il file di parametri che regola la creazione di un'istanza di Oracle contiene:

- parametri che assegnano nomi a elementi (ad esempio il nome della base di dati, il nome e la localizzazione dei file di controllo della base di dati, i nomi di segmenti privati per il ripristino o rollback);
- parametri dimensionali (ad esempio le dimensioni massime possibili per l'area SGA e le dimensioni massime dei buffer);
- parametri che influenzano il carico, denominati variabili (ad esempio il parametro DB_BLOCK_BUFFERS che imposta il numero di blocchi di dati da allocare nell'area SGA).

L'amministratore della base di dati può modificare tali parametri durante le procedure di gestione e di controllo della base di dati.

12.3 Struttura della base di dati e sua manipolazione in Oracle

In origine Oracle fu progettato come sistema di gestione di basi di dati relazionali. Dalla versione 8 del prodotto, Oracle è stato trasformato in un sistema di gestione di basi di dati a oggetti relazionale (ORDBMS: object relational database management system). Nel seguito si darà una visione d'insieme di Oracle, comprese le sue funzionalità di modellazione relazionale e a oggetti relazionale. Le principali differenze tra Oracle 8 e le versioni precedenti sono descritte nel Paragrafo 12.6.

12.3.1 Oggetti dello schema

In Oracle il termine *schema* si riferisce a una serie di oggetti che definiscono dati. Gli oggetti dello schema sono singoli elementi che descrivono tavole, viste e così via. Vi è, quindi, una netta distinzione tra questi oggetti dello schema logico e i componenti di memorizzazione fisica chiamati *spazi delle tavole*. Qui di seguito si presentano gli oggetti dello schema supportati da Oracle; è da notare che per denominarli Oracle utilizza una propria terminologia che si aggiunge alle definizioni dei concetti di base del modello relazionale.

- *Tabelle*: unità base conformi al modello relazionale presentato nei Capitoli 7 e 8. Ogni colonna (attributo) ha un nome, un tipo di dati e una grandezza (che dipende dal tipo e dalla precisione).
- *Viste* (si veda anche il Capitolo 8): tavole virtuali che possono essere definite partendo da tavole base o da altre viste. Se la chiave di una vista di join (una vista la cui interroga-

Oltre alle informazioni di dizionario appena descritte, Oracle controlla costantemente l'attività della base di dati e la registra in tabelle chiamate **tabelle dinamiche delle prestazioni**. L'amministratore della base di dati ha accesso ad esse per controllare le prestazioni del sistema e può concedere ad alcuni utenti l'accesso a tali viste.

12.3.3 SQL in Oracle

Il linguaggio SQL implementato in Oracle è compatibile con lo standard SQL ANSI/ISO. Fornisce, quindi, le funzionalità SQL descritte nel Capitolo 8, ma con alcune varianti. Tutte le operazioni su una base di dati Oracle sono eseguite usando **istruzioni SQL**, quindi stringhe del linguaggio SQL che vengono fornite al server Oracle per l'esecuzione. Si fa riferimento a una interrogazione completa SQL come a una **frase SQL (SQL sentence)**. Sono gestite le seguenti istruzioni SQL (Capitolo 8):

- **istruzioni DDL:** definiscono gli oggetti dello schema trattati nel Sottoparagrafo 12.2.1 e permettono di fornire e revocare privilegi di accesso;
- **istruzioni DML:** specificano operazioni di interrogazione, inserimento, cancellazione e aggiornamento. Fa parte delle operazioni DML anche il blocco di una tabella o di una vista per regolare gli accessi a dati condivisi oppure l'esame del piano di esecuzione di un'interrogazione;
- **istruzioni di controllo di transazione:** specificano unità atomiche di elaborazione sui dati della base di dati. Una **transazione** è un'unità atomica di elaborazione che inizia con un'istruzione eseguibile e finisce quando viene eseguito il *commit* (i cambiamenti sono trascritti su memoria permanente) o il *rollback* (i cambiamenti sono stati annullati). Le istruzioni di controllo delle transazioni in SQL, sono COMMIT (WORK), SAVEPOINT e ROLLBACK;
- **istruzioni di controllo delle sessioni:** consentono agli utenti di controllare le proprietà della loro sessione, se necessario abilitando o disabilitando alcuni ruoli o cambiando le impostazioni di linguaggio (ad esempio ALTER SESSION, CREATE ROLE);
- **istruzioni di controllo del sistema:** consentono all'amministratore di cambiare impostazioni di base, come il numero minimo di server condivisi, o di eliminare una sessione. L'unica istruzione di questo tipo è ALTER SYSTEM;
- **istruzioni SQL encapsulate:** le istruzioni SQL possono essere incapsulate in un linguaggio di programmazione procedurale come PL/SQL di Oracle o il linguaggio C. Per elaborare le istruzioni SQL incapsulate in un programma C, Oracle usa un apposito precompilatore. Queste istruzioni comprendono operazioni di gestione del cursore come OPEN, FETCH, CLOSE e altre come EXECUTE.

Il linguaggio PL/SQL è un'estensione del linguaggio procedurale di Oracle che aggiunge funzionalità procedurali a SQL. Compilando e salvando il codice PL/SQL in una base di dati Oracle come procedura memorizzata, il traffico di rete tra le applicazioni e la base di dati è ridotto con un evidente guadagno in termini di prestazioni. I blocchi PL/SQL possono anche essere trasmessi da un'applicazione a una base di dati Oracle per eseguire localmente operazioni complesse senza generare un eccessivo traffico di rete.

12.3.4 Metodi di Oracle 8

I metodi (operazioni) sono stati aggiunti a Oracle 8 come parte dell'estensione a oggetti relazionale del sistema. Un **metodo** è una procedura o una funzione che fa parte della definizione di un **tipo di dati astratto** definito dall'utente. I metodi sono scritti in PL/SQL e memorizzati nella base di dati oppure scritti in un linguaggio di programmazione come C e memorizzati esternamente. Essi differiscono dalle procedure memorizzate per i seguenti motivi:

- un programma esegue un metodo facendo un riferimento a un oggetto del tipo che gli è associato;
- un metodo Oracle ha completo accesso agli attributi dell'oggetto che gli è associato e alle informazioni sul suo tipo (si noti che ciò in generale non vale per i modelli di dati a oggetti).

Ogni tipo di dati (astratti) ha un **metodo costruttore** definito dal sistema che è un metodo che crea un nuovo oggetto seguendo la specificazione del tipo di dati. Il nome del metodo costruttore è identico al nome del tipo definito dall'utente; si comporta come una funzione e restituisce il nuovo oggetto come suo valore. Oracle supporta alcuni tipi speciali di metodi:

- i metodi di confronto definiscono una relazione d'ordine tra gli oggetti di un dato tipo di dati;
- i metodi di map sono funzioni definite su tipi incorporati per confrontarli rapidamente (ad esempio un metodo map chiamato area può essere usato per confrontare dei rettangoli in base alle loro aree);
- i metodi che ordinano usano una loro logica interna per restituire un valore che codifichi l'ordine tra due oggetti dello stesso tipo. Ad esempio per un tipo di oggetto insurance_policy (che contiene una polizza assicurativa) possono essere definiti due diversi metodi di ordinamento: uno che ordina le polizze per (issue_date, lastname, firstname), cioè secondo la data di stipulazione e il cognome e nome del cliente e un altro per (policy_number), cioè secondo il numero della polizza.

12.3.5 Trigger

Oracle permette di impostare delle regole attive attraverso i **trigger**. Si tratta di procedure (o regole) memorizzate che sono eseguite (o attivate) automaticamente quando nella tabella a cui sono associate avviene un inserimento, un'eliminazione o un aggiornamento. I trigger possono essere usati per imporre vincoli di integrità o per eseguire automaticamente operazioni aggiuntive che sono richieste da regole o da politiche gestionali che vanno al di là dei consueti vincoli di chiave, d'integrità d'entità e d'integrità referenziale imposti dal sistema.

tuale *minima* di spazio nel blocco che deve essere raggiunta (grazie alle istruzioni DELETE e UPDATE che riducono le dimensioni dei dati) prima che nuove righe possano essere aggiunte al blocco. Ad esempio, se nell'istruzione CREATE TABLE si imposta

PCTUSED 50

il blocco di dati usato per il segmento di dati di questa tabella, che ha già raggiunto il 70 per cento del suo spazio di memoria come determinato da PCTFREE, è considerato non disponibile per l'inserimento di nuove righe finché la quantità di spazio usato nel blocco non diminuisce sotto il 50 per cento.⁵ In questo modo il 30 per cento del blocco rimane utilizzabile per aggiornamenti delle righe esistenti; nuove righe possono essere introdotte solo quando la quantità di spazio usato scende sotto il 50 per cento, mentre gli inserimenti possono essere eseguiti finché il 70 per cento dello spazio non è utilizzato.

Quando si usano tipi di dati specifici di Oracle come LONG o LONG RAW o in alcune altre situazioni che utilizzano oggetti grandi, può capitare che una riga abbia una dimensione superiore a quella di un blocco di dati riservati in quel segmento. In tal caso Oracle memorizza i dati della riga in una catena di blocchi di dati riservati per quel segmento: questo processo è detto concatenamento delle righe. Se una riga in origine si adattava a un blocco, ma la sua dimensione aumenta troppo a seguito di un aggiornamento dei dati, Oracle utilizza la migrazione, spostando l'intera riga in un nuovo blocco di dati e cercando di adattarla a questa nuova posizione. In questo caso, nella riga originale viene posto un puntatore al nuovo blocco di dati. Si noti che il concatenamento delle righe e la migrazione richiedono l'accesso a più blocchi per rintracciare i dati, e le prestazioni di conseguenza peggiorano.

12.4.2 Estensioni

Quando viene creata una tabella, Oracle le assegna un'estensione iniziale. Estensioni incrementali vengono allocate automaticamente quando l'estensione iniziale non è più sufficiente. La clausola STORAGE di CREATE TABLE viene usata per stabilire per ogni tipo di segmento quanto spazio assegnare inizialmente, la quantità massima di spazio e il numero di estensioni.⁶ Le estensioni allocate ai segmenti di un indice rimangono assegnate fin tanto che esiste l'indice. Quando un indice associato a una tabella o a un cluster viene eliminato, Oracle riutilizza lo spazio.

12.4.3 Segmenti

Un segmento è costituito da diverse estensioni e appartiene a uno spazio delle tabelle. Oracle usa i seguenti quattro tipi di segmenti.

- *Segmenti dei dati*: ogni tabella non raggruppata a cluster e ogni cluster dispone di un singolo segmento per mantenere tutti i suoi dati. Oracle crea il segmento dei dati quando l'applicazione crea la tabella o il cluster con il comando CREATE. I parametri di memorizzazione possono essere impostati e modificati con i comandi CREATE e ALTER.
- *Segmenti dell'indice*: ogni indice di una base di dati Oracle dispone di un unico segmento dell'indice che viene creato con il comando CREATE INDEX. L'istruzione deve indicare lo spazio delle tabelle a cui si riferisce e specificare i parametri di memoria del segmento.
- *Segmenti temporanei*: sono creati da Oracle per essere usati dalle istruzioni SQL che hanno bisogno di un'area di lavoro temporanea. Quando l'istruzione completa l'esecuzione, le estensioni da essa utilizzate vengono restituite al sistema per un uso futuro. Le istruzioni che richiedono un segmento temporaneo sono CREATE INDEX, SELECT . . . {ORDER BY | GROUP BY}, SELECT DISTINCT e (SELECT . . .) {UNION | MINUS | INTERSECT} (SELECT . . .). Anche alcuni join non indicizzati e le sottointerrogazioni relative possono richiedere segmenti temporanei. Le interrogazioni con le clausole ORDER BY, GROUP BY o DISTINCT che richiedono un'operazione di ordinamento possono essere effettuate in modo più efficiente usando il parametro SORT_AREA_SIZE.
- *Segmenti di rollback*: ogni base di dati deve contenere uno o più segmenti di rollback, usati per "annullare" le transazioni. Un segmento di rollback registra (che esegua o meno il commit) i valori vecchi dei dati usati per garantire la coerenza nella lettura (quando si usa il controllo multiversione) per realizzare il rollback di una transazione o per recuperare la versione precedente di una base di dati. Oracle crea un segmento di rollback iniziale chiamato SYSTEM ogni volta che viene creata una base di dati, il quale si trova nello spazio delle tabelle SYSTEM e utilizza i parametri di memorizzazione predefiniti di quello spazio.

12.5 La programmazione delle applicazioni Oracle

La programmazione in Oracle può essere effettuata in più modi:

- scrivendo interrogazioni interattive SQL nella modalità di interrogazione SQL;
- scrivendo programmi in un linguaggio host come COBOL, C o PASCAL e encapsulando SQL all'interno del programma. In questo caso, viene usato un precompilatore come PRO*COBOL o PRO*C per collegare l'applicazione a Oracle;
- scrivendo in PL/SQL, che è il linguaggio procedurale proprio di Oracle;
- usando le librerie OCI (Oracle Call Interface, Interfaccia di Chiamata di Oracle) e la libreria runtime SQLLIB di Oracle.

⁵ Queste istruzioni sono esempi di quanto chiamato linguaggio di definizione della memorizzazione nel Capitolo 2. Esse non fanno parte dello standard SQL.

⁶ I dettagli degli algoritmi di allocazione sono descritti in Oracle (1997a).

⁷ MINUS è equivalente a EXCEPT (Capitolo 8).

12.5.1 Programmazione in PL/SQL

PL/SQL è un linguaggio procedurale di Oracle che permette di includere interrogazioni SQL. Esso offre agli sviluppatori tutte le principali funzioni di ingegneria del software, quali l'incapsulamento e la protezione dei dati (data encapsulation), il nascondere informazioni (information hiding), il sovraccarico (overloading) e la gestione delle eccezioni, ed è la tecnica più usata per lo sviluppo di applicazioni in Oracle.

PL/SQL è un linguaggio strutturato a blocchi. Le unità base, cioè procedure, funzioni e blocchi anonimi, sono blocchi logici e possono contenere qualsiasi numero di sottoblocchi nidificati. Un blocco o sottoblocco raggruppa logicamente dichiarazioni e istruzioni. Le dichiarazioni sono locali al blocco in cui vengono fatte e non sono visibili al di fuori di esso. Un blocco PL/SQL ha tre parti: (1) una **parte dichiarativa** in cui vengono dichiarati variabili e oggetti, (2) una **parte eseguibile** in cui tali variabili sono manipolate e (3) una **parte delle eccezioni** in cui possono essere gestite le eccezioni o gli errori sorti durante l'esecuzione.

```
[ DECLARE
    --declarations ]
BEGIN
    --statements
[ EXCEPTION
    --handlers ]
END ;
```

Nella parte dichiarativa, che è facoltativa, vengono dichiarate le variabili. Le variabili possono essere di qualsiasi tipo di dati SQL e anche di tipi di dati aggiuntivi propri di PL/SQL. In questa parte possono venire assegnati alle variabili anche dei valori. Gli oggetti sono poi manipolati nella parte eseguibile, che è l'unica parte obbligatoria. Qui i dati possono essere elaborati usando istruzioni di controllo di flusso sequenziali, iterative e condizionali come IF-THEN-ELSE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN e GO-TO. Dal canto suo, la parte delle eccezioni gestisce qualsiasi condizione di errore verificatasi nella parte eseguibile. Le eccezioni possono essere errori definiti dall'utente, errori di basi di dati o errori di sistema. Quando ha luogo un errore o un'eccezione, viene evocata un'eccezione e la normale esecuzione si blocca e il controllo viene trasferito alla parte di gestione delle eccezioni del sotto-programma o del blocco PL/SQL.

Si supponga di voler scrivere programmi PL/SQL per elaborare la base di dati di Figura 7.5. Come primo esempio, E1, si è scritto un segmento di programma che stampa alcune informazioni sul dipendente che ha lo stipendio più alto:

```
E1:
DECLARE
    v_nome_batt  impiegato.nome_batt%TYPE;
    v_iniz_int   impiegato.iniz_int%TYPE;
    v_cognome   impiegato.cognome%TYPE;
    v_indirizzo  impiegato.indirizzo%TYPE;
    v_stipendio  impiegato.stipendio%TYPE;
```

```
BEGIN
    SELECT nome_batt, iniz_int, cognome, indirizzo, stipendio
    INTO v_nome_batt, v_iniz_int, v_cognome, v_indirizzo, v_stipendio
    FROM IMPIEGATO
    WHERE stipendio = (select max (stipendio) from impiegato);

    DBMS_OUTPUT.PUT_LINE (v_nome_batt, v_iniz_int, v_cognome,
    v_indirizzo, v_stipendio);

EXCEPTION
    WHEN OTHERS
        DBMS_OUTPUT.PUT_LINE ('Individuato errore');
END;
```

In E1 è stato necessario dichiarare alcune variabili dello stesso tipo e che conterranno gli attributi della base di dati che il programma elaborerà. Esse possono avere o non avere gli stessi nomi degli attributi corrispondenti. La clausola %TYPE in ogni dichiarazione significa che quella variabile è dello stesso tipo della colonna corrispondente nella tabella. DBMS_OUTPUT.PUT_LINE è la funzione di stampa di PL/SQL. La parte di gestione degli errori del programma esemplificativo effettua la stampa di un messaggio di errore se, durante l'esecuzione della frase SQL, Oracle individua un errore (in questo caso l'errore si verifica se viene selezionato più di un dipendente). Il programma comprende una clausola INTO che specifica le variabili del programma che contengono i valori degli attributi estratti dalla base di dati.

L'esempio successivo, E2, è un semplice programma per aumentare lo stipendio degli impiegati il cui salario è inferiore del 10 per cento al compenso medio. Il programma ricalcola e stampa lo stipendio medio se è superiore a 50.000 una volta eseguito l'aggiornamento.

```
E2:
DECLARE
    stipendio_medio NUMBER;

BEGIN
    SELECT avg(stipendio)INTO stipendio_medio
    FROM impiegato;

    UPDATE impiegato
    SET stipendio = stipendio*1.1
        WHERE stipendio < stipendio_medio;

    SELECT avg(stipendio) INTO stipendio_medio
    FROM impiegato;

    IF stipendio_medio > 50000 THEN
        dbms_output.put_line ('Lo stipendio medio è ' || stipendio_medio);
    END IF;

    COMMIT;
```

```

EXCEPTION
WHEN OTHERS THEN
dbms_output.put_line ('Errore in aggiornamento stipendio ')
ROLLBACK;

END;

```

In E2, `stipendio_medio` è definita come una variabile e contiene il valore della media dello stipendio dei dipendenti calcolato dalla prima istruzione `SELECT`; tale valore è usato per scegliere gli impiegati i cui salari saranno aggiornati. La parte `EXCEPTION` annulla l'intera transazione (cioè elimina qualsiasi effetto della transazione sulla base di dati) se un errore di qualsiasi tipo avviene durante l'esecuzione.

12.5.2 Cursori in PL/SQL

L'insieme di righe restituite da un'interrogazione può consistere in zero, una o più righe, a seconda di quante righe soddisfano i criteri di ricerca. Quando un'interrogazione restituisce più righe è necessario dichiarare esplicitamente un *cursor* per elaborarle. Un cursore è simile a un *puntatore a file* (file variable o file pointer) che fa riferimento a una singola riga (tupla) del risultato di un'interrogazione. Esso deve essere dichiarato nella parte dichiarativa dei programmi PL/SQL ed è controllato da tre comandi: `OPEN`, `FETCH` e `CLOSE`. Il cursore è inizializzato con l'istruzione `OPEN` che esegue l'interrogazione, recupera l'insieme di righe risultato e posiziona il cursore prima della prima riga nel risultato dell'interrogazione: tale posizione diventa la riga corrente per il cursore. L'istruzione `FETCH` quando è eseguita per la prima volta, copia la prima riga del risultato nelle variabili del programma e posiziona il cursore su quella riga. Le esecuzioni successive di `FETCH` fanno avanzare il cursore alla riga successiva dell'insieme risultato, copiando quella riga nelle variabili del programma. Questo procedimento è simile all'elaborazione tradizionale dei file, un record per volta. Quando l'ultima riga è stata elaborata, il cursore viene rilasciato con l'istruzione `CLOSE`. L'esempio E3 visualizza il numero di previdenza sociale (SSN) degli impiegati il cui stipendio è superiore a quello dei loro direttori.

```

E3:
DECLARE
stipendio_imp NUMBER;
super_stipendio_imp NUMBER;
ssn_imp CHAR (9);
superssn_imp CHAR (9);
CURSOR cursore_stipendio IS
SELECT ssn, stipendio, superssn FROM impiegato;
BEGIN
OPEN cursore_stipendio;

```

```

LOOP
FETCH cursore_stipendio INTO ssn_imp, stipendio_imp, superssn_imp;
EXIT WHEN cursore_stipendio%NOTFOUND;

IF superssn_imp is NOT NULL THEN
SELECT stipendio INTO super_stipendio_imp
FROM impiegato
WHERE ssn = superssn_imp;

IF stipendio_imp > super_stipendio_imp THEN
dbms_output.put_line(ssn_imp);
END IF;
END IF;
END LOOP;
IF cursore_stipendio%ISOPEN THEN CLOSE cursore_stipendio;

EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line ('Errori negli ssn ' || ssn_imp);
IF cursore_stipendio%ISOPEN THEN CLOSE cursore_stipendio;

END;

```

Nell'esempio precedente CURSOR_STIPENDIO scorre nell'intera tabella dei dipendenti finché il cursore non localizza altre righe. La parte delle eccezioni gestisce la situazione in cui un codice di supervisore ssn non corretto può essere assegnato a un dipendente. `%NOTFOUND` è uno dei quattro possibili attributi del cursore. Ecco un elenco:

- * `%ISOPEN` restituisce TRUE, cioè VERO, se il cursore è già aperto;
- * `%FOUND` restituisce TRUE se l'ultimo `FETCH` ha restituito una riga e restituisce FALSE, cioè FALSO, se l'ultimo `FETCH` non è riuscito a restituire una riga;
- * `%NOTFOUND` è l'opposto logico di `%FOUND`;
- * `%ROWCOUNT` fornisce il numero di righe localizzate.

Come ultimo esempio, E4 mostra un segmento del programma che prende un elenco di tutti i dipendenti, aumenta del 10 per cento lo stipendio di ogni impiegato e visualizza il vecchio e il nuovo salario.

```

E4:
DECLARE
v_nome_batt  impiegato.nome_batt%TYPE;
v_iniz_int   impiegato.iniz_int%TYPE;
v_cognome    impiegato.cognome%TYPE;
v_indirizzo  impiegato.indirizzo%TYPE;
v_stipendio  impiegato.stipendio%TYPE;

CURSOR IMP IS
SELECT ssn, nome_batt, iniz_int, cognome, stipendio
FROM impiegato;

```


solito, gli attributi impliciti del cursore contengono informazioni sull'esecuzione di un'istruzione INSERT, UPDATE, DELETE o SELECT INTO. I valori di questi attributi del cursore fanno sempre riferimento all'istruzione SQL eseguita per ultima. Ad esempio, in E6 l'attributo del cursore NOTFOUND è una variabile隐式a che restituisce TRUE se l'istruzione SQL non ha restituito nessuna riga.

```
E6:
... /* stesse istruzioni include e dichiarazioni di variabili di E5

main ()
{
strcpy (username.arr , "Scott") ;
username.len= strlen(username.arr);
strcpy(password.arr,"TIGER") ;
password.len = strlen(password.arr);

EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL CONNECT :username IDENTIFIED BY :password ;
EXEC SQL DECLARE IMP CURSOR FOR
    SELECT   ssn, nome_batt, iniz_int, cognome, stipendio
    FROM     impiegato ;

EXEC SQL OPEN IMP ;
EXEC SQL WHENEVER NOTFOUND DO BREAK ;

for (;;)
{
EXEC SQL FETCH EMP INTO :v_ssn, :v_nome_batt, :v_iniz_int, :v_cognome,
:f_stipendio ;
printf ("Numero di previdenza sociale : %d , Vecchio stipendio : %f " ,
v_ssn, f_stipendio ) ;

EXEC SQL UPDATE  impiegato
    SET      stipendio = stipendio*1.1
    WHERE    ssn = :v_ssn ;
EXEC SQL COMMIT;
    printf ("Numero di previdenza sociale : %d Nuovo stipendio : %f " ,
v_ssn, f_stipendio*1.1
}
}

sql_error()
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf(" Individuato errore \n");
}
```

12.6 Strumenti di sviluppo di Oracle

Esistono vari strumenti per sviluppare applicazioni e per progettare basi di dati con RDBMS, e molti assistono il progettista in tutte le fasi del progetto della base di dati, a partire dalla progettazione concettuale attraverso varianti dei diagrammi estesi entità-relazione per arrivare al progetto fisico. Oracle, per questo specifico scopo, mette a disposizione un proprio strumento, chiamato Designer 2000.

Designer 2000 consente lo sviluppo rapido basato su modelli e fornisce *diagrammi Entità-Relazione* per la modellazione dei dati, l'*approccio a gerarchia funzionale* per la modellazione dei processi e la *gestione di flussi di dati* per catturare i flussi d'informazione all'interno di un sistema informativo. L'unità per lo sviluppo di diagrammi Entità-Relazione di Designer 2000 supporta la creazione, la visualizzazione e la manipolazione di tutti i tipi di entità e relazioni. Questi costrutti, compresi gli attributi, vengono definiti in termini di proprietà. Le proprietà degli attributi sono visualizzate usando simboli grafici predefiniti per indicare attributi obbligatori, quelli facoltativi e quelli univocamente identificativi (chiavi). Le entità e le relazioni sono visualizzate tramite diagrammi per consentire una miglior comprensione.

L'unità di sviluppo diagrammi a gerarchia funzionale rappresenta invece le attività (processi organizzativi) sviluppate da un'azienda. Si utilizza la tecnica di decomposizione funzionale, per cui la descrizione ad alto livello di una funzione professionale o commerciale è divisa in funzioni progressivamente più dettagliate. Questo aiuta a identificare le funzioni aziendali candidate a essere informatizzate e le aree comuni a tutta l'organizzazione.

L'unità di sviluppo diagrammi a matrice è uno strumento di uso generale che può essere usato per calcolare la portata del progetto, eseguire l'analisi d'impatto, la pianificazione della rete e il controllo di qualità del progetto di sviluppo di applicazioni di basi di dati. Fornisce anche informazioni relative ai differenti nodi di rete su cui localizzare le tabelle della base di dati e i moduli che usano queste tabelle.

Per sviluppare le applicazioni sono disponibili molti strumenti interattivi di prototipizzazione tra cui Powerbuilder di Sybase. Oracle mette a disposizione uno strumento chiamato Developer 2000 che permette all'utente di costruire la GUI del progetto e di sviluppare interattivamente programmi composti da interrogazioni e transazioni. Lo strumento di sviluppo interagisce con le basi di dati di Oracle e il back end. Il Developer 2000 comprende una serie di generatori di moduli, report, interrogazioni; inoltre, offre la possibilità di costruire oggetti, grafici e procedure che rendono più semplice per gli sviluppatori costruire applicazioni che si basano su una base di dati. La versione 2.0 comprende varie autocomposizioni grafiche per rendere automatica la creazione delle applicazioni. L'ambiente di sviluppo visuale permette ai programmatore di riutilizzare i componenti semplicemente trascinandoli nelle loro applicazioni. Se necessario, gli sviluppatori possono usare l'ambiente visuale anche per spostare il codice dal client al server, in modo da ridurre il traffico sulla rete. Developer 2000, infine, comprende uno strumento di gestione dei progetti che consente lo sviluppo cooperativo delle applicazioni e un debugger che permette di identificare e correggere gli errori in tutti gli strati dell'applicazione. Developer 2000 si integra perfettamente con il Designer 2000 di Oracle, offre l'accesso a tutte le più diffuse basi di dati commerciali e consente d'incorporare controlli ActiveX nelle applicazioni.

12.7 Una visione d'insieme di Microsoft Access

Access è una delle implementazioni più diffuse del modello di dati relazionale sulla piattaforma PC. Fa parte di una serie integrata di strumenti per creare e gestire le basi di dati sulla piattaforma Windows. Le applicazioni delle basi di dati Access possono andare da applicazioni personali, come la gestione dell'inventario di una collezione personale di video o di CD, a piccole applicazioni aziendali, come la gestione dei clienti di una piccola azienda. Inoltre, grazie alla conformità del prodotto Microsoft allo standard ODBC (Open Database Connectivity) e la prevalenza oggi di architetture client/server, le basi di dati relazionali su PC possono anche essere usate come un'interfaccia verso basi di dati memorizzate su piattaforme non PC, ad esempio un utente finale può specificare graficamente delle interrogazioni in Access per poi eseguirle su una base di dati Oracle installata su un server UNIX.

Access mette a disposizione un motore di basi di dati e un'interfaccia utente grafica per la definizione e l'interrogazione dei dati in linguaggio SQL. Fornisce anche un proprio linguaggio di programmazione chiamato Access Basic. Gli utenti possono sviluppare rapidamente maschere e report per operazioni d'input/output sulla base di dati attraverso l'utilizzo delle creazioni guidate, programmi interattivi dialogici che assistono l'utente ponendogli una serie di domande. La definizione delle maschere e dei report è eseguita interattivamente dall'utente che progetta il layout collegando i diversi campi posti sulla maschera o sul report a elementi della base di dati. Access 2000 (la più recente versione di Access) permette di memorizzare collegamenti ipertestuali, definendo allo scopo un apposito tipo di dati, che estende le funzionalità della base di dati, rendendo più facile condividere informazioni su Internet.

12.7.1 Architettura di Access

Il RDBMS Access si compone di diversi moduli, di cui uno è il motore di basi di dati detto Microsoft Jet Engine,⁸ responsabile della gestione dei dati. Un altro componente è l'interfaccia utente che utilizza il motore per fornire servizi come la memorizzazione e il recupero dei dati. Il motore salva tutti i dati di una applicazione (tabelle, indici, maschere, report, macro e moduli) in un unico file del tipo proprietario di basi di dati Microsoft (file con estensione .mdb). Il motore mette a disposizione anche capacità avanzate come l'accesso a dati esterni attraverso ODBC, la convalida dei dati, il controllo della concorrenza con l'uso di "locks" (blochi) e l'ottimizzazione delle interrogazioni.

Access costituisce un ambiente di sviluppo applicativo completo, con il motore interno che offre tutti i servizi di un RDBMS. L'interfaccia utente di Access comprende Creazioni guidate e Generatori per aiutare l'utente nella progettazione delle applicazioni. I Generatori sono programmi interattivi per creare espressioni sintatticamente corrette. Il modello di programmazione usato da Access è guidato dagli eventi. L'utente costruisce una sequenza di operazioni semplici, chiamate **macro**, che devono essere eseguite in risposta ad azioni che avvengono durante l'utilizzo dell'applicazione di basi di dati. Molte applicazioni semplici possono essere scritte interamente usando le macro; altre, più complesse, possono richiedere le funzionalità di **Access Basic**, il linguaggio di programmazione fornito da Access.

Questo linguaggio permette di strutturare un'applicazione Access come un componente, in grado di interagire con altre. Un **componente** (nella terminologia Microsoft) è un modulo applicativo che rende i suoi oggetti disponibili per altre applicazioni. In Visual Basic, ad esempio, è possibile lavorare con componenti sviluppati in altri linguaggi per costruire senza problemi un'applicazione integrata. Utilizzando la tecnologia OLE (object linking and embedding), un utente può inserire dei documenti creati da un altro componente all'interno di un report o di una maschera di Access. La tecnologia OLE fa parte del **modello a oggetti componenti** (COM: component object model), uno standard proposto da Microsoft.

12.7.2 Definizione dei dati delle basi di dati di Access

Sebbene Access permetta un approccio programmatico alla definizione dei dati attraverso Access SQL, il suo dialetto di SQL, la GUI di Access mette a disposizione un comodo metodo grafico per definire le tabelle e le relazioni tra di esse. Le tabelle possono essere create direttamente in visualizzazione **Struttura** oppure interattivamente, seguendo una creazione guida-

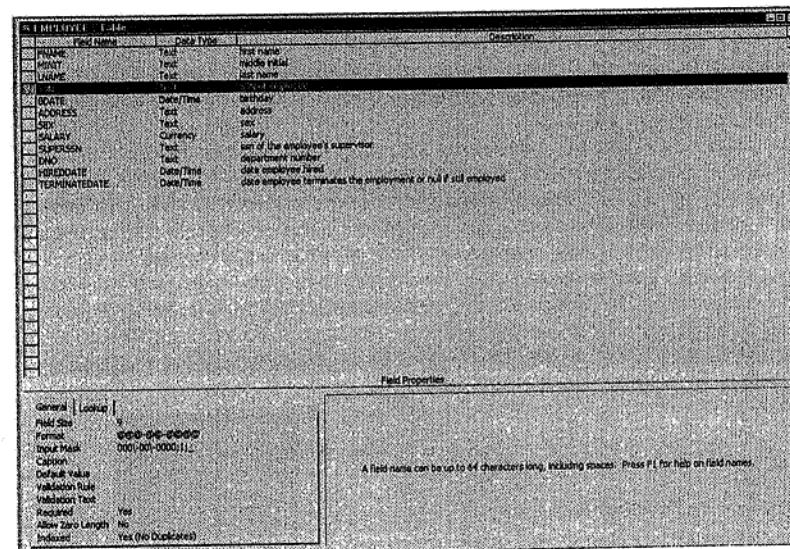


Figura 12.3 Definizione della relazione IMPIEGATO (EMPLOYEE) usando l'interfaccia utente Access.

⁸ Nel resto del capitolo vi si farà riferimento semplicemente come "motore".

data. La definizione di una tabella contiene non solo la struttura della tabella, ma anche la formattazione dei campi, le maschere per gli input dei campi stessi, le regole di convalida, le didascalie, i valori predefiniti, gli indici e così via. I tipi di dati per gli attributi sono Testo, Memo, Numerico, Data/ora, Valuta, Contatore, Sì/No (booleano), Oggetto OLE, Collegamento ipertestuale e Ricerca guidata. Access consente anche d'importare i dati da tabelle esterne e creare collegamenti con tabelle di altre base di dati.

In Figura 12.3 è mostrata la tabella EMPLOYEE dello schema della base di dati relazionale COMPANY, aperta in visualizzazione Struttura. È selezionato (evidenziato) il campo SSN e le sue proprietà sono visualizzate nella finestra Proprietà campo posta nella parte inferiore dello schermo. Viene utilizzato il formato di visualizzazione predefinito che prevede che il codice SSN abbia per convenzione dei trattini dopo la terza e la quinta posizione. La maschera d'input fornisce i caratteri di formattazione automatica per la visualizzazione durante l'immissione dei dati, rendendo facile la convalida dei dati d'input.

In altri termini, la maschera d'input del campo SSN visualizza le posizioni dei trattini e indica che gli altri caratteri sono cifre numeriche. La proprietà didascalia specifica il nome visualizzato sulle maschere e sui report per descrivere questo campo. Se l'etichetta è lasciata vuota viene visualizzato il nome predefinito che è il nome stesso del campo. Può essere specificato anche un valore predefinito se appropriato per uno specifico campo. La convalida dei campi comprende la specificazione di apposite regole di convalida e del testo di avvertimento che viene visualizzato quando una regola di convalida viene violata. Nel caso del codice SSN, la maschera d'input fornisce già la regola di convalida del campo. Altri campi, tuttavia, possono richiedere regole di convalida aggiuntive; ad esempio può essere necessario che il campo SALARY sia maggiore di un certo valore minimo. Altre proprietà dei campi permettono di specificare se il campo è richiesto, cioè se il valore NULL non è ammesso, e se i campi testuali consentono stringhe di lunghezza zero. Un'altra proprietà dei campi include la specificazione dell'indice che dà tre possibilità: nessun indice, un indice con duplicati o un indice senza duplicati. Poiché SSN è la chiave primaria di EMPLOYEE nell'esempio, il campo è indicizzato e non è ammesso alcun duplicato.

Oltre alla finestra Proprietà campo, Access fornisce anche una finestra Proprietà tabella. Questa viene usata per specificare le regole di convalida delle tabelle che sono i vincoli di integrità esistenti tra le colonne di una tabella o tra tabelle diverse. Ad esempio l'utente può definire una regola di convalida sulla tabella EMPLOYEE specificando che un dipendente non può essere il supervisore di se stesso.

12.7.3 Definizione delle relazioni e dei vincoli di integrità referenziale

Attraverso la finestra Relazioni Access consente la definizione interattiva delle relazioni tra le tabelle, che permette anche di specificare i vincoli d'integrità referenziale. Per definire una relazione, l'utente anzitutto deve aggiungere le due tabelle collegate nella finestra, poi selezionare la chiave primaria di una tabella e trascinarla nell'altra tabella in cui compare come chiave esterna. Ad esempio, per definire la relazione tra DNUMBER di DEPARTMENT e DNO di EMPLOYEE, l'utente deve selezionare DNUMBER di DEPARTMENT e trascinarlo su

EMPLOYEE.DNO. Questa operazione fa comparire un'altra finestra che chiede all'utente ulteriori informazioni su come stabilire la relazione, come mostrato in Figura 12.4. L'utente seleziona la casella "Applica integrità referenziale" se Access deve applicare automaticamente l'integrità referenziale specificata dalla relazione. Selezionando le caselle appropriate, l'utente può anche specificare la necessità di aggiornare automaticamente i campi correlati a catena oppure di eliminare i record correlati a catena. Il tipo di relazione viene determinato automaticamente da Access in base alla definizione dei campi correlati. Se solo uno dei campi correlati è una chiave primaria o ha un indice univoco, Access crea una relazione uno-a-molti; questo significa che ogni istanza (valore) della chiave primaria può comparire molte volte come istanza della chiave esterna nella tabella correlata. È il caso dell'esempio trattato, poiché DNUMBER è la chiave primaria di DEPARTMENT e DNO non è la chiave primaria di EMPLOYEE (si noti che su DNO non è nemmeno definito un indice univoco). Se entrambi i campi sono chiavi o hanno indici univoci, Access crea una relazione uno-a-uno. Ad esempio si consideri la definizione di una relazione tra EMPLOYEE.SSN e DEPARTMENT.MGRSSN. Se la chiave MGRSSN di DEPARTMENT ha un indice senza duplicati (un indice univoco) e SSN è la chiave primaria di EMPLOYEE, Access crea automaticamente una relazione uno-a-uno.

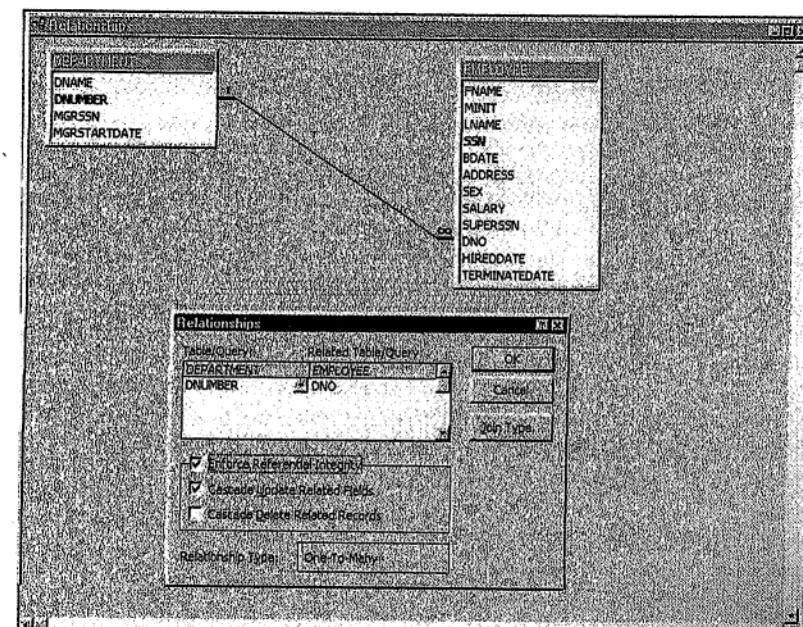


Figura 12.4 Come viene stabilita l'integrità referenziale tra DNUMBER e DNO

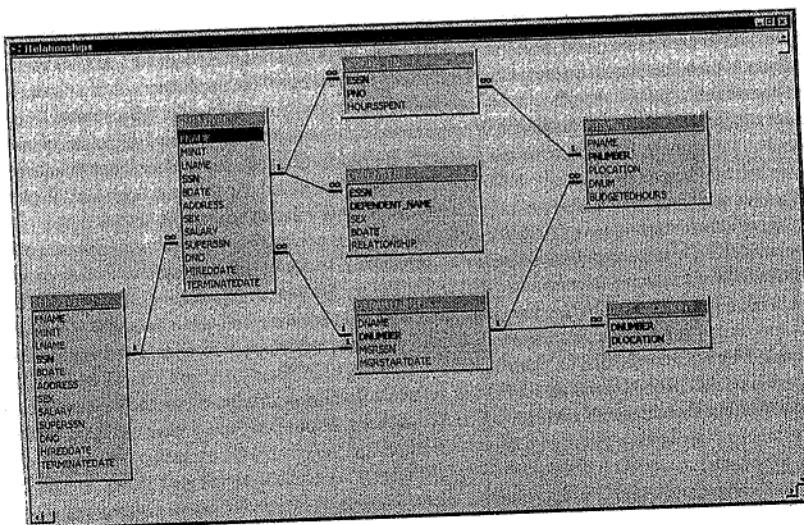


Figura 12.5 La finestra Relazioni (Relationship) per lo schema COMPANY.

Sebbene in Access l'indicazione di una relazione sia il meccanismo più usato per specificare l'integrità referenziale tra tabelle, l'utente non è obbligato a selezionare l'opzione per applicare l'integrità referenziale, perché le relazioni sono usate anche per rilevare le condizioni di join implicite per le interrogazioni. Durante l'esecuzione delle interrogazioni che coinvolgono più tabelle, anche se nessuna relazione è stata specificata, viene comunque eseguito un join predefinito sui campi correlati, indipendentemente dal fatto che l'integrità referenziale sia impostata o meno.⁹ Access utilizza il join interno come join predefinito, ma l'utente può scegliere un join esterno destro o sinistro facendo clic sulla casella "Tipo join" e selezionando l'appropriato tipo di join (Figura 12.4).

In Figura 12.5 è mostrata la finestra Relazioni per lo schema di base di dati COMPANY in Access. Si noti la somiglianza con la Figura 7.7 che mostra gli otto vincoli d'integrità referenziale della base di dati COMPANY. La principale differenza è che Access visualizza il rapporto tra cardinalità associato a ciascuna relazione.¹⁰ Un'altra diversità è la visualizzazione duplicata della relazione EMPLOYEE come EMPLOYEE e EMPLOYEE_1 in Figura 12.5.

Questa duplicazione è necessaria quando si definiscono relazioni multiple tra due tabelle o una relazione ricorsiva (cioè una relazione tra una tabella e se stessa). In figura al fine di definire una relazione ricorsiva tra EMPLOYEE.SSN ed EMPLOYEE.SUPERSSN, l'utente ha

aggiunto un'altra copia della tabella EMPLOYEE alla finestra Relazioni prima di trascinare la chiave primaria SSN sulla chiave esterna SUPERSSN. Anche se la relazione ricorsiva non fosse esistita nello schema COMPANY, sarebbe stato comunque necessario duplicare EMPLOYEE (oppure in alternativa DEPARTMENT) perché esistono due relazioni tra EMPLOYEE e DEPARTMENT: SSN in MGRSSN e DNUMBER in DNO.

12.7.4 Manipolazione dei dati in Access

Le operazioni di manipolazione dei dati nel modello relazionale sono divise in due categorie: le interrogazioni e gli aggiornamenti (operazioni di inserimento, eliminazione e modifica). Access consente la definizione visuale delle interrogazioni attraverso un'interfaccia grafica del tipo "point-and-click" (indica e seleziona) chiamate anche Query-By-Example (QBE), perché l'interrogazione viene formulata fornendo indicazioni grafiche e non con l'uso di una sintassi esplicita. È possibile, però, operare anche in modo programmatico attraverso Access SQL. L'utente ha la possibilità di progettare l'interrogazione in forma grafica e poi passare alla visualizzazione SQL per esaminare la sintassi SQL creata dal programma. Per quanto riguarda gli aggiornamenti, Access consente di eseguire tali operazioni in vari modi: attraverso maschere predisposte dal programmatore dell'applicazione, usando la manipolazione diretta dei dati delle tabelle nella visualizzazione Foglio dati oppure attraverso il linguaggio di programmazione Access Basic.

Le modalità di esecuzione dell'interrogazione sono specificate graficamente tramite l'interfaccia Access QBE. Si consideri l'Interrogazione 2 sulla base di dati COMPANY che recupera i nomi e gli indirizzi di tutti i dipendenti che lavorano per il dipartimento "Ricerca". In Figura 12.6 mostra l'interrogazione in QBE in SQL.

Per definire l'interrogazione QBE l'utente deve prima aggiungere le tabelle EMPLOYEE e DEPARTMENT nella finestra dell'interrogazione (finestra della query). Il join predefinito tra DEPARTMENT.DNUMBER ed EMPLOYEE.DNO che è stato stabilito attraverso la finestra Relazioni al momento della definizione dei dati viene automaticamente incorporato nella definizione dell'interrogazione, come illustrato dalla riga che collega i campi correlati. Se questo join predefinito non è necessario per l'interrogazione, l'utente può evidenziare il collegamento nella finestra della query e premere il tasto di cancellazione della tastiera. Per stabilire un join che non era stato prespecificato, basta selezionare l'attributo del join da una tabella e trascinarlo sull'attributo corrispondente nell'altra tabella. Per includere un attributo nell'interrogazione, basta trascinarlo dalla finestra superiore alla finestra inferiore. Perché gli attributi siano visualizzati nel risultato, occorre selezionare la casella "Mostra". Per specificare una condizione di selezione su un attributo, si può digitare l'espressione direttamente nella griglia "Criteri" oppure usare l'aiuto di un Generatore di espressioni. Per vedere l'interrogazione equivalente in Access SQL, l'utente può commutare dalla visualizzazione Struttura QBE alla visualizzazione SQL.¹¹

⁹ Specificare una relazione equivale a definire una *condizione di join implicita* nelle interrogazioni che coinvolgono le due tabelle, a meno di precisare una condizione diversa (condizione di join) durante la specificazione delle query.

¹⁰ I rapporti di cardinalità sono simili a quelli usati nei diagrammi ER (Figura 3.2), ma al posto di *N* è usato in Access il simbolo di infinito.

¹¹ Si noti che SQL Access consente di specificare un join nella clausola FROM, come consentito dallo standard SQL2.

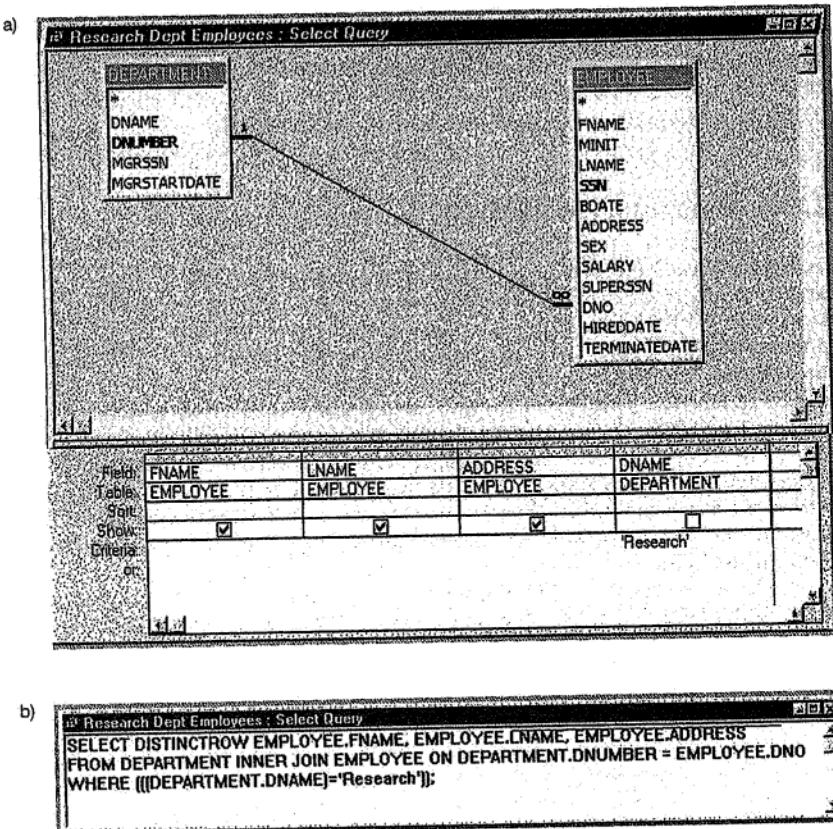


Figura 12.6 La definizione della Interrogazione 2 in Access QBE (a) e Access SQL (b).

In modo simile è possibile creare interrogazioni che comprendono join multipli. Sono disponibili creazioni guidate per assistere l'utente nella definizione delle interrogazioni, sebbene la facilità d'utilizzo di Access QBE non lo renda quasi mai necessario.

Le operazioni di aggiornamento sulla base di dati sono invece eseguite per mezzo di maschere che incorporano la logica di funzionamento dell'applicazione. Vi è anche una visualizzazione Foglio dati che l'utente finale sofisticato può usare per inserire, eliminare o modificare i dati direttamente nelle tabelle. Questi aggiornamenti sono soggetti a vincoli specificati al momento della definizione dei dati, tra cui i tipi di dati, le maschere d'input, le regole di controllo della tabella e dei campi e le relazioni.

12.8 Caratteristiche e funzionalità di Access

12.8.1 Le maschere

Access fornisce una Creazione guidata maschera per assistere il programmatore nello sviluppo di maschere. Un suo tipico scenario di utilizzo comprende le operazioni seguenti:

- scelta di una tabella o di una interrogazione da cui provengono o in cui verranno inseriti i dati della maschera;
- selezione dei campi che compaiono nella maschera;
- scelta del layout desiderato (A colonne, Tabulare, Foglio dati o Giustificato);
- scelta di uno stile per le etichette;
- specificazione di un titolo per la maschera.

Si noti che usare le interrogazioni nella definizione di una maschera significa trattarle come viste. La Creazione guidata crea automaticamente la maschera sulla base delle impostazioni dell'utente. La maschera, se lo si desidera, può essere aperta in visualizzazione Struttura per eventuali modifiche. In Figura 12.7 è mostrata una maschera, intitolata "Employee", creata attraverso la Creazione guidata maschera: essa comprende tutti i campi della tabella EMPLOYEE. Dal punto di vista grafico, è stato scelto un layout giustificato, con uno stile standard per le didascalie. La maschera mostrata è essenzialmente quella creata automaticamente dalla Creazione guidata, ma vi sono alcune accezioni. Le dimensioni di alcuni campi sono state modificate in visualizzazione Struttura selezionando il riquadro con il mouse e trascinandolo in modo che assumesse la dimensione appropriata. Questa semplice maschera consente all'utente di visualizzare, inserire, eliminare e modificare i record EMPLOYEE, rispettando però alcuni vincoli. L'utente visualizza i dati nella relazione EMPLOYEE facendo scorrere ripetutamente la pagina usando PgGiù (oppure il pulsante > sulla riga inferiore). È possibile cercare un particolare record usando la funzione "Trova" nel menu "Modifica". Una volta che il dipendente desiderato è stato trovato, è possibile aggiornare direttamente i suoi dati o eliminarlo usando la funzione "Elimina record" che si trova sempre nel menu "Modifica".

Per inserire un nuovo dipendente, l'utente inserisce i dati in un record vuoto, a cui si può accedere portandosi nella pagina successiva (oppure usando il pulsante > nella parte inferiore), oltre l'ultimo record nella tabella.

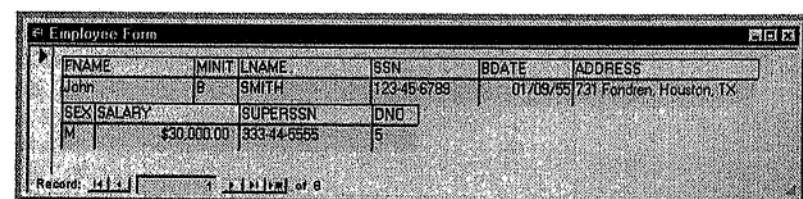


Figura 12.7 Una maschera per la tabella EMPLOYEE.

L'interfaccia utente di Access fornisce una sofisticata visualizzazione Struttura per la creazione di maschere più complicate. Il progettista di maschere dispone di una "casella degli strumenti" che mette a disposizione vari controlli da incorporare nella maschera, ad esempio pulsanti, caselle di controllo, caselle combinate e sottomaschere. I pulsanti e le caselle di controllo permettono di scegliere facilmente tra le opzioni presenti nella maschera. Le "caselle combinate" sono un comodo meccanismo per selezionare un elemento da un elenco di possibili valori invece di digitarlo, assicurando così la correttezza dei dati immessi: ad esempio, una casella combinata può essere usata per scegliere il codice del dipartimento a cui un dipendente appartiene. Le sottomaschere sono maschere all'interno di altre maschere e consentono alla maschera principale di includere informazioni provenienti da più tabelle: ad esempio, nella maschera Project può essere usata una sottomaschera per visualizzare informazioni sui dipendenti che lavorano su quel progetto. Vi sono poi "creazioni guidate Controllo" che assistono il progettista di maschere nella scelta dei controlli da usare.

12.8.2 I report

I report fanno parte integrante di qualsiasi sistema di basi di dati e forniscono vari metodi per raggruppare, ordinare e riassumere i dati per la stampa in base alle esigenze dell'utente. Come le maschere, i report si basano su tabelle o interrogazioni. Access mette a disposizione una Creazione guidata Report. Un suo tipico scenario di utilizzo comprende le operazioni seguenti:

- scelta di una tabella o di un'interrogazione da cui provengono i dati del report;
- selezione dei campi che compaiono nel report;

Salaries by Department			
DNO	LNAME	FNAME	SALARY
1	BORG	James	\$55,000.00
			\$55,000.00
4	JABBAR	Ahmad	\$25,000.00
	WALLACE	Jennifer	\$43,000.00
	ZELAYA	Alicia	\$25,000.00
			\$93,000.00
5	ENGLISH	Joyce	\$25,000.00
	NARAYAN	Ramesh	\$38,000.00
	SMITH	John	\$50,000.00
	WONG	Franklin	\$40,000.00
			\$133,000.00
	Total Salaries		\$281,000.00

Figura 12.8 Un report Access con il titolo Salaries by Department (Stipendi per Dipartimento).

- specificazione di livelli di raggruppamento dei dati all'interno del report;
- indicazione del tipo di ordinamento prescelto per il report;
- scelta del layout e dell'orientamento;
- specificazione di uno stile per il titolo del report.

La creazione guidata genera automaticamente il report in base alle impostazioni; il report poi può essere aperto in visualizzazione Struttura per eventuali modifiche.

In Figura 12.8 è mostrato un report con il titolo "Stipendi per Dipartimento" creato con la Creazione guidata Report usando i dati dalla relazione EMPLOYEE e scegliendo i campi nell'ordine in cui dovevano comparire sul report: DNO, LNAME, FNAME e SALARY.

Si noti che è stato specificato il raggruppamento dei dati secondo il campo DNO, in modo da raggruppare i salari in base al codice di dipartimento (DNO). È stato, inoltre, specificato l'ordinamento dei record sul campo LNAME ed è stata richiesta l'esecuzione della somma del campo SALARY. Il layout del report, il suo orientamento e lo stile sono stati scelti tra quelli predefiniti messi a disposizione da Access. Il report mostrato è essenzialmente quello fornito dalla Creazione guidata, con alcune eccezioni: ad esempio, i valori predefiniti delle didascalie per la nota a piè di pagina del gruppo e la nota a piè di pagina del report sono stati modificati selezionandoli in visualizzazione Struttura.

L'interfaccia utente di Access fornisce anche una visualizzazione Struttura sofisticata per creare report più complicati. La casella degli strumenti disponibile in questa visualizzazione è simile a quella del progettista di maschere; analogamente a quanto visto per le maschere, varie Creazioni guidate controllo sono a disposizione del progettista di report e lo assistono nella scelta dei controlli da aggiungere al report: ad esempio, un controllo "Sottoreport" unisce più report in uno e può essere usato per creare un report di dipartimento che include dei sottoreport per specificare analiticamente le informazioni sui progetti svolti da quel dipartimento.

12.8.3 Le macro e Access Basic

Il modello di programmazione di Access è guidato dagli eventi. Access è responsabile del riconoscimento degli eventi e l'utente può specificare come rispondere a ciascun evento scrivendo una **macro** che è una sequenza di semplici operazioni chiamate **azioni**. Gli eventi rilevati da Access comprendono le modifiche ai dati, l'esecuzione di azioni su una finestra, il digitarre su tastiera e le operazioni compiute con il mouse. Ad esempio si può scrivere una macro perché risponda all'evento di chiusura di una finestra, specificando l'azione "ApriMaschera" per aprire un'altra finestra.

Le azioni disponibili permettono ai programmati di macro di costruire intere applicazioni, ma per le applicazioni più complesse è consigliabile ricorrere al linguaggio Access Basic. Si tratta di un linguaggio di programmazione completo che consente l'utilizzo di costrutti di controllo del flusso esecutivo. Inoltre in Access Basic è possibile passare argomenti a procedure personalizzate (anche scritte in Access Basic) e manipolare i record uno alla volta. I moduli sulla barra degli strumenti principale corrispondono ad altrettante procedure preprogrammate scritte in Access Basic.

12.8.4 Altre caratteristiche

Access supporta vari tipi di interrogazioni avanzate, come le **interrogazioni a campi incrociati**, che permettono di raggruppare i dati secondo i valori in una colonna ed eseguire funzioni di aggregazione come somme e medie all'interno dei gruppi. In Excel queste interrogazioni prendono il nome di tabelle pivot.

Access, infine, consente di scambiare agevolmente informazioni con altre applicazioni. L'uso di controlli Active X all'interno di Access estende l'utilizzo dell'applicazione con poca o nessuna nuova programmazione.

Sicurezza. Access ha un modello di sicurezza a livello utente simile a quello di Windows NT Server in cui gli utenti forniscono un nome e una password quando attivano il programma, e i loro identificativi utente e di gruppo determinano i privilegi che possono essere impostati dall'amministratore usando una Creazione guidata. Access, inoltre, fornisce i seguenti metodi per proteggere un'applicazione:

- l'opzione di avvio dell'applicazione Access può essere usata per limitare l'accesso alla finestra Base di dati e ad alcune combinazioni di tasti;
- un'applicazione può essere salvata come file MDE per eliminare il codice sorgente Visual Basic ed evitare cambiamenti alla struttura delle maschere, dei report e dei moduli attraverso indebite modifiche al codice dei programmi.

Replica. Access supporta anche la replica della base di dati. Attraverso il menu degli strumenti è possibile impostare la **replica completa** o parziale di una base di dati. Si noti, però, che la base di dati Access deve essere convertita in una "Replica Master" prima che i comandi di replica possano essere usati. Una volta fatto questo, si possono creare due o più copie della base di dati chiamate **repliche**; ogni replica può contenere anche altri oggetti locali. Una **Replica Master** è la replica su cui è possibile eseguire modifiche alla struttura della base di dati e agli oggetti. Il comando **Replica** sul menu degli strumenti consente la creazione di una replica e la sincronizzazione della replica con un altro membro dell'insieme delle repliche. La sincronizzazione tra repliche può essere eseguita da un comando di menu o programmaticamente in Visual Basic. Esistono anche un comando di menu **Risolvi conflitti** e un modulo aggiuntivo chiamato **Manager delle repliche** che fornisce un'interfaccia visuale per convertire le basi di dati in repliche master, crea ulteriori repliche, visualizza le relazioni tra le repliche, imposta le loro proprietà e così via. Il Manager delle repliche consente anche la sincronizzazione dei dati su Internet o su una Intranet.

Operazioni multutente. Per rendere disponibile un'applicazione per l'accesso multiutente, l'applicazione deve essere installata su un server di rete. Microsoft Access fornisce la nozione di blocco per garantire la correttezza degli aggiornamenti concorrenti. Il blocco può essere eseguito programmaticamente in Visual Basic, ma viene fatto automaticamente da Access quando utenti indipendenti usano maschere che si basano sulla stessa tabella. Access tiene un file LDB che contiene le informazioni correnti sui blocchi. Le opzioni di blocco sono: Nessun blocco, tutti i record, le Record, le righe, le righe specifiche, i propri record, i record relativi a que-

ste opzioni può essere impostata per una singola maschera oppure per l'intera base di dati (dal menu Strumenti si scelga il comando Opzioni e poi il comando Avanzate).

Edizione dello sviluppatore. Una base di dati Access può essere salvata come un file MDE compilando il codice sorgente Visual Basic. Con l'**Edizione dello sviluppatore** (DE: Developer's Edition) il file MDE consente la distribuzione dell'applicazione su più computer, senza che sia necessario che una copia di Access sia disponibile su ciascun computer. Questa edizione estesa dell'ambiente fornisce anche una certa capacità di configurazione. Senza l'edizione dello sviluppatore un file MDE è solo una versione compilata e compattata dell'applicazione, ma non può essere distribuita.

Sommario

In questo capitolo abbiamo esaminato due sistemi relazionali di gestione di basi di dati (RDBMS: relational database management system) molto diffusi e rappresentativi: Oracle e Microsoft Access. L'obiettivo era far conoscere al lettore la tipica architettura e le funzionalità di un prodotto di fascia alta come Oracle e di un RDBMS più piccolo come Access. Mentre Oracle può essere considerato un RDBMS completo, Access è uno strumento di gestione dei dati rivolto all'utente meno sofisticato. Abbiamo fatto una panoramica storica dello sviluppo dei sistemi relazionali di gestione delle basi di dati, poi abbiamo descritto l'architettura e le funzioni principali del sistema Oracle. Abbiamo visto come Oracle rappresenta internamente le basi di dati entrando nel dettaglio dell'organizzazione di memoria nel sistema. Sono stati quindi forniti alcuni esempi di programmazione in Oracle usando PL/SQL, il linguaggio di programmazione di Oracle con SQL incorporato, e PRO*C, un precompilatore per il linguaggio C. Abbiamo esaminato alcuni degli strumenti disponibili in Oracle per il progetto di basi di dati e lo sviluppo di applicazioni. Abbiamo infine dato una panoramica di Microsoft Access, della sua architettura, della definizione dei dati, della definizione delle relazioni e della manipolazione delle basi di dati usando QBE e SQL, e descritto alcune altre sue caratteristiche.

Bibliografia selezionata

Numerosi manuali descrivono il sistema Oracle, e particolarmente significativi sono quelli di Oracle editi nel 1997. Sunderraman (1999) è un buon punto di partenza per la programmazione in Oracle. L'editrice Oracle Press ha pubblicato molti libri sui diversi aspetti del sistema e persino un periodico intitolato *Oracle System Journal* che rende conto del costante sviluppo del prodotto. Anche su Access esistono molti manuali. Su come utilizzare il sistema Microsoft sono stati pubblicati diversi volumi, ma non è possibile darne qui un elenco completo.

cializzate e libri di successo hanno definito questo termine in più modi, mentre i fornitori ne hanno sfruttato la popolarità per mettere sul mercato prodotti eterogenei e i consulenti hanno messo a disposizione vari servizi, presentandoli tutti sotto l'etichetta collettiva dell'analisi multidimensionale dei dati. I data warehouse, tuttavia, si distinguono nettamente dalle basi di dati tradizionali per quanto riguarda struttura, funzionamento, prestazioni e obiettivi.¹

13.1.1 Terminologia e definizioni

W. H. Inmon² ha definito l'analisi multidimensionale dei dati come "una raccolta di dati integrata e permanente, ma focalizzata su un argomento e variabile nel tempo, che può fornire supporto alle decisioni di gestione". I data warehouse forniscono l'accesso ai dati necessario alle procedure di analisi complessa, di scoperta della conoscenza e di supporto alle decisioni.

Essi garantiscono elevate prestazioni nell'esecuzione di interrogazioni sulle informazioni e sui dati di un'organizzazione, fornendo l'ambiente esecutivo per parecchi tipi di applicazioni, come OLAP, DSS e le applicazioni di data mining. **OLAP (on-line analytical processing)** è il termine usato per descrivere l'analisi di dati prelevati dal data warehouse. Nelle mani di ingegneri della conoscenza esperti, gli strumenti OLAP possono sfruttare sistemi di elaborazione distribuiti per eseguire analisi che richiedono più memoria e potenza di calcolo di quante possa dare in modo conveniente ed efficiente una singola stazione di lavoro. I sistemi di supporto alle decisioni (DSS: decision-support systems) noti anche come sistemi informativi per dirigenti (EIS: executive information systems) (da non confondere con i sistemi di integrazione dei dati aziendali) supportano i decisori delle organizzazioni con dati aggregati per le decisioni complesse e importanti. Il data mining (che sarà presentato in dettaglio nel Paragrafo 13.2) è usato per la scoperta di conoscenza, il processo per ricercare all'interno dei dati conoscenza nuova e non prevista.

Le basi di dati tradizionali supportano l'**elaborazione delle transazioni in linea (OLTP: on-line transaction processing)**, che permette di eseguire inserimenti, aggiornamenti e cancellazioni, e anche interrogazioni, e sono progettate per elaborare in modo ottimale interrogazioni che coinvolgono una piccola parte della base di dati e transazioni che richiedono inserimenti o aggiornamenti di alcune tuple per relazione. Esse, quindi, non sono ottimizzate per finalità di OLAP, DSS o data mining. Al contrario, i data warehouse sono progettati proprio per supportare un'efficiente estrazione, elaborazione e presentazione dei dati per analisi e assunzione di decisioni. Rispetto alle basi di dati tradizionali, i data warehouse generalmente contengono grandi quantità di dati provenienti da più sorgenti che possono comprendere basi di dati di diversi modelli di dati e persino file provenienti da piattaforme e sistemi eterogenei.

¹ Nel Capitolo 1 si è definita una base di dati come una raccolta di dati correlati e un sistema di gestione di basi di dati come l'insieme costituito da una base di dati e dal suo software di gestione. Anche un data warehouse è una raccolta di informazioni con un sistema software di supporto. Esiste, comunque, una netta distinzione: le basi di dati tradizionali (relazionali, orientate agli oggetti, reticolari o gerarchiche) sono transazionali, i data warehouse, invece, sono intesi principalmente per applicazioni di supporto decisionale e sono ottimizzati per il recupero dei dati, non per l'elaborazione di transazioni di routine.

² Inoltre nel 1990 fu proposto il primo uso del termine "data warehouse".

13.1.2 Le caratteristiche dei data warehouse

Per presentare le caratteristiche dei data warehouse ed esplicarne le differenze rispetto alle basi di dati transazionali è necessario definire un modello di dati appropriato. Il modello di dati multidimensionale (spiegato più dettagliatamente in seguito) è specificamente concepito per gli strumenti OLAP e le tecnologie di supporto alle decisioni. Rispetto alle multibasi di dati che forniscono accesso a basi di dati disgiunte e normalmente eterogenee, il data warehouse è una raccolta di dati integrati provenienti da più sorgenti e memorizzati secondo un modello multidimensionale. Diversamente dalla maggior parte delle basi di dati transazionali, i data warehouse supportano tipicamente serie temporali e analisi di tendenza, due tecniche che richiedono più dati cronologici rispetto a quelli che di solito sono presenti nelle basi di dati transazionali. Rispetto a queste ultime, i data warehouse sono non volatili: ciò significa che le informazioni presenti in essi cambiano molto meno spesso, non in tempo reale, anche se possono essere soggette ad aggiornamenti periodici. Nei sistemi transazionali, le transazioni sono l'unità atomica e l'agente di aggiornamento della base di dati; al contrario, le informazioni dei data warehouse sono a granularità più grossa e vengono modificate secondo un'attenta politica di aggiornamento usualmente incrementale. Gli aggiornamenti sono gestiti dal componente di acquisizione del warehouse che esegue tutta la preelaborazione richiesta.

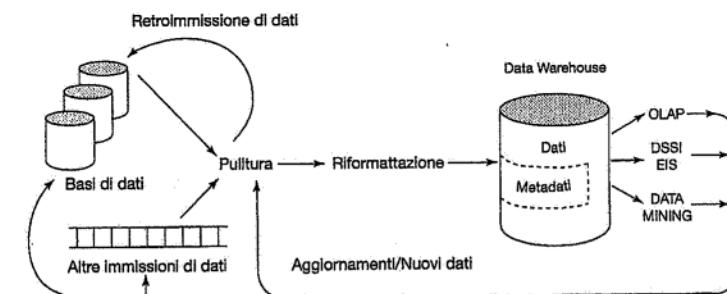


Figura 13.1 Il processo dell'analisi multidimensionale dei dati.

L'analisi multidimensionale dei dati può anche essere definita, in modo più generale, come "l'insieme delle tecnologie di supporto alle decisioni, progettate per consentire all'ingegnere della conoscenza (dirigente, manager, analista) di prendere decisioni migliori e più rapide".³ In Figura 13.1 viene fornita una panoramica della struttura concettuale di un data warehouse; mostrando l'intero processo di analisi multidimensionale dei dati. Questo processo comprende l'eventuale ripulitura e riformattazione dei dati prima dell'esecuzione dell'analisi

³ Chaudhuri e Dayal (1997) forniscono un eccellente saggio introduttivo sull'argomento, dando questa come definizione iniziale.

multidimensionale. Poi, OLAP, data mining e DSS possono servire a ricavare nuove importanti informazioni come regole di decisione o classificazione; queste informazioni tornano a far parte del data warehouse, come mostrato in figura. Le sorgenti possono essere comuni file di dati.

I data warehouse hanno le seguenti, fondamentali caratteristiche:⁴

- vista concettuale multidimensionale;
- dimensionalità generica;
- dimensioni e livelli di aggregazione non limitati a priori;
- operazioni tra una dimensione e l'altra non vincolate;
- gestione di matrici dinamiche sparse;
- architettura client/server;
- supporto multiutente;
- accessibilità;
- trasparenza;
- manipolazione intuitiva dei dati;
- prestazioni elevate nella generazione di report;
- struttura flessibile dei report.

Poiché racchiudono grandi quantità di dati, i data warehouse generalmente sono un ordine di grandezza (talvolta due ordini di grandezza) più grandi delle basi di dati tradizionali. Grandi volumi di dati (dell'ordine dei terabyte) possono essere gestiti attraverso:

- i **data warehouse a livello d'impresa**: si tratta di grandi infrastrutture che richiedono un massiccio investimento di tempo e di risorse;
- i **data warehouse virtuali**: forniscono viste su basi di dati operative che sono materializzate per fornire un accesso efficace;
- i **datamart**: di solito sono rivolti a un sottoinsieme dell'organizzazione, ad esempio un reparto, e sono più strettamente focalizzati.

13.1.3 Modelli dei dati per i data warehouse

I modelli multidimensionali usano le relazioni implicite presenti nei dati per inserirli in matrici multidimensionali chiamate cubi di dati (di solito chiamati ipercubi se hanno più di tre dimensioni). Per i dati conformi a questa operazione, le prestazioni delle interrogazioni su matrici multidimensionali possono essere migliori rispetto a quelle delle interrogazioni nel modello di dati relazionale. Tre esempi di dimensioni di un data warehouse aziendale sono i periodi fiscali, i prodotti e le regioni in cui opera l'azienda.

Un foglio elettronico standard è una matrice bidimensionale. Un esempio ne è un foglio elettronico delle vendite regionali di un prodotto in un determinato periodo di tempo: i pro-

		REGIONE		
		REG1	REG2	REG3
PRODOTTO	P123			
	P124			
	P125			
	P126			
	:			

Figura 13.2 Matrice bidimensionale.

dotti possono essere mostrati come righe, mentre le vendite per ciascuna regione costituiscono le colonne del foglio (in Figura 13.2 viene mostrata quest'organizzazione bidimensionale); aggiungendo una dimensione temporale, ad esempio i trimestri fiscali, si produce una matrice tridimensionale che può essere rappresentata usando un cubo di dati.

In Figura 13.3 vi è un cubo di dati tridimensionale che organizza i dati delle vendite dei prodotti per trimestri fiscali e per regioni: ciascuna cella contiene i dati per ciascun prodotto, trimestre fiscale e regione; inserendo ulteriori dimensioni si potrebbe creare un ipercubo di dati, sebbene le strutture con più di tre dimensioni non possano essere facilmente visualizzate o presentate graficamente.

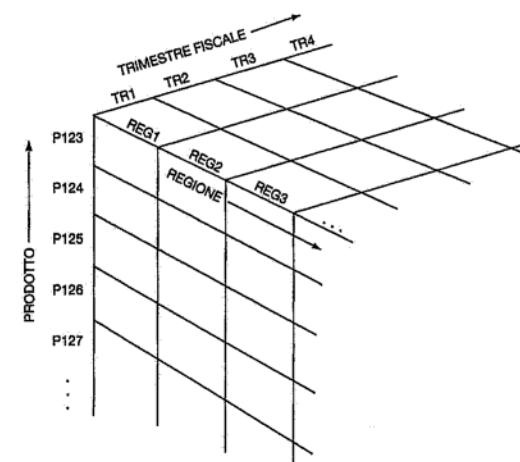


Figura 13.3 Un cubo di dati.

⁴ Codd (1993) coniò il termine OLAP e menzionò queste caratteristiche. L'elenco originale di Codd è stato qui riordinato.

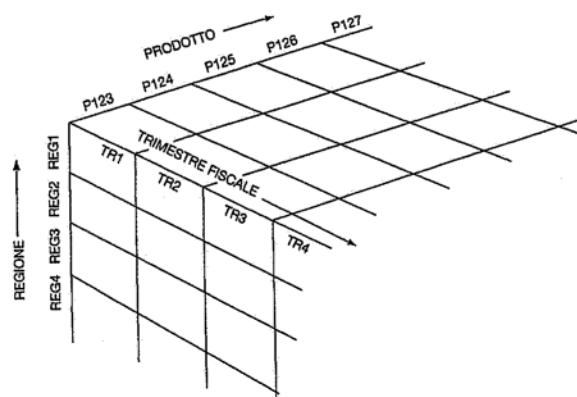


Figura 13.4 Versione ruotata del cubo di dati.

I dati possono essere interrogati direttamente in qualsiasi combinazione di dimensioni, evitando le usuali tecniche di interrogazione delle basi di dati. Esistono strumenti per visualizzare i dati secondo le dimensioni scelte dall'utente.

Il cambiamento da una gerarchia dimensionale a un'altra (orientamento) è facilmente attuabile in un cubo di dati usando un metodo chiamato tecnica del pivot o rotazione. Questa tecnica prevede di ruotare il cubo di dati, mostrando un diverso orientamento degli assi. Ad esempio, si può ruotare il cubo di dati per mostrare i ricavi delle vendite regionali come righe, i totali dei ricavi dei trimestri fiscali come colonne e i prodotti della società nella terza dimensione (Figura 13.4). Essa, quindi, equivale ad avere una tabella delle vendite regionali per ciascun prodotto separatamente; ciascuna tabella contiene le vendite trimestrali per quel prodotto regione per regione.

I modelli multidimensionali si prestano ottimamente alle visualizzazioni gerarchiche, in particolare a quelle conosciute come visualizzazione "roll-up" e visualizzazione "drill-down". La visualizzazione roll-up sale nella gerarchia, raggruppando lungo una dimensione unità più

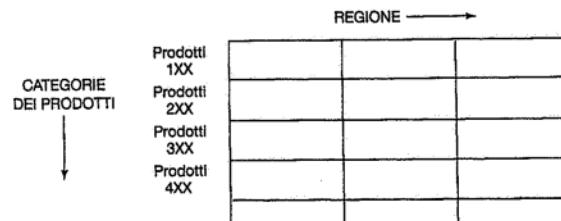


Figura 13.5 L'operazione roll-up.

		REGIONE 1					REGIONE 2	
		SOTTOREGIONE1	SOTTOREGIONE2	SOTTOREGIONE3	SOTTOREGIONE4	SOTTOREGIONE1		
STILI	P123	A						
		B						
STILI	P124	C						
		D						
STILI	P125	A						
		B						
STILI	P126	C						
		D						

Figura 13.6 L'operazione drill-down.

grandi (ad esempio riassumendo i dati settimanali per trimestre o per anno). In Figura 13.5 è mostrata una visualizzazione roll-up che passa da prodotti singoli a una granularità più grossolana di categorie di prodotti. In Figura 13.6 è mostrata una visualizzazione drill-down, che svolge la funzione opposta, fornendo una visualizzazione di maggior dettaglio, ad esempio, disaggregando le vendite regionali in sottoregioni o dividendo i prodotti secondo gli stili.

Il modello di memorizzazione multidimensionale definisce due tipi di tabelle: le tabelle delle dimensioni e le tabelle dei fatti, una **tabella della dimensione** consiste di tuple di attributi della dimensione, una **tabella dei fatti** può essere pensata come se fosse composta di tu-

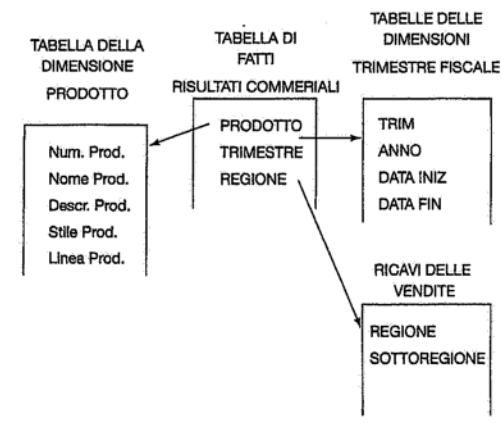


Figura 13.7 Uno schema a stella con le tabelle delle dimensioni e dei fatti.

Il termine metadati, introdotto nel Capitolo 2, è stato definito come la descrizione di una base di dati comprendente la definizione degli schemi. Il dizionario dei metadati è un componente fondamentale del data warehouse e comprende metadati sia tecnici sia gestionali: i metadati tecnici descrivono le operazioni di acquisizione, le strutture di memorizzazione, le descrizioni dei dati, il funzionamento del data warehouse e le funzionalità di supporto dell'accesso; i metadati gestionali descrivono le regole aziendali e gli aspetti organizzativi di supporto al warehouse.

L'architettura di come è distribuito il sistema di calcolo dell'organizzazione è una caratteristica determinante per il progetto del data warehouse. Vi sono due architetture base: il data warehouse distribuito e quello confederato. Nel progetto di un **data warehouse distribuito**, hanno grande importanza tutti gli aspetti relativi alle basi di dati distribuite, come la replica, il partizionamento, le comunicazioni e i problemi relativi all'uniformità e consistenza dei dati. Un'architettura distribuita può garantire elevate prestazioni del data warehouse, ad esempio sfruttando algoritmi evoluti di bilanciamento del carico e la scalabilità delle prestazioni. È necessario prevedere la replicazione dei metadati in ciascun sito di distribuzione. L'idea del **warehouse confederato** è analoga a quella di una base di dati confederata: una federazione decentralizzata di data warehouse autonomi, ciascuno con la propria raccolta di metadati. Tenendo presente lo sforzo richiesto per costruire un data warehouse, è probabile che tali federazioni consistano di componenti di scala più piccola, ad esempio singoli datamart. Le grandi organizzazioni possono scegliere di confederare datamart esistenti piuttosto che creare da zero grandi data warehouse.

13.1.5 Funzionalità tipiche dei data warehouse

I data warehouse esistono per rendere più immediate interrogazioni complesse, frequenti e ad hoc su grandi quantità di dati; di conseguenza devono fornire un supporto alle interrogazioni più efficiente e articolato di quanto sia richiesto alle basi di dati transazionali. Il componente di accesso ai data warehouse comprende funzionalità tipiche dei fogli elettronici, l'elaborazione efficiente delle interrogazioni, le interrogazioni strutturate e quelle ad hoc, il data mining e le viste materializzate. Ciò significa che il data warehouse può interagire con i più diffusi programmi di foglio elettronico (ad esempio MS Excel) e con le applicazioni OLAP. Questi ultimi offrono funzionalità preprogrammate come le seguenti:

- roll-up: i dati vengono riassunti diminuendo la granularità e aumentando la generalità (ad esempio da settimanalmente a trimestralmente ad annualmente);
- drill-down: i dati vengono mostrati con livelli di dettaglio crescenti (complementare a roll-up);
- pivot: viene eseguita la tabulazione incrociata (detta anche rotazione);
- slice e dice: esecuzione di operazioni di proiezione sulle dimensioni;
- ordinamento: i dati vengono ordinati;
- selezione: i dati sono resi disponibili per valore o intervallo;
- attributi derivati (calcolati): nuovi attributi sono calcolati eseguendo operazioni sui valori memorizzati e derivati.

Poiché i data warehouse non hanno le restrizioni tipiche dell'ambiente transazionale, vi è una maggior efficacia nell'elaborazione delle interrogazioni. Tra gli strumenti e le tecniche

usati vi sono: la trasformazione delle interrogazioni, l'unione e l'intersezione di indici, funzioni speciali ROLAP (OLAP relazionali) e MOLAP (OLAP multidimensionali), estensioni SQL, metodi di join avanzati e tecniche intelligenti di scansione (quali sono le interrogazioni multiple integrate in un'unica soluzione).

Le elevate prestazioni necessarie sono state raggiunte anche ricorrendo all'elaborazione parallela. Le architetture parallele utilizzate per il data warehouse comprendono le architetture a multiprocessori simmetrici (SMP: symmetric multiprocessor), cluster ed elaborazione ad alto parallelismo (MPP: massively parallel processing) e le loro combinazioni.

Gli ingegneri della conoscenza e i responsabili delle decisioni usano strumenti che vanno dalle interrogazioni parametriche alle interrogazioni ad hoc al data mining. Il componente di accesso del data warehouse, quindi, deve fornire il supporto necessario per le interrogazioni strutturate (sia parametriche sia ad hoc); prese insieme costituiscono un ambiente di interrogazioni assistito. Il data mining stesso utilizza tecniche di analisi statistica e d'intelligenza artificiale. L'analisi statistica può essere eseguita usando fogli elettronici avanzati, software di analisi statistica commerciali oppure ricorrendo a programmi personalizzati. Vengono usate comune mente anche tecniche come le medie mobili e l'analisi della regressione. Le tecniche di intelligenza artificiale come gli algoritmi genetici e le reti neurali sono utilizzati per la classificazione e per scoprire la conoscenza presente nel data warehouse che può essere inattesa e difficile da specificare nelle interrogazioni (il data mining sarà trattato in dettaglio nel Paragrafo 13.2).

Analisi multidimensionale dei dati e viste. Alcuni hanno considerato i data warehouse come un'estensione delle viste delle basi di dati. Come ricordato sopra, le viste materializzate sono considerate un metodo per soddisfare i requisiti di un migliore accesso ai dati (per ulteriori informazioni sulle viste si consulta il Capitolo 8). Le viste materializzate sono state studiate perché permettono di ottenere un miglioramento nelle prestazioni. Tuttavia, esse forniscono solo alcune delle funzioni dei data warehouse. Le viste e i data warehouse sono simili per il fatto che entrambi sono costituiti da dati in sola lettura estratti da basi di dati e focalizzati su uno specifico argomento, ma i data warehouse se ne differenziano per i seguenti motivi:

- richiedono una memorizzazione persistente invece di essere materializzati a richiesta;
- di solito sono multidimensionali, mentre le viste di una base di dati relazionale sono relazionali;
- possono essere indicizzati per ottimizzare le prestazioni, mentre le viste non possono essere indicizzate indipendentemente dalle basi di dati sottostanti;
- forniscono funzionalità specifiche, mentre le viste non possono farlo;
- gestiscono grandi quantità di dati integrati, spesso temporali, di solito più di quanti ne siano contenuti in una base di dati, mentre le viste sono un estratto e quindi un sottoinsieme di una base di dati.

13.1.6 Difficoltà di implementazione dei data warehouse

L'analisi multidimensionale dei dati comporta alcuni problemi operativi significativi: la costruzione, l'amministrazione e il controllo di qualità del data warehouse. La gestione del pro-

base di dati. SQL supporta operazioni di algebra relazionale che permettono a un utente di selezionare dati da tabelle (righe e colonne) oppure di collegare informazioni provenienti da più tabelle sulla base di campi comuni. Nel paragrafo precedente si è visto che la tecnologia di data warehouse fornisce oggi altri tipi di funzionalità, come l'aggregazione e la sintesi di dati; inoltre consente di visualizzare le informazioni su più dimensioni. In questo paragrafo si concentrerà l'attenzione su un'altra area di interesse molto popolare nota come "data mining". Il **data mining** si riferisce alla scoperta di nuove informazioni in forma di modelli o regole partendo da grandi quantità di dati. Per essere utile esso deve essere eseguito su file e basi di dati molto estesi. Fino a oggi le sue funzioni *non* sono ben integrate con i sistemi di gestione di basi di dati.

Verrà qui presentata una breve rassegna sulle conoscenze tecnico-scientifiche in questo campo, in cui ci si affida a tecniche prese da ambiti diversi, come l'apprendimento automatico, la statistica, le reti neurali e gli algoritmi genetici. Verrà posta in rilievo la natura delle informazioni scoperte, i tipi di problemi che si possono incontrare nelle basi di dati e le potenziali applicazioni. Si effettuerà una rassegna dello stato dell'arte di un grande numero di strumenti commerciali (si veda il Sottoparagrafo 13.2.5) e si descriveranno quali sono i progressi nella ricerca necessari al fine di rendere quest'area possibile.

13.2.1 Una visione d'insieme della tecnologia del data mining

In rapporti sulle tendenze di mercato come il famoso Rapporto Gartner,⁵ il data mining è stato indicato come una delle tecnologie più importanti per il prossimo futuro. In questo paragrafo il data mining viene messo in relazione al campo più vasto denominato "scoperta di conoscenza" (*knowledge discovery*), individuandone i rapporti per mezzo di un esempio illustrativo. Alcune tecniche e algoritmi di data mining saranno poi discussi nel Sottoparagrafo 13.2.3.

Data mining e data warehouse. Lo scopo del data warehouse è quello di aiutare il processo decisionale con dei dati. Il data mining può essere usato insieme al data warehouse per aiutare ad assumere alcuni tipi di decisioni e può essere applicato alle basi di dati funzionanti con transazioni singole. Per rendere il data mining più efficace, il data warehouse dovrebbe comprendere una raccolta di dati già aggregati o sintetizzati. Il data mining aiuta a estrarre nuovi modelli previsionali significativi che non sarebbe possibile trovare interrogando o elaborando i dati o i metadati nel data warehouse. Le applicazioni di data mining dovrebbero quindi essere prese in considerazione fin dai momenti iniziali della progettazione di un data warehouse. Inoltre gli strumenti di data mining dovrebbero essere progettati per essere usati insieme ai data warehouse. Nel caso di basi di dati molto estese che occupano terabyte di dati, infatti, il successo delle applicazioni di data mining dipende prima di tutto dalla disponibilità di un data warehouse.

Il data mining come parte del processo di scoperta di conoscenza. La scoperta di conoscenza nelle basi di dati, in genere abbreviata come KDD (knowledge discovery in databases), comprende tipicamente ben più del semplice data mining. Il processo di scoperta di conoscenza è costituito da sei fasi:⁶ la selezione, la ripulitura, l'arricchimento, la trasformazione o codifica dei dati, il data mining e infine la visualizzazione delle informazioni scoperte.

Come esempio si prenda in considerazione una base di dati transazionale mantenuta da una catena di rivendite al dettaglio di beni di consumo. Si supponga che i dati relativi ai clienti includano il nome del cliente, il codice di avviamento postale, il numero telefonico, la data d'acquisto, il codice del prodotto, il prezzo, la quantità e il totale. Eseguendo una procedura di KDD su questa base di dati di clienti è possibile estrarre molta conoscenza utile. Nella fase di *selezione dei dati* possono essere selezionati i dati che riguardano prodotti specifici o categorie di prodotti, oppure prodotti acquistati in negozi posti in una specifica regione o in un'area del paese. Il processo di *ripulitura dei dati* può correggere i codici di avviamento postale errati oppure eliminare le registrazioni con prefissi telefonici sbagliati. L'*arricchimento* di solito amplia i dati con ulteriori sorgenti d'informazione: ad esempio dati i nomi e i numeri telefonici dei clienti, la catena può acquistare altri dati sull'età, sul reddito e sul credito di cui godono e aggiungere questi dati a quelli già disponibili nelle registrazioni. La *trasformazione* e la codifica dei dati hanno invece lo scopo di ridurre la quantità di dati: ad esempio i codici degli articoli possono essere raggruppati in categorie di prodotto più vaste, come quella di prodotti audio, video, gadget elettronici, videocamere, accessori e così via; i codici di avviamento postale possono essere raggruppati in regioni geografiche, il reddito può essere classificato in dieci fasce e così via. In figura 13.1 è stata mostrata una fase chiamata ripulitura come passo preliminare della creazione del data warehouse. Se il data mining si basa su un warehouse già esistente per questa catena di negozi, ci si può aspettare che la ripulitura sia già stata applicata. È solo dopo tale preelaborazione che vengono applicate le tecniche del *data mining* per estrarre le regole e andamenti previsionali. Il risultato del data mining può scoprire:

- regole di associazione (ad esempio, ogni volta che un cliente acquista un'apparecchiatura video, compra anche un gadget elettronico);
- modelli sequenziali di comportamento (ad esempio, si supponga che un cliente acquisti una macchina fotografica, poi nel giro di tre mesi compri prodotti fotografici e nel giro di altri sei mesi un accessorio: un cliente che acquista più di due volte nei periodi morti, probabilmente compra almeno una volta durante il periodo di Natale);
- alberi di classificazione (ad esempio, i clienti possono essere classificati in base alla frequenza delle visite, secondo i tipi di finanziamento a cui ricorrono, per quantità di prodotti acquistati o per la propensione ad alcuni tipi di prodotti; sulla base della classificazione è possibile calcolare alcune statistiche rivelatrici).

Esistono, quindi, molte possibilità per scoprire nuova conoscenza sui modelli di acquisto mettendo in relazione fattori come l'età, il reddito, il luogo di residenza e quali e quanti prodotti vengono acquistati dai clienti. Queste informazioni possono poi essere utilizzate per localizzare nuove sedi di vendita in base alla presenza demografica, per eseguire promozioni,

⁵ Il Rapporto Gartner è un esempio delle molteplici pubblicazioni d'indagine tecnologica su cui i dirigenti d'azienda si basano per prendere decisioni.

⁶ Questa descrizione si basa ampiamente su Adriaans e Zantinge (1995).

per abbinare i prodotti nella pubblicità oppure per progettare strategie di vendita stagionali. L'esempio comunque mostra anche che il data mining deve essere preceduto da una significativa preparazione dei dati prima che possa fornire informazioni utili che possano influenzare direttamente le decisioni commerciali.

I risultati del data mining possono essere visualizzati in molti formati, quali elenchi, grafici, indici o visualizzazioni.

Scopi del data mining e della scoperta di conoscenza. In generale gli scopi del data mining rientrano nelle seguenti classi: previsione, identificazione, classificazione e ottimizzazione.

- **Previsione.** Il data mining permette di stimare l'andamento futuro di certi attributi dei dati. Esempi di data mining previsionale comprendono l'analisi delle transazioni di acquisto per predire ciò che i consumatori acquisteranno con certi sconti, il volume delle vendite che farà il negozio in un dato periodo e se l'eliminazione di una linea di prodotti porterà o meno maggiori profitti. In queste applicazioni la logica commerciale viene usata insieme al data mining. In un contesto scientifico i modelli di certe onde sismiche possono servire a predire con alta probabilità un terremoto.
- **Identificazione.** La presenza di andamenti ripetitivi nei dati può essere usata per identificare l'esistenza di un articolo, un evento o un'attività. Ad esempio, intrusi che cercano di violare un sistema informatico possono essere identificati attraverso i programmi che eseguono, i file a cui accedono e il tempo di CPU impiegato per sessione. Nelle applicazioni biologiche l'esistenza di un gene può essere identificata da alcune sequenze di nucleotidi nella sequenza del DNA. Anche l'area nota come *autenticazione* è una forma di identificazione. Ci si assicura se un utente è veramente chi dichiara di essere oppure proviene da una classe autorizzata; ciò richiede un confronto di parametri, immagini o segnali in una base di dati.
- **Classificazione.** Il data mining può suddividere i dati in modo da identificare diverse classi o categorie in base a combinazioni di parametri: ad esempio i clienti di un supermercato possono essere divisi in clienti alla ricerca di sconti, clienti frettolosi, clienti regolari e clienti occasionali. Tale classificazione può essere usata nelle successive analisi delle transazioni di acquisto dei clienti come attività post-mining. Talvolta la classificazione che si basa sulla conoscenza del dominio viene usata come input per suddividere il problema del data mining e renderlo più semplice da affrontare: ad esempio, gli alimenti biologici, gli stuzzichini per aperitivi e i cibi per le mense scolastiche sono categorie distinte nel commercio di alimentari del supermarket. È quindi consigliabile analizzare le relazioni tra le categorie come problemi separati. Questa classificazione empirica può essere utilizzata per codificare i dati in modo appropriato prima di sottoporli a ulteriore data mining.
- **Ottimizzazione.** Un possibile scopo del data mining è ottimizzare l'utilizzo di risorse limitate come il tempo, lo spazio, il denaro o i materiali massimizzando le variabili di output come le vendite o i profitti tenendo conto di alcuni vincoli. In questo caso lo scopo del data mining è simile alla funzione obiettivo usata nei problemi di ottimo vincolato tipici della ricerca operativa.

Il termine "data mining" è usato in senso molto ampio. In alcune situazioni comprende l'analisi statistica e l'ottimizzazione vincolata, oltre all'apprendimento automatico. Non vi è linea di demarcazione netta che separa il data mining da queste discipline. Va al di là delle fi-

nalità di questo libro, quindi, discutere dettagliatamente l'intero panorama di applicazioni che costituiscono questa vasta area di ricerca e di attività.

Tipi di conoscenza scoperti durante il data mining. Il termine "conoscenza" è interpretato in modo molto ampio e implica un certo grado di intelligenza. La conoscenza spesso è classificata in induttiva e deduttiva. Il data mining si occupa della conoscenza induttiva. La conoscenza può essere rappresentata in varie forme: in modo non strutturato, da regole oppure attraverso la logica proposizionale. In forma più strutturata può essere rappresentata usando alberi di decisione, reti semantiche, reti neurali o gerarchie di classi o strutture (frame). La conoscenza scoperta durante il data mining può essere descritta nei cinque modi seguenti.

1. **Regole di associazione.** Queste regole mettono in relazione la presenza di una serie di elementi con una serie di valori per un altro insieme di variabili. Esempi: (1) quando una donna acquista una borsetta, è probabile che compererà anche le scarpe; (2) un'immagine a raggi X che contiene le caratteristiche *a* e *b* è probabile che mostri anche la caratteristica *c*.
2. **Gerarchie di classificazione.** Lo scopo è elaborare un insieme di eventi o transazioni per creare una gerarchia di classi. Esempi: (1) una popolazione può essere divisa in cinque fasce di affidabilità in base alla storia delle richieste di credito precedenti; (2) è possibile sviluppare un modello dei fattori che determinano il posizionamento di un negozio in una certa zona su una scala 1 a 10; (3) i fondi comuni d'investimento possono essere classificati in base ai dati relativi alle prestazioni finanziarie che esprimono caratteristiche come le quotazioni, la rendita e la stabilità.
3. **Modelli di comportamento sequenziali.** Viene cercato un modello in grado di giustificare una sequenza di azioni o di eventi. Esempio: se un paziente è stato sottoposto a un intervento chirurgico di inserimento di un bypass cardiaco a causa delle arterie ostruite e di un aneurisma e nel giro di un anno si trova un alto livello di urea nel sangue, è probabile che entro 18 mesi soffra di insufficienza renale. La scoperta di modelli di comportamento sequenziali equivale alla scoperta di un'associazione tra eventi con determinati rapporti temporali.
4. **Strutture all'interno di serie temporali.** Si possono scoprire similitudini tra le posizioni delle serie temporali. Ecco tre esempi in cui il prezzo di mercato azionario costituisce una serie temporale: (1) le azioni della società elettrica Energia ABC e della società finanziaria Titoli XYZ mostrano lo stesso andamento durante il 1998 in termini di prezzo di chiusura delle azioni; (2) due prodotti presentano lo stesso andamento delle vendite in estate, ma un diverso andamento in inverno; (3) la configurazione del vento magnetico solare può essere usata per predire i cambiamenti nelle condizioni atmosferiche della Terra.
5. **Categorizzazione e segmentazione.** Un dato insieme di eventi o di elementi può essere partitionato (segmentato) in sottoinsiemi di elementi "simili". Esempi: (1) un intero gruppo di dati riguardanti la cura di una malattia può essere diviso in sottogruppi basati sulle similitudini degli effetti collaterali; (2) la popolazione adulta degli Stati Uniti può essere classificata in cinque gruppi, da "quello che acquista con probabilità più alta" a "quello che acquista con probabilità più bassa" un nuovo prodotto; (3) gli accessi eseguiti da un gruppo di utenti a una serie di documenti sul Web, ad esempio in una biblioteca digitale, possono essere analizzati in relazione alle parole chiave presenti nei documenti per rivelare categorie o gruppi di utenti.

Per la maggior parte delle applicazioni, la conoscenza desiderata è una combinazione dei tipi visti. Nei paragrafi seguenti si tratteranno in maggior dettaglio i tipi di conoscenza qui presentati.

13.2.2 Regole di associazione

Una delle più importanti tecnologie del data mining è quella relativa alla scoperta delle regole di associazione. La base di dati è considerata una raccolta di transazioni, ognuna delle quali coinvolge una serie di elementi. Un esempio comune è quello dei dati del panier d'acquisto. Il panier corrisponde a ciò che un consumatore acquista durante una visita a un supermercato. Si considerino le seguenti quattro transazioni di un campione casuale:

ID della transazione	Ora	Prodotti acquistati
101	6:35	latte, pane, succo di frutta
792	7:38	latte, succo di frutta
1130	8:05	latte, uova
1735	8:40	pane, biscotti, caffè

La forma di una regola di associazione è la seguente:

$$X \Rightarrow Y,$$

dove $X = \{x_1, x_2, \dots, x_n\}$, e $Y = \{y_1, y_2, \dots, y_m\}$ sono due insiemi di elementi, con x_i e y_j che sono elementi distinti per qualunque i e j . Questa associazione specifica che se un cliente acquista X , è probabile che acquisti anche Y . In genere qualsiasi regola di associazione ha la forma LHS (left-hand side, parte sinistra) \Rightarrow RHS (right-hand side, parte destra), dove LHS e RHS sono insiemi di elementi. Le regole di associazione dovrebbero garantire insieme il supporto e il livello di confidenza.

Il supporto per la regola LHS \Rightarrow RHS è la percentuale di transazioni che coinvolgono tutti gli elementi nell'unione, l'insieme $LHS \cup RHS$. Se il supporto è basso, implica che non esiste una schiacciatrice evidenza che gli elementi in $LHS \cup RHS$ si verifichino insieme, perché l'unione accade solo in una piccola frazione di transazioni. La regola Latte \Rightarrow Succo di frutta ha un supporto del 50%, mentre Pane \Rightarrow Succo di frutta ha un supporto solo del 25%. Un altro termine che indica il supporto è la prevalenza di una regola.

Per calcolare il livello di confidenza si prendano in considerazione tutte le transazioni che includono gli elementi di LHS. Il livello di confidenza della regola di associazione LHS \Rightarrow RHS è la percentuale di tali transazioni che includono RHS rispetto al totale delle transazioni. Un altro termine sinonimo del livello di confidenza è la forza della regola.

Per Latte \Rightarrow Succo di frutta il livello di confidenza è 66,7%, il che significa che delle tre transazioni in cui vi è il latte, due contengono anche il succo, e Pane \Rightarrow Succo di frutta ha il livello di confidenza del 50%, il che significa che una delle due transazioni che contengono il pane contiene anche il succo.

Come si può vedere il supporto e il livello di confidenza non vanno necessariamente di pari passo. Lo scopo della ricerca di regole di associazione del data mining è creare tutte le re-

gole possibili che superino un valore di soglia minima, specificato dall'utente, di supporto e di livello di confidenza. Il problema viene quindi suddiviso in due sottoproblemi:

1. generare tutti gli insiemi di elementi, detti **insiemi estesi**, che hanno un supporto che oltrepassa il livello del valore di soglia (si noti che il termine "esteso" in questo caso indica un grande supporto);
2. per ciascun insieme esteso, tutte le regole che hanno almeno il livello di confidenza minimo sono create come segue: dato un insieme esteso X e $Y \subset X$ [si legga "Y è un sottoinsieme proprio di X"], sia $Z = X - Y$; allora se supporto (X)/supporto (Z) \Rightarrow livello di confidenza minimo, la regola $Z \Rightarrow Y$ (cioè $X - Y \Rightarrow Y$) è una regola valida.

Creare le regole usando tutti gli insiemi estesi e i loro supporti è relativamente semplice. Scoprire, tuttavia, tutti gli insiemi estesi insieme al valore del loro supporto è un problema complesso se la cardinalità dell'insieme di elementi è molto alta. Un supermercato ha in genere migliaia di prodotti: il numero dei sottoinsiemi di prodotti distinti è 2^m , dove m è il numero dei prodotti, e calcolare il supporto per tutti i possibili insiemi di prodotti diventa computazionalmente molto oneroso.

Per ridurre lo spazio di ricerca combinatoriale, gli algoritmi per trovare le regole di associazione devono avere le proprietà seguenti:

- un sottoinsieme di un insieme esteso deve essere a sua volta esteso (ad esempio ogni sottoinsieme di un insieme esteso deve superare il supporto minimo richiesto);
- al contrario, anche l'estensione di un piccolo insieme deve essere piccola (il che implica che non abbia supporto sufficiente).

La seconda proprietà aiuta a eliminare un insieme da ogni ulteriore considerazione di estensione se risulta essere troppo piccolo.

Algoritmi base per trovare le regole di associazione. Gli algoritmi utilizzati per identificare insiemi estesi sono progettati per operare come descritto qui di seguito.

1. Si esamina il supporto degli insiemi di cardinalità 1, chiamati 1-insiemi, esaminando in dettaglio la base di dati, e si eliminano quelli che non soddisfano il supporto minimo richiesto.
2. Si estendono gli 1-insiemi estesi per ottenere i 2-insiemi aggiungendo un elemento alla volta in modo da generare tutti i possibili candidati di cardinalità 2. Si verifica il supporto per tutti gli insiemi candidati esaminando in dettaglio la base di dati ed eliminando i 2-insiemi che non soddisfano il supporto minimo.
3. Si ripetono le fasi precedenti; alla fase k , i $(k-1)$ insiemi precedentemente trovati vengono estesi per ottenere i k -insiemi, e controllati per valutare il supporto minimo.

Il procedimento viene ripetuto fino a quando si riescono a trovare insiemi estesi. La versione più semplice di questo algoritmo, tuttavia, è un vero incubo combinatorio. Molti algoritmi più sofisticati sono stati proposti per ricavare le regole di associazione. Essi differiscono tra loro soprattutto in relazione a come vengono creati gli insiemi candidati e a come sono calcolati i loro supporti.

Alcuni algoritmi utilizzano strutture di dati evolute come mappe di bit (*bitmaps*) e alberi hash per mantenere le informazioni sugli insiemi. Sono stati proposti diversi algoritmi che utilizzano più scansioni della base di dati perché il numero potenziale di insiemi, 2^m , può essere trop-

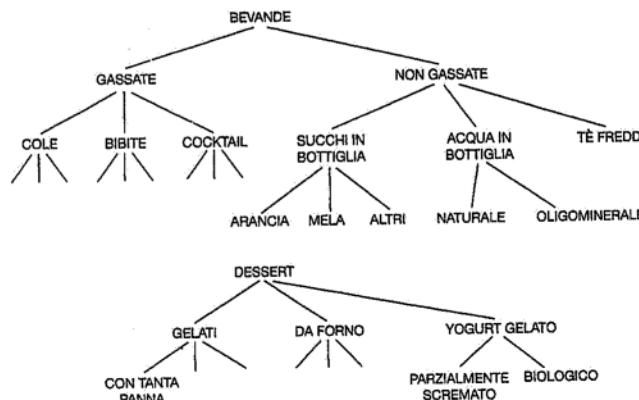


Figura 13.10 Una tassonomia dei prodotti di un supermercato.

po grande per i contatori se si esegue una singola scansione. Gli autori di questo libro hanno proposto un loro algoritmo, chiamato **algoritmo di partizione**,⁷ qui brevemente presentato.

Se viene data una base di dati con un piccolo numero di potenziali insiemi estesi, ad esempio alcune migliaia, il loro supporto può essere esaminato in una scansione usando una tecnica di partizione. La partizione divide la base di dati in parti disgiunte. Ogni parte viene presa in considerazione come base di dati separata, creando tutti gli insiemi estesi relativi a una parte in un solo passo. Alla fine della prima scansione, quindi, si ottiene l'elenco degli insiemi estesi di tutte le parti che compongono la partizione. Quando questi elenchi vengono unificati, si scopre che contengono alcuni falsi positivi, cioè alcuni degli insiemi considerati sono estesi rispetto a una parte ma possono non qualificarsi come tali in molte altre parti e quindi non superare il supporto minimo rispetto alla base di dati originale. Si noti che non esistono falsi negativi, cioè nessun insieme esteso verrà perso. L'unione di tutti gli insiemi estesi identificati nel passo uno sono usati come input del passo due come insiemi candidati e viene calcolato il loro supporto effettivo per l'*intera* base di dati. Alla fine della fase due sono stati identificati tutti gli insiemi estesi effettivi. La partizione viene eseguita in modo che tutte le parti possano essere collocate in memoria principale e una parte venga letta una volta sola in ogni fase. L'algoritmo di partizione, per il quale sono stati suggeriti ulteriori miglioramenti,⁸ si presta all'implementazione parallela, per efficienza.

Regole di associazione tra gerarchie. Esistono alcuni tipi di associazioni particolarmente interessanti perché si verificano tra gerarchie distinte di elementi. A volte è possibile attribuire

gli elementi da classificare a gerarchie disgiunte in base alla natura del dominio: ad esempio, i cibi in un supermercato, i prodotti in un grande magazzino oppure gli articoli in un negozio sportivo possono essere classificati in classi e sottoclassi che danno origine a gerarchie. Si prenda in considerazione la Figura 13.10 che mostra la tassonomia dei prodotti di un supermercato. Essa mostra due gerarchie, rispettivamente di bevande e dessert: i gruppi completi possono anche non creare associazioni della forma bevande => dessert, oppure dessert => bevande.

Le associazioni del tipo yogurt gelato della linea biologica => acqua in bottiglia oppure gelato della linea con tanta panna => tè freddi possono produrre un livello di confidenza e un supporto tali da garantire la loro validità come regole di associazione.

Allora, se l'area applicativa ha una classificazione naturale degli insiemi in gerarchie, non è di particolare interesse scoprire associazioni *all'interno* delle gerarchie. Ben più interessanti sono le associazioni *tra* una gerarchia e l'altra, che possono verificarsi tra gruppi di elementi posti a diversi livelli.

Associazioni negative. Il problema della scoperta di un'associazione negativa è più impegnativo rispetto a quello della scoperta di un'associazione positiva. Un'associazione negativa è del seguente tipo: "il 60% dei clienti che ha acquistato patatine non acquista acqua in bottiglia" (qui il 60% si riferisce al livello di confidenza per la regola di associazione negativa). In una base di dati con 10.000 elementi, esistono $2^{10.000}$ possibili combinazioni di elementi, la maggioranza dei quali non appare neanche una volta nella base di dati. Se l'assenza di una determinata combinazione di elementi inducesse un'associazione negativa, si avrebbero a livello potenziale milioni e milioni di regole di associazione negativa, con RHS di nessun interesse. Il problema è allora trovare solamente le regole negative *interessanti*. In generale si è interessati ai casi in cui due specifici insiemi di elementi appaiono molto raramente nella stessa transazione. Ciò pone due problemi.

- Dato un catalogo di 10.000 prodotti, la probabilità che due qualsiasi di essi vengano acquistati insieme è $(1/10.000)*(1/10.000) = 10^{-8}$. Anche se si scopre che il supporto effettivo dell'evento dell'acquisto congiunto è zero, questo non significa necessariamente un allontanamento significativo dalle aspettative, e quindi non è un'associazione negativa interessante.
- L'altro problema è più serio: si stanno cercando combinazioni di prodotti con supporto molto basso, e ci sono milioni di combinazioni con un supporto basso o anche uguale a zero: ad esempio, un insieme di 10 milioni di transazioni relative a 10.000 prodotti genera più di 2,5 miliardi di combinazioni mancanti che corrispondono a miliardi di regole inutili.

Di conseguenza, per trovare regole di associazione negative interessanti, si deve utilizzare la conoscenza disponibile sugli insiemi di elementi. Un approccio è l'utilizzo delle gerarchie. Si supponga di usare le gerarchie di patatine e di bevande analcoliche mostrate in Figura 13.11. Tra le bevande analcoliche e le patatine è già stata indicata una forte associazione positiva. Se si trova un supporto ampio per il fatto che quando i clienti acquistano patatine Days di solito acquistano anche la bevanda Topsy, e *non* Joke e *non* Wakeup, ciò sarebbe interessante. Questo si verifica perché di solito ci si aspetta che se esiste un'associazione forte tra

⁷ Si veda Savasere e altri (1995) per ulteriori dettagli sull'algoritmo, sulle strutture di dati usate per implementarlo e per una valutazione delle sue prestazioni.

⁸ Si consulti Cheung e altri (1996) e Lin e Dunham (1998).

Scoperta di modelli di comportamento in serie temporali. Le serie temporali sono sequenze di eventi; ciascun evento può essere un dato tipo fisso di una transazione: ad esempio, il prezzo di chiusura di un'azione o di un fondo d'investimento è un evento che si verifica ogni giorno lavorativo per ciascuna azione e ciascun fondo; la sequenza di questi valori per azione o fondo costituisce una serie temporale. All'interno di una serie temporale è possibile cercare configurazioni significative analizzando sequenze e sottosequenze come si è fatto precedentemente: ad esempio si può trovare un periodo durante cui l'azione è salita o si è mantenuta stabile per n giorni, oppure trovare il periodo più lungo in cui l'azione ha avuto una fluttuazione di non più di 1% sul prezzo di chiusura precedente oppure il trimestre in cui l'azione ha avuto il maggiore guadagno o la maggiore perdita percentuale. Le serie temporali possono essere confrontate stabilendo delle misure di similitudine per identificare le società le cui azioni si comportano in modo simile. L'analisi e il mining delle serie temporali è un'estensione delle funzionalità delle basi di dati temporali.

Scoperta di regole di classificazione. La classificazione è il processo di apprendimento di una funzione che classifica un dato oggetto di interesse in una di molte classi possibili. Le classi possono essere predefinite oppure determinate durante il compito di classificazione. Un semplice esempio è il seguente. Una banca desidera classificare chi fa richiesta di un mutuo nelle categorie di quelli che meritano di avere il prestito e di quelli che non lo meritano. Per fare questo si può utilizzare una semplice regola che dice che se la rata mensile del mutuo (che è un elemento noto per ciascun richiedente) supera il 25% delle entrate nette mensili (un altro elemento noto per il richiedente), il richiedente appartiene alla classe dei "non meritevoli del mutuo"; altrimenti appartiene alla classe di "chi può usufruire del mutuo".

In generale le regole di classificazione possono essere più complesse e della forma seguente:

(var₁ nell'intervallo₁) e (var₂ nell'intervallo₂) e ... (var_n nell'intervallo_n) => l'Oggetto O appartiene alla classe C_i.

È necessario stabilire una serie simile di regole per ciascuna classe.

Le variabili var₁, ..., var_n sono gli attributi dell'oggetto O e costituiscono le colonne di una relazione con una tupla per oggetto. Ciascuna tupla di questa tabella deve essere attribuita a una classe. In questo modo è possibile scrivere una serie di interrogazioni SQL che converte la popolazione della tabella negli elementi delle classi, una volta che queste classi sono definite. Il problema del mining è scoprire le classi e le condizioni che le definiscono.

La principale differenza tra la scoperta delle regole viste precedentemente e la scoperta delle regole di associazione è che le variabili nelle regole precedenti prendono i valori da un dominio discreto o continuo (ad esempio il numero di bambini, le entrate in dollari), mentre nel caso di regole di associazione gli insiemi sono formati da una serie di elementi predefiniti. Inoltre, una regola di associazione si riferisce a un insieme di transazioni (registrazioni di input), mentre una regola di classificazione specifica come attribuire ciascuna registrazione a una classe.

Regressione. La regressione è un'applicazione speciale della regola di classificazione. Se una regola di classificazione viene definita come una funzione calcolata sulle variabili che attri-

buisce le variabili stesse a una classe di destinazione, la regola è detta **regola di regressione**. Un'applicazione generale della regressione si verifica quando, invece di attribuire una tupla di dati di una relazione a una classe specifica, il valore di una variabile viene previsto in base a quella tupla. Ad esempio si prenda in considerazione la relazione

LAB_TESTS (ID paziente, esame 1, esame 2, ..., esame n)

contenente valori che sono i risultati da una serie di n esami di laboratorio eseguiti su un paziente. La variabile di destinazione che si desidera prevedere è P , la probabilità di sopravvivenza del paziente. La regola di regressione prende la forma seguente:

(esame 1 nell'intervallo₁) e (esame 2 nell'intervallo₂) e ... (esame n nell'intervallo_n) => $P = x$ oppure $x < P < y$

A volte è possibile predire un valore unico per P , mentre in altri casi bisogna accontentarsi di un intervallo di valori. Se ci si riferisce a P come a una funzione:

$P = f(\text{esame 1, esame 2, ..., esame n})$

la funzione è detta **funzione di regressione** per prevedere P . In generale, se la funzione ha la forma

$y = f(x_1, x_2, \dots, x_n)$

e f è lineare nelle variabili di dominio x_i , il processo di derivazione di f da un dato insieme di tuple per $\langle x_1, x_2, \dots, x_n, y \rangle$ è chiamato **regressione lineare**. La regressione lineare è una tecnica statistica comunemente usata per adattare un insieme di osservazioni o di punti in n -dimensioni rispetto alla variabile di destinazione y .

L'analisi di regressione è uno strumento molto comune per l'analisi di dati in molti domini di ricerca. La scoperta della funzione per predire la variabile di destinazione equivale a un'operazione di data mining.

Reti neurali. Le reti neurali sono una tecnica derivata dalla ricerca in intelligenza artificiale che utilizza la regressione generalizzata e fornisce un metodo iterativo per esegirla. Esse realizzano l'adattamento progressivo di una curva per ricavare una funzione da un insieme di campioni. Tale tecnica costituisce un "approccio di apprendimento": è governata da un campione di esempi che viene usato per l'apprendimento e l'inferenza iniziali. Con questo metodo di apprendimento, le risposte a nuovi input possono essere interpolate dagli esempi noti. Questa interpolazione dipende dal modello della realtà d'interesse (rappresentazione interna del dominio del problema) adottata dal metodo di apprendimento.

Le reti neurali possono essere classificate in due grandi categorie: le reti con supervisione e quelle senza supervisione. I metodi adattativi che tentano di ridurre l'errore in output sono metodi che realizzano un **apprendimento con supervisione**, mentre quelli che sviluppano rappresentazioni interne senza output ottenuti a seguito di campionamento, sono chiamati **metodi ad apprendimento senza supervisione**.

ze e l'analisi statistica. Vengono utilizzati anche gli alberi di decisione, una rappresentazione delle regole utilizzata nella classificazione o nel clustering, e le analisi statistiche, che possono comprendere la regressione e molte altre tecniche. Altri prodotti commerciali usano tecniche avanzate come gli algoritmi genetici, tecniche di ragionamento basato su studio di casi, le reti Bayesiane, la regressione non lineare, l'ottimizzazione combinatoriale, il pattern matching e la logica fuzzy.

La maggior parte degli strumenti di data mining usa l'interfaccia ODBC (Open DataBase Connectivity), uno standard industriale che permette di operare con le basi di dati e consente l'accesso ai dati nella maggior parte dei programmi di basi di dati più diffusi, come Access, dBASE, Informix, Oracle e SQL Server. Alcuni di questi pacchetti software forniscono interfacce a specifici programmi di basi di dati; i più comuni sono Oracle, Access e SQL Server. La maggior parte degli strumenti funziona nell'ambiente Microsoft Windows, ma alcuni di essi sono disponibili anche per il sistema operativo UNIX. La tendenza è quella di fare in modo che tutti i prodotti siano in grado di operare nell'ambiente Microsoft Windows.

In generale questi programmi vengono eseguiti in modo sequenziale su una singola macchina e molti funzionano in modalità client/server. Alcuni prodotti incorporano funzionalità di elaborazione parallela per essere eseguiti su architetture parallele e possono interoperare con altri strumenti di elaborazione analitica in linea (OLAP).

Interfaccia utente. La maggior parte degli strumenti interagisce con l'utente tramite un'interfaccia utente grafica (GUI). Alcuni prodotti ricorrono a tecniche di visualizzazione sofisticate per visualizzare dati e regole, ad esempio MineSet di SGI, e sono anche in grado di manipolare i dati interattivamente. Le interfacce a caratteri sono meno usate e più comuni negli strumenti per UNIX come Intelligent Miner di IBM.

Interfaccia per programmi applicativi. La disponibilità di un'interfaccia per programmi applicativi (API: application programming interface) non è sempre garantita. La maggior parte dei prodotti di data mining non permette l'utilizzo delle loro funzioni interne. Alcuni di essi, tuttavia, consentono ai programmatore di applicazioni di riutilizzare il loro codice. La maggior parte delle interfacce applicative sono costituite da librerie C e da librerie di collegamenti dinamici (DLL: dynamic link libraries). Alcuni strumenti includono linguaggi di comandi di basi di dati proprietari.

In Tabella 13.1 sono elencati 10 strumenti di data mining particolarmente rappresentativi. Oggi vi sono più di 70 prodotti commerciali di data mining disponibili in tutto il mondo. Tra i prodotti non americani vi sono Data Surveyor, che è olandese, e Polyanalyst, che è russo.

Sviluppi futuri. Gli strumenti di data mining sono in continua evoluzione sulla base delle idee sviluppate dalla ricerca scientifica più recente. Molti di essi incorporano algoritmi messi a punto nell'intelligenza artificiale (AI: artificial intelligence), nella ricerca statistica e nell'ottimizzazione.

L'uso di tecniche moderne di basi di dati, come l'elaborazione distribuita, su architetture client/server, in basi di dati parallele e nell'analisi multidimensionale dei dati garantisce buone prestazioni. Per il futuro la tendenza è decisamente quella dello sviluppo delle capacità di et... Inoltre gli approach ibridi diventeranno più comuni e l'elaborazione verrà eseguita

Tabella 13.1 Alcuni strumenti di data mining particolarmente rappresentativi.

Società	Prodotto	Tecnica	Piattaforma	Interfaccia*
Acknosoft	Kate	Alberi di decisione, tecniche di ragionamento basate su studio di casi	Win NT UNIX	Microsoft Access
Angoss	Knowledge Seeker	Alberi di decisione, tecniche statistiche	Win NT	ODBC
Business Objects	Business Miner	Reti neurali, apprendimento automatico	Win NT	ODBC
CrossZ	QueryObject	Tecniche di ottimizzazione di analisi statistica	Win NT MVS UNIX	ODBC
Data Distilleries	Data Surveyor	Repertorio completo di tecniche, uso integrato di vari algoritmi DM	UNIX	ODBC ODMG
IBM	Intelligent Miner	Classificazione, regole di associazione, modelli di predizione	UNIX (AIX)	IBM DB2
Megaputer Intelligence	Polyanalyst	Acquisizione della conoscenza simbolica, programmazione evolutiva	Win NT OS/2	ODBC Oracle DB2
NCR	Management Discovery Tool (MDT)	Regole di associazione	Win NT	ODBC
SAS	Enterprise Miner	Alberi di decisione, regole di associazione, reti neurali, regressione, clustering	UNIX (Solaris) Win NT Macintosh	ODBC Oracle AS/400
Silicon Graphics	MineSet	Alberi di decisione regole di associazione	UNIX (Irix)	Oracle Sybase Informix

*ODBC: Open DataBase Connectivity; ODMG: Object Data Management Group

usando tutte le risorse disponibili e traendo vantaggio sia dagli ambienti di calcolo distribuiti sia da quelli paralleli. Questa evoluzione è importante soprattutto perché le basi di dati moderne contengono quantità molto grandi di informazioni. Le basi di dati multimediali stanno diventando più diffuse, ma il recupero e la memorizzazione delle immagini sono ancora operazioni lente. Il costo della memoria di massa sta però diminuendo e il massiccio immagazzinamento di informazioni sarà utilizzabile anche da piccole società. I futuri programmi di data mining dovranno quindi essere in grado di operare su più grandi insiemi di dati di più organizzazioni.

In un prossimo futuro probabilmente i sistemi operativi Microsoft Windows NT e UNIX saranno le piattaforme standard, con una predominanza di Windows. La maggior parte del software di data mining userà ODBC come standard per estrarre i dati dalle basi di dati aziendali; i formati di input proprietari sono destinati a scomparire.

Vi è una precisa necessità di eseguire il data mining su dati non standard, tra cui immagini e altri dati multimediali. Gli sviluppi algoritmici per il data mining su dati non standard non hanno però ancora raggiunto un livello di maturità sufficiente per la commercializzazione.

Sommario

In questo capitolo abbiamo esaminato due aspetti molto importanti della tecnologia delle basi di dati che avranno un ruolo significativo nel prossimo decennio: l'analisi multidimensionale dei dati e il data mining.

L'analisi multidimensionale dei dati può essere vista come un processo che richiede diverse attività preliminari; il data mining può essere pensato come l'attività che estrae la conoscenza da un data warehouse di dati. Sono stati introdotti concetti fondamentali relativi all'analisi multidimensionale dei dati, discutendo le particolari funzionalità della visualizzazione multidimensionale. Abbiamo trattato le modalità con cui i data warehouse forniscono a chi deve prendere decisioni informazioni al livello corretto di dettaglio, in base a un'organizzazione e a una prospettiva appropriate.

Abbiamo esaminato anche un esempio eloquente di scoperta della conoscenza nelle basi di dati, un ambito più vasto rispetto al data mining. Per il data mining, tra le varie tecniche, si è dato particolare rilievo alle regole di associazione, fornendo un'introduzione dettagliata alla loro scoperta, ma non sono stati trascurati le reti neurali e gli algoritmi genetici.

La ricerca è oggi impegnata sia sul fronte dell'analisi multidimensionale dei dati sia su quello del data mining, e sono state sottolineate alcune delle direzioni di ricerca più promettenti. L'attenzione, infine, è stata rivolta al futuro mercato dei prodotti della tecnologia delle basi di dati. Abbiamo esaminato 10 dei 70 strumenti di data mining oggi disponibili; ci si aspetta che la ricerca futura ne aumenti significativamente il numero e le funzionalità.

Esercizi

- 13.1 Cos'è un data warehouse? In cosa differisce da una base di dati?
- 13.2 Si definiscano i termini: *OLAP* (Online Analytical Processing), *ROLAP* (Relational OLAP), *MOLAP* (multidimensional OLAP), *DSS* (Decision Support Systems).
- 13.3 Si descrivano le caratteristiche di un data warehouse, suddividendole tra le funzionalità di un data warehouse e i vantaggi che gli utenti ne derivano.
- 13.4 Cos'è un modello multidimensionale di dati? Come viene usato nel data warehousing?
- 13.5 Si definiscano i seguenti termini: *schema a stella*, *schema a fiocco di neve*, *costellazione dei fatti*, *datamart*.

- 13.6 Quali tipi di indici vengono usati per i data warehouse? Si illustrino gli utilizzi di ciascuno con un esempio.
- 13.7 Si descrivano le fasi di creazione di un data warehouse.
- 13.8 Quali considerazioni giocano un ruolo importante nella progettazione di un data warehouse?
- 13.9 Si descrivano le attività che un utente può eseguire su un data warehouse e si illustriano i risultati di queste attività su un data warehouse multidimensionale.
- 13.10 Qual è il concetto di vista relazionale in relazione ai data warehouse e ai datamart? In cosa differiscono?
- 13.11 Si elenchino i problemi implementativi di un data warehouse.
- 13.12 Si elenchino le questioni aperte e i problemi di ricerca nel settore dei data warehouse.
- 13.13 Cos'è il data mining? Come si collega tale tecnologia a quella del data warehouse?
- 13.14 Quali sono le diverse fasi della scoperta di conoscenza nelle basi di dati? Si descriva uno scenario applicativo completo in cui nuova conoscenza può essere estratta da una base di dati transazionale esistente.
- 13.15 Quali sono gli scopi che il data mining si propone?
- 13.16 Quali sono i cinque tipi di conoscenza prodotti dal data mining?
- 13.17 Cosa sono le regole di associazione come tipo di conoscenza? Si diano le definizioni di supporto e di livello di confidenza e le si utilizzi per definire una regola di associazione.
- 13.18 Si descriva una regola di associazione tra gerarchie con un esempio.
- 13.19 Cos'è una regola di associazione negativa nel contesto della gerarchia di Figura 13.10?
- 13.20 Quali sono le difficoltà di estrazione delle regole di associazione da basi di dati estese?
- 13.21 Si definisca un modello di comportamento sequenziale che si basi sulle sequenze di Hemsets.
- 13.22 Si dia un esempio di una configurazione ripetitiva in una serie temporale.
- 13.23 Cosa sono le regole di classificazione? Quale legame c'è tra regressione e classificazione?
- 13.24 Si descrivano le reti neurali e gli algoritmi genetici visti come tecniche per il data mining. Quali sono le principali difficoltà nel loro utilizzo?
- 13.25 Si descrivano il clustering e la segmentazione come tecniche di data mining.
- 13.26 Quali sono le caratteristiche principali degli strumenti di data mining? Si valutino le funzioni di uno strumento del data mining non menzionato nell'elenco di Tabella 13.1.

Bibliografia selezionata

L'analisi multidimensionale dei dati negli ultimi anni è stata oggetto di molte pubblicazioni. Si attribuisce a Inmon (1992) l'aver fatto accettare in modo diffuso il termine "analisi multidimensionale". Codd (1993) ha reso popolare il termine OLAP (online analytical processing) e ha definito una serie di caratteristiche dell'analisi multidimensionale dei dati atte a fornire supporto OLAP. Mattison (1996) fornisce una rassegna completa delle tecniche dell'analisi multidimensionale dei dati disponibili presentando le strategie che le società dovrebbero utilizzare nel loro sviluppo. Bischoff e Alexander (1997) offrono una raccolta di consigli forniti

ni sono caratterizzate dalla loro natura "statica": una situazione in cui l'utente finale può solo recuperare dati dalla base di dati, mentre l'aggiornamento con ulteriori informazioni è limitato agli esperti che controllano e analizzano i nuovi dati che vengono immessi.

14.1 Basi di dati e World Wide Web

Il World Wide Web (WWW), conosciuto da tutti come "il Web", fu sviluppato in origine in Svizzera presso il CERN¹ all'inizio degli anni novanta come un sistema ipermidia² su larga scala che consentiva ai biologi di condividere le informazioni. Oggi questa tecnologia permette l'accesso universale a informazioni condivise a chiunque abbia l'accesso su Internet: essa contiene centinaia di milioni di pagine alla portata di milioni di utenti.

Nella tecnologia Web tutte le attività si basano su un'architettura client/server. Le informazioni sono memorizzate in calcolatori denominati server Web in file condivisi e accessibili pubblicamente, codificati attraverso il linguaggio HTML (hypertext markup language). Molti strumenti consentono agli utenti di creare pagine Web formattate con i marcatori (tag) HTML, includendovi anche contenuti multimediali, sia immagine sia audio o video. Una pagina Web contiene molti collegamenti ipertestuali; letteralmente un collegamento ipertestuale è un collegamento che permette a un utente di passare da un punto all'altro all'interno di una pagina o muoversi da una pagina all'altra attraverso Internet (in inglese "browse"). Tale possibilità ha permesso agli utenti finali notevole libertà nella ricerca e nella "navigazione" tra informazioni correlate, memorizzate su server posti in paesi o in continenti diversi.

Le informazioni sul Web sono organizzate tramite il localizzatore universale di risorse (URL: uniform resource locator). Un URL è simile a un indirizzo che fornisce il percorso completo per la localizzazione di un file. Il percorso di ricerca consiste in una stringa di nomi di directory e computer separati da barre oblique e termina con il nome di un file. Ad esempio, il sommario di questo libro si trova all'URL seguente:

<http://www.awl.com/cseng/authors/clmasri/Dbase3e.html>

Un URL inizia sempre con un protocollo per il trasferimento di ipertesti (http: hypertext transport protocol) che indica il protocollo usato dai browser Web, programmi che comunicano con i server Web e viceversa. I browser Web interpretano e presentano i documenti HTML agli utenti. Tra i browser Web più diffusi vi sono Internet Explorer di Microsoft e Netscape Navigator. Una raccolta formata da documenti HTML e da altri file accessibili attraverso l'URL su un server Web viene definita un sito Web. Nell'URL precedente "www.awl.com" indica il sito Web dell'editore Addison Wesley.

¹ CERN è acronimo francese che sta per "Conseil Européen pour la Recherche Nucléaire", cioè Consiglio Europeo per la Ricerca Nucleare.

² Questa idea d'incredibile successo è attribuita a Berners-Lee e al suo gruppo; si veda Berners-Lee e altri (1992, 1994).

14.1.1 Accesso Web alle basi di dati

La tecnologia odierna si è spostata rapidamente dalle pagine Web statiche a quelle dinamiche in cui il contenuto può variare in continuazione. Il server Web usa un'interfaccia standard chiamata interfaccia comune di gateway (CGI: Common Gateway Interface) che funge da middleware: si tratta di uno strato software aggiuntivo posto tra l'interfaccia utente e il motore della base di dati che consente l'accesso a basi di dati eterogenee. Il middleware CGI esegue programmi esterni o script per ottenere le informazioni dinamiche e restituisce le informazioni al server in formato HTML; quest'ultimo provvede poi a inviarle al browser che le ha richieste.

In questo modo, è diventato necessario consentire agli utenti l'accesso non solo ai file presenti nel file system, ma anche alle basi di dati e ai DBMS, supportando l'elaborazione delle interrogazioni, la creazione di resoconti e così via. Gli approcci esistenti possono essere divisi in due categorie.

Accesso usando script CGI. Si può far interagire il server di gestione della base di dati con il server Web attraverso l'interfaccia CGI. In Figura 14.1 è mostrato uno schema di architettura per l'accesso alla base di dati via Web attraverso script CGI scritti in linguaggi quali PERL, Tcl o C. Il principale svantaggio di questo approccio è che per ogni richiesta utente il server Web deve attivare un nuovo processo CGI: ogni processo crea una nuova connessione con il DBMS e il server Web deve aspettare finché non gli sono forniti i risultati. Raggruppando le richieste di più utenti, non si ottiene quindi alcun vantaggio; tra l'altro, lo sviluppatore deve memorizzare gli script solo nelle sottodirectory CGI-bin, fatto che rende possibile delle intru-

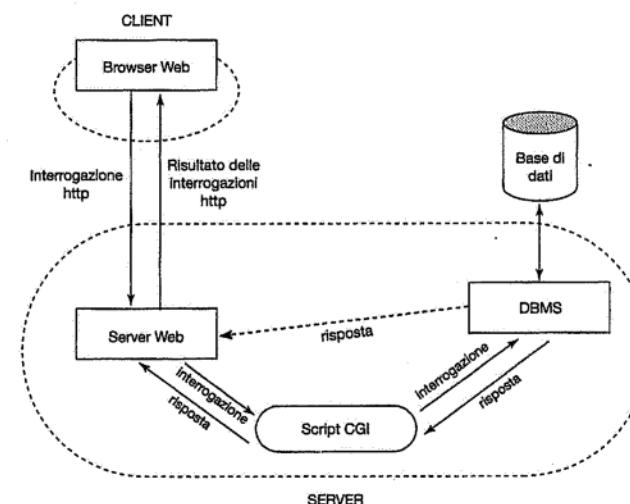


Figura 14.1 Accesso alle basi di dati via Web attraverso gli script CGI.

sioni e perdita di sicurezza. Il fatto che CGI non abbia un linguaggio standard, ma richieda agli sviluppatori di applicazioni di basi di dati di imparare PERL o Tcl è un altro aspetto negativo, così come la gestione degli script, soprattutto se sono sparsi ovunque.

Accesso usando JDBC. JDBC è un insieme di classi Java sviluppato da Sun Microsystems per consentire l'accesso alle basi di dati relazionali tramite l'esecuzione di interrogazioni SQL. Si tratta di un modo di accedere alle basi di dati che non richiede l'attivazione di processi aggiuntivi per ogni richiesta ulteriore da parte di un client. Si noti che JDBC è un nome registrato come marchio depositato dalla Sun e che, al contrario di quanto molti credono, non è l'acronimo di Java Data Base Connectivity. JDBC permette di connettersi a una base di dati, inviare istruzioni SQL e recuperare i risultati dell'interrogazione usando rispettivamente le classi Java Connection, Statement e ResultSet. Visto che Java è indipendente dalla piattaforma, l'applicazione può girare su qualsiasi browser in grado di eseguire Java, cioè di scaricare il codice Java dal server ed eseguirlo sul browser del client. Il codice Java è trasparente al DBMS; sono i driver JDBC specifici per specifici DBMS sul lato server che interagiscono con lo specifico DBMS usato dall'applicazione. Se il driver JDBC si trova sul client, l'applicazione gira interamente su quest'ultimo e le sue richieste sono comunicate al DBMS direttamente dal driver. Per interrogazioni standard SQL, si può accedere a molti RDBMS. Il lato negativo dell'utilizzo di JDBC è la necessità di eseguire Java usando macchine virtuali a bassa efficienza. Il ponte da JDBC a ODBC (Open Database Connectivity) è un altro modo per interrogare i RDBMS.

In aggiunta a CGI, altri fornitori di server Web stanno lanciando i loro prodotti middleware per permettere la connessione simultanea a più basi di dati. Tra questi vi sono ISAPI (Internet Server Application Programming Interface, Interfaccia di Programmazione Applicativa del Server Internet) di Microsoft e NSAPI (Netscape API di Netscape). Nel prossimo sottoparagrafo viene descritta la tecnica di accesso al Web fornita da Informix. Altri produttori DBMS forniscono già o forniranno soluzioni simili per supportare l'accesso alle basi di dati sul Web.

14.1.2 Tecnica di integrazione Web di INFORMIX

Informix ha affrontato le limitazioni di CGI e le incompatibilità tra CGI, NSAPI e ISAPI creando WIO (Web Integration Option, tecnica d'integrazione Web) che elimina la necessità degli script. Gli sviluppatori usano appositi strumenti per creare direttamente all'interno della base di dati pagine HTML intelligenti, dette Pagine Applicative. Queste eseguono istruzioni SQL in modo dinamico, ristrutturano i risultati in HTML e restituiscono la pagina Web risultante agli utenti finali. L'architettura è mostrata in modo schematico in Figura 14.2. WIO utilizza il Driver Web, un processo CGI "leggero" che è attivato automaticamente quando una richiesta URL viene ricevuta dal server Web. Anche se viene generato un identificatore univoco di sessione per ciascuna richiesta, l'applicazione WIO è persistente e *non* termina dopo ogni richiesta.

Quando riceve una richiesta dal driver Web, l'applicazione WIO si connette alla base di dati ed esegue Web Explode, una funzione che esegue le interrogazioni SQL presenti all'interno delle pagine Web e ristruttura i risultati come pagina Web restituendoli al browser attraverso il driver Web.

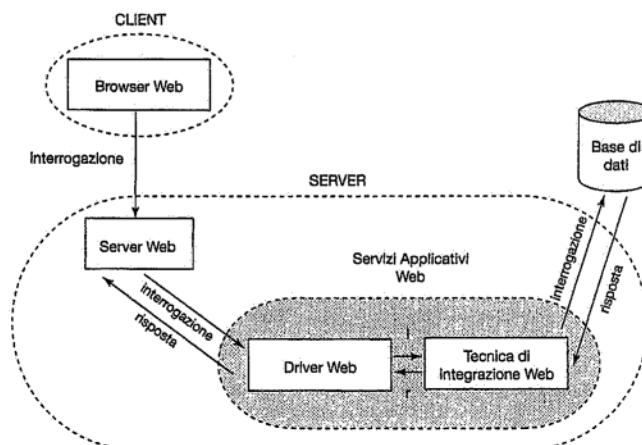


Figura 14.2 Implementazione CGI della tecnica di integrazione Web di Informix.

Le estensioni ai marcatori HTML proposte da Informix consentono agli autori di siti Web di creare applicazioni che possono costruire in modo dinamico schemi di riferimento (*templates*) per pagine Web con il Server Dinamico di Informix e presentarli agli utenti finali.

WIO consente anche agli utenti di creare marcatori personalizzati per eseguire compiti specifici. In questo modo si possono progettare applicazioni complesse, senza dover ricorrere allo sviluppo di script o di programmi. Un'altra caratteristica di WIO è la possibilità di scrivere applicazioni transazionali utilizzando un'interfaccia di programmazione applicativa (API: Application Programming Interface) che offre una serie di servizi transazionali di base, come la gestione delle sessioni e la connessione, che possono essere incorporate nell'applicazione Web.

WIO supporta applicazioni sviluppate in C, C++ e Java. Questa flessibilità consente agli sviluppatori di portare le applicazioni esistenti sul Web o di svilupparne di nuove in tali linguaggi. WIO è uno strumento software integrato con il server Web e utilizza il meccanismo di sicurezza nativo del Server Dinamico di Informix. L'architettura aperta di WIO consente l'uso di diversi server Web e browser.

14.1.3 WebServer di ORACLE

ORACLE supporta l'accesso Web alle basi di dati attraverso i componenti mostrati in Figura 14.3. Il client richiede dei file che sono chiamati "statici" o "dinamici" dal server Web. I file statici hanno un contenuto prefissato, mentre il contenuto di quelli dinamici può includere i risultati di interrogazioni SQL alla base di dati. L'architettura proposta da Oracle comprende un daemon HTTP (un processo che è in continua esecuzione) chiamato Web Listener che si avvia sul

Indicizzazione delle immagini. Vi sono due approcci per indicizzare le immagini: identificare gli oggetti presenti nelle immagini automaticamente utilizzando tecniche di elaborazione di immagini e assegnare frasi e termini indice attraverso l'indicizzazione manuale. Un problema importante nell'utilizzo delle tecniche di elaborazione di immagini per indicizzare le figure è legato alla scalabilità. Lo stato attuale della tecnologia consente l'indicizzazione automatica solo di semplici motivi geometrici. La complessità aumenta con il numero di caratteristiche riconoscibili. Un'altra questione rilevante è collegata alla complessità dell'interrogazione. Le regole e i meccanismi di inferenza possono essere usati per dedurre da semplici caratteristiche delle immagini dei fatti a livello superiore. Ciò consente di incorporare interrogazioni di alto livello come "trova gli alberghi che hanno atri aperti che consentono la massima esposizione al sole nell'area riservata alla reception" in un'applicazione architettonica. Attualmente l'indicizzazione delle immagini si basa su uno dei tre schemi di indicizzazione seguenti.

1. *Sistemi classificatori.* Classificano le immagini gerarchicamente in categorie predeterminate. In questo approccio l'indicizzatore e l'utente dovrebbero avere una buona conoscenza delle categorie disponibili. Non possono essere colti e registrati i dettagli più fini di un'immagine complessa e le relazioni tra gli oggetti in un'immagine.
2. *Sistemi basati su parole chiave.* Utilizzano un vocabolario di indicizzazione simile a quello usato nell'indicizzazione dei documenti testuali. Possono essere registrati semplici fatti rappresentati nell'immagine (ad esempio "una regione avvolta dalla calotta di ghiaccio") e i fatti derivati come risultato di un'interpretazione ad alto livello da parte di esseri umani (come la distinzione tra ghiaccio permanente, una recente nevicata e il ghiaccio polare).
3. *Sistemi entità-attributo-relazione.* Vengono identificati tutti gli oggetti nell'immagine e le relazioni fra gli oggetti e gli attributi degli oggetti.

Si noti che nel caso di documenti testuali un indicizzatore può scegliere le parole chiave da una serie di parole disponibili nel documento che deve essere indicizzato, ciò che non è possibile nel caso di dati video e visivi.

Problemi del recupero di dati testuali. Si tratta da sempre dell'elemento chiave nelle applicazioni gestionali e nei sistemi di gestione delle biblioteche, e sebbene molto lavoro sia già stato fatto, rimane la necessità di ulteriori miglioramenti, soprattutto per quanto riguarda le questioni seguenti.

- *Indicizzazione di frasi.* Miglioramenti sostanziali possono essere realizzati se ai documenti vengono assegnati descrittori a frasi (a differenza dei termini a indice, costituiti da singole parole), purché le frasi prescelte siano buoni indicatori del contenuto dei documenti.
- *Utilizzo di un vocabolario dei sinonimi o di un thesaurus.* Una causa della scarsa capacità di richiamo (recall) dei sistemi correnti è che il vocabolario usato dall'utente è diverso da quello usato per indicizzare i documenti. Una soluzione è utilizzare un vocabolario dei sinonimi o un thesaurus per ampliare l'interrogazione dell'utente con termini aggiuntivi collegati.
- *Risoluzione dell'ambiguità.* Uno dei motivi della scarsa precisione (il rapporto del numero di oggetti pertinenti recuperati rispetto al numero totale degli oggetti recuperati) nei sistemi di recupero delle informazioni è che nel linguaggio naturale molte parole hanno più significati. Un modo per risolvere l'ambiguità è usare un dizionario in linea; un altro è confrontare i contesti in cui le due parole si presentano.

primi anni dello sviluppo dei DBMS, più o meno dal 1965 al 1995, l'applica-

zione principale è stata la gestione di dati industriali e gestionali, che sono principalmente numerici. Nei prossimi anni le informazioni testuali e non numeriche probabilmente domineranno il contenuto delle basi di dati. Di conseguenza, ai DBMS verranno aggiunte molte funzioni relative al confronto, alla concettualizzazione, alla comprensione, all'indicizzazione e alla produzione di sintesi dei documenti. I sistemi di gestione delle informazioni multimediali permetteranno di arrivare a un collegamento tra discipline che storicamente sono state aree separate: il recupero delle informazioni e la gestione di basi di dati.

14.2.4 Applicazioni delle basi di dati multimediali

È prevedibile che applicazioni di basi di dati multimediali di grandi dimensioni interessino un gran numero di discipline e quindi migliorino le proprie funzionalità. Di seguito sono elencate alcune applicazioni importanti.

- *Gestione delle registrazioni e dei documenti.* Molte industrie e organizzazioni tengono registrazioni molto dettagliate e producono una grande quantità di documenti. I dati possono comprendere dati di produzione e progetti meccanici, cartelle mediche dei pazienti, materiale destinato alla pubblicazione e registrazioni delle compagnie di assicurazione.
- *Diffusione della conoscenza.* L'editoria multimediale, un mezzo molto efficace di diffusione della conoscenza, comprenderà una crescita fenomenale dei libri elettronici, dei cataloghi, dei manuali, delle encyclopedie e delle raccolte di informazioni sui più diversi argomenti.
- *Didattica e formazione professionale.* Materiale d'insegnamento per tutti gli ordini di scuola, dagli studenti elementari agli operatori di strumentazione ai professionisti, può essere progettato partendo da fonti multimediali. Le biblioteche digitali avranno in futuro una grande influenza sul modo in cui studenti e ricercatori, ma anche altri utenti, avranno accesso a vaste raccolte di materiale didattico (si veda il Paragrafo 14.6 sulle biblioteche digitali).
- *Vendite, pubblicità, distribuzione, intrattenimento e viaggi.* In queste applicazioni non vi sono limiti all'utilizzo di informazioni multimediali: si va da efficaci presentazioni di merci in vendita alle gite virtuali in città e in gallerie d'arte. L'industria cinematografica ha già dimostrato la potenza degli effetti speciali nella creazione di animazioni, animali e alieni progettati artificialmente. L'utilizzo di oggetti preprogettati memorizzati in basi di dati multimediali amplierà il campo di tali applicazioni.
- *Controllo e monitoraggio in tempo reale.* La presentazione multimediale delle informazioni, insieme alla tecnologia delle basi di dati attive, può essere uno strumento molto efficace per monitorare e controllare compiti complessi come operazioni industriali, centrali nucleari, pazienti in unità mediche intensive e sistemi di trasporto.

Sistemi commerciali per la gestione delle informazioni multimediali. Non vi è alcun DBMS progettato con l'unico scopo di gestire dati multimediali, e quindi non ve n'è nessuno che abbia il campo di funzionalità richiesto per supportare pienamente tutte le applicazioni di gestione delle informazioni multimediali che si sono esaminate precedentemente. Molti DBMS, tuttavia, supportano oggi tipi di dati multimediali, fra questi il Server Dinamico Informix (Informix Dynamic Server), la base di dati universale (TIDB: Universal Database) DB2 della IBM, Oracle 8.0

2. la base di dati è distribuita tra componenti cablati e unità mobili; la responsabilità di gestione dei dati è condivisa tra stazioni base e unità mobili.
- I problemi relativi alla gestione dei dati distribuiti possono essere applicati anche alle basi di dati mobili con le considerazioni e variazioni seguenti.
1. *Replicazione e distribuzione dei dati.* I dati vengono distribuiti in modo non uniforme tra stazioni base e unità mobili. La necessità di mantenere la consistenza tra i dati memorizzati sulle unità mobili e quelli presenti alle stazioni base pone un problema di gestione delle memorie cache. Le cache cercano di fornire i dati a cui si accede e che vengono modificati più frequentemente alle unità mobili, che eseguono le loro transazioni e possono essere disconnesse per lunghi periodi.
 2. *Modelli di transazioni.* Gli usuali problemi di tolleranza degli errori e correttezza delle transazioni sono più critici nell'ambiente mobile. Una transazione mobile viene eseguita sequenzialmente attraverso molte stazioni base e su diverse copie dei dati multipli, a seconda del movimento dell'unità mobile. Manca, quindi, un coordinamento centrale nell'esecuzione della transazione, in particolare nello scenario 2 sopra illustrato. Può capitare che le proprietà ACID tradizionali delle transazioni, debbano essere modificate definendo nuovi modelli di transazione.
 3. *Elaborazione delle interrogazioni.* Sapere dove si trovano i dati è importante e influenza l'analisi costi/benefici dell'elaborazione delle interrogazioni. In ambiente mobile, la risposta alle interrogazioni deve essere fornita a unità mobili che possono essere in movimento o che stanno attraversando i confini delle celle, tuttavia devono ricevere i risultati delle interrogazioni completi e corretti.
 4. *Ripristino e tolleranza degli errori.* L'ambiente delle basi di dati mobili deve saper gestire malfunzionamenti nella comunicazione, nei siti, nei dispositivi e nelle transazioni. I malfunzionamenti del sito per un MU sono spesso causati dalla limitata autonomia delle batterie. Se poi una MU viene volontariamente spenta, questo evento *non* dovrebbe essere gestito come un malfunzionamento. I fallimenti delle transazioni sono più frequenti quando una MU passa da una cella all'altra. I malfunzionamenti delle MU causano la partizione della rete e influenzano gli algoritmi di instradamento.
 5. *Progetto di basi di dati mobili.* Il problema globale della risoluzione dei nomi per gestire le interrogazioni è particolarmente complesso a causa della mobilità e dei frequenti spostamenti delle unità. Per questo motivo, il progetto delle basi di dati mobili deve incorporare tecniche avanzate per la gestione dei metadati, come il costante aggiornamento delle informazioni relative alla posizione delle unità mobili.

14.3.4 Basi di dati mobili sincronizzate a intermittenza

Il telelavoro basato sulla comunicazione mobile sta diventando sempre più comune e sono in molti a svolgere il proprio lavoro lontano dagli uffici e dalle abitazioni ricorrendo a unità mobili: si pensi ad attività come la vendita di beni di consumo, industriali e farmaceutici, le consulenze e pianificazioni finanziarie e assicurative, le attività relative alla gestione di proprietà e beni immobili e così via. In queste applicazioni un server o un gruppo di server gestisce una base

di dati centrale e gli utenti si servono di computer portatili o palmari con un software di gestione di basi di dati residente per eseguire transazioni che restano "locali" per la maggior parte del tempo. I client si collegano con i server attraverso una rete o una connessione telefonica (o attraverso Internet), tipicamente per una sessione breve, ad esempio da 30 a 60 minuti. Inviano i loro aggiornamenti al server, che deve a sua volta immetterli nella sua base di dati centrale la quale deve tenere i dati aggiornati e preparare copie appropriate per tutti i client sul sistema. Ogni volta che i client si connettono, ricevono, attraverso un processo noto come sincronizzazione del client con il server, una serie di aggiornamenti che devono essere eseguiti sulla loro base di dati locale. La caratteristica principale di questo scenario è che i client sono per lo più sconnessi dalla rete e il server non è necessariamente in grado di raggiungerli. Questo ambiente pone problemi simili a quelli delle basi di dati distribuite e delle basi di dati client/server e alcuni anche delle basi di dati mobili, ma presenta anche alcuni problemi aggiuntivi di ricerca che vanno studiati. Esso è detto **ambiente di basi di dati sincronizzate a intermittenza (ISDBE: Intermittently Synchronized Database Environment)** e le basi di dati corrispondenti vengono definite basi di dati sincronizzate a intermittenza (ISDB: Intermittently Synchronized Database).

Le caratteristiche seguenti contraddistinguono gli ambienti ISDB rispetto alle basi di dati mobili finora esaminate:

1. un client si connette al server quando vuole ricevere o inviare aggiornamenti oppure eseguire transazioni che necessitano di dati non locali; questa comunicazione può essere *unicast*, cioè uno-a-uno tra il server e il client, oppure *multicast*, quando un server comunica periodicamente con una serie di riceventi o aggiorna le basi di dati locali di un gruppo di client;
2. un server non può collegarsi a un client a piacimento;
3. i problemi delle connessioni dei client mobili, al contrario delle connessioni dei client cablati e del mantenimento dell'alimentazione elettrica, sono problemi di scarso rilievo;
4. un client è libero di gestire i propri dati e di eseguire transazioni locali mentre non è connesso, e può inoltre effettuare delle funzionalità di ripristino;
5. un client ha più modi per collegarsi a un server: nel caso ve ne siano molti a disposizione, può scegliere un particolare server per connettersi in base alla distanza, alla disponibilità di nodi o ad altri fattori.

A causa di tali differenze, molti problemi relativi agli ISDB sono diversi da quelli dei sistemi di basi di dati mobili. Tra questi vi è il progetto della base di dati del server per le basi di dati server, la gestione della consistenza fra le basi di dati del client e del server, l'elaborazione delle transazioni e degli aggiornamenti, l'utilizzo efficace della larghezza di banda disponibile al server e la scalabilità negli ambienti ISDB.

14.3.5 Bibliografia selezionata per le basi di dati mobili

Negli ultimi cinque-sei anni c'è stata un'improvvisa ondata d'interesse per il calcolo mobile con una crescita significativa della ricerca su basi di dati mobili. Tra i libri scritti su questo argomento, Dhawan (1997) è un'eccellente fonte di informazioni sul calcolo mobile. Le reti di comunicazione mobile e il loro futuro vengono discusse in Holtzman e Goodman (1993). Imielinski e Badrinath (1994) forniscono una buona panoramica dei problemi relativi alle ba-

si di dati in ambiente mobile. Dunham e Helal (1995) trattano i problemi di elaborazione delle interrogazioni, distribuzione dei dati e gestione delle transazioni per le basi di dati mobili. Foreman e Zahorjan (1994) descrivono le funzionalità e i principali problemi del calcolo mobile e offrono argomenti convincenti a suo favore, in quanto possibile soluzione in futuro per molte applicazioni di sistemi informativi. Pitoura e Samaras (1998) illustrano tutti gli aspetti dei problemi e delle soluzioni delle basi di dati mobili. Chintalapati e altri (1997) descrivono un algoritmo adattivo di gestione delle posizioni, mentre Bertino e altri (1998) considerano gli approcci al ripristino e alla tolleranza degli errori nelle basi di dati mobili. Il numero di giugno 1995 della rivista *Byte* affronta diversi aspetti del calcolo mobile. Per una discussione iniziale sul problema della scalabilità degli ISDB e la descrizione di un suo approccio attraverso l'aggregazione di dati e il raggruppamento di client si consulti Mahajan e altri (1998).

14.4 Sistemi informativi geografici

I sistemi informativi geografici (GIS: geographic information systems) vengono usati per raccogliere, modellare, memorizzare e analizzare le informazioni che descrivono le proprietà fisiche del mondo geografico. L'ambito dei GIS comprende due tipi di dati: (1) i dati spaziali che hanno origine da mappe, immagini digitali, confini politici e amministrativi, strade, reti di trasporto, nonché i dati fisici quali i fiumi, le caratteristiche del terreno, le regioni climatiche; (2) i dati non spaziali come conteggi del censimento, dati economici, informazioni sulle vendite e così via. I GIS rappresentano attualmente un campo in rapido sviluppo che offre approcci altamente innovativi per rispondere ad alcune richieste tecniche di difficile soluzione.

14.4.1 Applicazioni GIS

È possibile dividere i GIS in tre categorie: applicazioni cartografiche, applicazioni digitali di modellazione del terreno e applicazioni per le gestioni di oggetti geografici (Figura 14.5).

Nelle applicazioni di modellazione del terreno e cartografiche vengono registrate le variazioni negli attributi spaziali, ad esempio le caratteristiche del suolo, la densità del raccolto e la qualità dell'aria. Nelle applicazioni per la gestione di oggetti geografici, gli oggetti d'interesse sono identificati all'interno di un dominio fisico, ad esempio centrali elettriche, distretti elettorali, appezzamenti di proprietà, zone di distribuzione dei prodotti e confini delle città. Questi oggetti geografici sono correlati con i dati di altre applicazioni che per questo specifico esempio possono essere il consumo energetico, i risultati elettorali, i volumi di vendita delle proprietà e dei prodotti e la densità del traffico.

I primi due tipi di applicazioni GIS consigliano una rappresentazione basata su campi, mentre la terza categoria ne esige una basata sugli oggetti. L'approccio cartografico, inoltre, richiede funzioni speciali, come la sovrapposizione di mappe di diversi livelli per calcolare i dati di attributi che consentiranno, ad esempio, la misurazione di nuove distanze nello spazio tridimensionale e la riclassificazione dei dati sulla mappa. La modellazione digitale del ter-

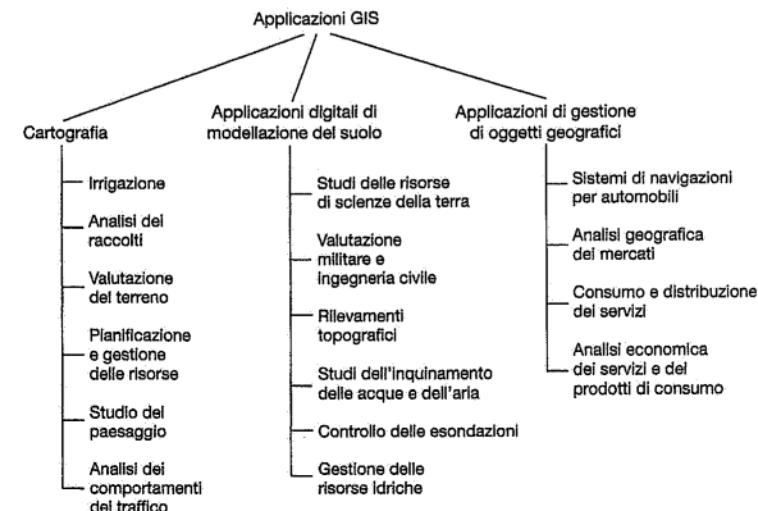


Figura 14.5 Una possibile classificazione delle applicazioni GIS (adattata da Adam e Gangopadhyay, 1997)

no richiede una rappresentazione digitale di parti della superficie terrestre, usando l'altezza rilevata in punti campione che vengono poi collegati per fornire un modello della superficie terreste come una rete tridimensionale (linee collegate in 3D).

La modellazione digitale del terreno richiede l'impiego di funzioni d'interpolazione tra i punti osservati e la visualizzazione. Nelle applicazioni geografiche basate su oggetti, sono necessarie delle funzioni aggiuntive per trattare i dati relativi a strade, oleodotti, cavi di comunicazione, linee elettriche e così via. Ad esempio per una data regione, possono essere usate delle mappe confrontabili per mostrare i cambiamenti in diversi momenti di certi dati come quelli della localizzazione delle strade, dei cavi, delle costruzioni e dei corsi d'acqua.

14.4.2 Requisiti di gestione dei dati dei GIS

I requisiti funzionali delle applicazioni GIS descritti precedentemente si traducono nei seguenti requisiti relativi alle basi di dati.

Rappresentazione e modellazione dei dati. I dati GIS possono essere rappresentati in due formati: vettoriale e raster. I dati vettoriali rappresentano oggetti geometrici come punti, linee e poligoni; ad esempio un lago può essere rappresentato come un poligono e un fiume con una serie di segmenti lineari. I dati raster sono caratterizzati da una matrice di punti in cui ciascuno rappresenta il valore di un attributo per una posizione del mondo reale. In modo non formale si può

dire che le immagini raster sono matrici n -dimensionali in cui ogni elemento è un'unità dell'immagine e rappresenta un attributo. Le unità bidimensionali vengono dette *pixel*, mentre quelle tridimensionali sono chiamate *voxel*. I dati di elevazione tridimensionali sono memorizzati in un formato raster detto **modello digitale dell'elevazione (DEM: digital elevation model)**. Un altro formato raster è detto **rete irregolare triangolare (TIN: triangular irregular network)** e si basa su un approccio topologico che utilizza vettori per modellare le superfici; i vettori vengono creati collegando punti campionati per formare triangoli e si ottiene una densità di punti che può variare con l'irregolarità del terreno. Le griglie rettangolari o matrici di elevazione sono strutture a matrici bidimensionali. Anche nella **modellazione del terreno (DTM: digital terrain modeling)** può essere usato questo modello, sostituendo l'elevazione con un altro attributo d'interesse come la densità della popolazione o la temperatura dell'aria. I dati GIS spesso includono una struttura temporale oltre che a una spaziale. Ad esempio la densità del traffico può essere misurata ogni 60 secondi in una serie di stazioni di campionamento sparse sul territorio.

Analisi dei dati. I dati GIS sono sottoposti a vari tipi di analisi. Ad esempio nelle applicazioni come gli studi dell'erosione del suolo, gli studi sull'impatto ambientale o le simulazioni di frane idrologiche, i dati DTM possono essere sottoposti a vari tipi di **analisi geomorfometrica**, misurando i valori di pendenza, i **gradienti** (il tasso di cambiamento in altitudine), l'**aspetto** (la direzione della bussola del gradiente), la **convessità del profilo** (il tasso di cambiamento del gradiente), la **convessità della pianta** (la convessità dei contorni delle regioni e altri parametri). Quando i dati GIS vengono usati per applicazioni di supporto decisionale possono essere sottoposti a operazioni di espansione e di aggregazione usando il data warehousing, come si è visto nel Sottoparagrafo 14.1.5. Possono essere eseguiti, inoltre, vari tipi di operazioni geometriche (per calcolare distanze, aree e volumi), topologiche (per calcolare sovrapposizioni, intersezioni e percorsi più brevi) e, infine, temporali (per calcolare interrogazioni basate su eventi o interne). Un'analisi approfondita in genere richiede molte operazioni spaziali e temporali.

Integrazione dei dati. I GIS devono integrare dati vettoriali e raster provenienti da diverse fonti. Talvolta si calcolano i profili dei bordi e le regioni di un'immagine raster per creare un modello vettoriale, in altri casi, vengono usate immagini raster come le fotografie aeree per aggiornare i modelli vettoriali. Diversi sistemi di coordinate come mercatore trasversale universale (UTM: universal transverse mercator), il sistema latitudine/longitudine e i sistemi catastali locali sono usati per localizzare località. I dati che vengono originati con sistemi di coordinate diversi richiedono appropriate trasformazioni. Le principali fonti pubbliche di dati geografici, tra cui i file TIGER gestiti dal Dipartimento del Commercio degli Stati Uniti d'America, vengono usati per tracciare carte stradali da molti strumenti di disegno di mappe presenti sul Web (ad esempio <http://maps.yahoo.com>). In queste applicazioni capita spesso che mappe con pochi attributi e alta precisione debbano essere integrate con altre che comprendono tanti attributi, ciascuno rilevato con poca precisione. L'integrazione viene allora eseguita con un processo detto "a banda elastica" in cui l'utente definisce una serie di punti di controllo in entrambe le mappe e la trasformazione della mappa di bassa precisione viene eseguita tenendo allineati i punti di controllo. Una questione fondamentale relativa all'integrazione è creare e mantenere i dati relativi agli attributi (come la qualità dell'aria o la densità del traffico) che possono essere messi in relazione e integrati con informazioni geografiche appropriate nel tempo, man mano che si modificano.

Rilevazione dei dati. La prima fase nello sviluppo di una base di dati spaziale per la modellazione cartografica consiste nel registrare le informazioni geografiche bidimensionali o tridimensionali nella forma digitale. Tale procedura talvolta è resa difficile dalle caratteristiche della mappa da digitalizzare, per la risoluzione, il tipo di proiezione, la scala utilizzata, la diversità delle tecniche di misurazione e dei sistemi di coordinate. I dati spaziali possono essere rilevati anche da sensori remoti nei satelliti come Landsat, NORA e il radiometro avanzato a risoluzione molto alta (AVHRR: Advanced Very High Resolution Radiometer) così come SPOT HRV (High Resolution Visible Range Instrument), uno strumento di facile interpretazione e molto preciso. Per la modellazione digitale del territorio i metodi di rilevazione dei dati vanno dai tradizionali sistemi manuali a quelli totalmente automatizzati. I rilevamenti topografici sono l'approccio tradizionale e il più preciso, ma richiedono molto tempo. Altre tecniche includono il campionamento con fotogrammi e i documenti cartografici digitalizzati.

14.4.3 Operazioni specifiche dei dati GIS

Le applicazioni GIS richiedono la definizione di operatori speciali come quelli di seguito riportati.

- **Interpolazione.** Questo processo permette di ottenere dati di elevazione anche per i punti in cui non è stato eseguito alcun campionamento. L'interpolazione può avvenire in punti singoli, su una griglia rettangolare o lungo un contorno e così via. I principali metodi di interpolazione si basano sulla triangolazione che usa il metodo TIN per interpolare le elevazioni all'interno di un triangolo sulla base di quelle campionate ai suoi vertici.
- **Interpretazione.** La modellazione digitale del suolo comporta l'interpretazione di operazioni sui dati del territorio come la modifica, lo smussamento, la riduzione dei dettagli e l'evidenziazione dei contorni. Altre operazioni sono relative al far coincidere e fondere i bordi di triangoli (in dati TIN), il che implica riconciliare i modelli e risolvere eventuali conflitti tra i dati degli attributi. Le conversioni tra i modelli di griglia, modelli di contorno e i dati TIN sono altre operazioni necessarie per l'interpretazione del territorio.
- **Analisi della prossimità.** Diverse tipologie di analisi di prossimità permettono il calcolo di "zone d'interesse" attorno agli oggetti, ad esempio la determinazione della distanza di sicurezza attorno a una macchina in viaggio su un'autostrada. Gli algoritmi per il calcolo del percorso più breve tra due punti che usano informazioni 2D o 3D sono una classe importante di strumenti allo scopo.
- **Elaborazione di immagini raster.** Si tratta di un insieme di tecniche che può essere diviso in due categorie: l'algebra delle mappe usata per integrare le caratteristiche geografiche di mappe sovrapponibili al fine di creare algebricamente nuove mappe; l'analisi delle immagini digitali per funzionalità quali il riconoscimento di confini e il rilevamento di oggetti all'interno delle immagini. L'individuazione delle strade in un'immagine satellitare di una città è un esempio di individuazione di oggetti.
- **Analisi di reti.** Le reti in GIS sono presenti in molti contesti e possono essere analizzate e soggette a segmentazioni, sovrapposizioni e così via. La sovrapposizione di reti si riferisce a un

sogna integrare più contesti possibili. I valori isolati non sono molto utili. Ad esempio, la sequenza di un segmento di DNA non è particolarmente utile senza informazioni aggiuntive che ne descrivano l'organizzazione, la funzione e così via. Un singolo nucleotide posto su un certo segmento di DNA, visto nel contesto di altri segmenti di DNA che non causano malattie, può essere individuato come causa pericolosa anemia delle cellule.

Caratteristica 8. *La definizione e la presentazione di interrogazioni complesse è estremamente importante per il biologo.* I sistemi biologici devono supportare interrogazioni complesse. Senza alcuna conoscenza della struttura dei dati (Caratteristica 6), gli utenti medi non possono costruire da soli un'interrogazione complessa sui dati. Quindi, per essere veramente utili, i sistemi devono fornire alcuni strumenti per creare tali interrogazioni. Come detto precedentemente, molti degli attuali sistemi offrono modelli di interrogazioni predefinite.

Caratteristica 9. *Gli utenti di informazioni biologiche spesso richiedono l'accesso a valori dei dati "vecchi", particolarmente quando cercano di verificare risultati ottenuti precedentemente.* È necessario predisporre un sistema di archivi storici che permetta l'accesso a diverse versioni dei valori. È necessario accedere ai dati più aggiornati ma anche alle versioni delle informazioni precedenti: i ricercatori vogliono poter interrogare i dati più aggiornati in modo consistente, ma devono al tempo stesso essere in grado di ricostruire il lavoro precedente e valutare nuovamente informazioni usate in precedenza insieme alle attuali. Di conseguenza, in una base di dati biologica i valori che stanno per essere aggiornati non possono essere semplicemente eliminati.

Tutte queste caratteristiche mostrano chiaramente che i DBMS odierni non soddisfano pienamente i requisiti dei complessi dati biologici. È necessario quindi sviluppare una nuova direzione di ricerca sui sistemi di gestione delle basi di dati⁵, specificamente orientata alla gestione dei dati biologici.

14.5.3 Il progetto del genoma umano e le basi di dati biologiche esistenti

Il termine *genoma* viene usato per indicare le informazioni genetiche totali che si possono ottenere su un'entità. Il *genoma umano*, ad esempio, in genere si riferisce alla serie completa di geni richiesti per creare un essere umano, stimati essere tra 100.000 e 300.000 e organizzati su 23 coppie di cromosomi che comprendono dai 3 ai 4 miliardi di nucleotidi. Il fine del progetto del genoma umano (HGP: Human Genome Project) era quello di ottenere la sequenza completa, ossia l'ordine delle basi, di quei nucleotidi: nel giugno 2000 è stata infatti an-

nunciata al mondo la mappatura del DNA, cioè il giusto ordine di tutte le "lettere" con cui è scritto il codice della vita d'un essere umano. Presa da sola, la sequenza umana del DNA non è particolarmente utile, ma se associata ad altri dati può essere usata come un utile strumento per affrontare svariati problemi di genetica, biochimica, medicina, antropologia e agricoltura. Nella costruzione delle basi di dati esistenti del genoma il punto principale è stato raccogliere (con l'opportuno controllo di qualità) e classificare le informazioni sulla sua sequenza. Oltre al genoma umano, sono stati studiati quelli di numerosi organismi come *E.coli*, *Drosophila* e *C.elegans*. Verranno qui brevemente presentati alcuni dei sistemi di basi di dati esistenti usati per il Progetto del Genoma Umano, o nati da esso.

Genbank. Oggi la base di dati di sequenze DNA più importante nel mondo è Genbank, sviluppato dal National Center for Biotechnology Information (NCBI) della National Library of Medicine (NLM). Istituita nel 1978 come un deposito centrale di dati relativi alle sequenze del DNA, da allora i suoi obiettivi sono stati ampliati per includere dati sui marcatori di sequenze, sulla sequenza e sulla struttura tridimensionale delle proteine, sulla classificazione e sui collegamenti alla letteratura biomedica (MEDLINE). La sua ultima versione contiene più di 602.000.000 nucleotidi di più di 920.000 sequenze da più di 16.000 specie, e approssimativamente dieci nuovi organismi vengono aggiunti ogni giorno. La dimensione della base di dati si è approssimativamente triplicata ogni diciotto mesi per cinque anni.

Pur essendo una base di dati complessa ed estesa, l'ambito del suo interesse si focalizza sulle sequenze umane e sui collegamenti alla letteratura. Altre fonti di dati (ad esempio sulla struttura tridimensionale e il sistema OMIM presentato nel seguito) sono state aggiunte recentemente ristrutturando le basi di dati esistenti dei sistemi OMIM e PDB e riprogettando la struttura del sistema Genbank per includere queste nuove serie di dati.

Il sistema è costituito da una combinazione di file piatti, basi di dati relazionali e file che contengono la **Notazione Sintattica Astratta 1** (ASN.1: Abstract Syntax Notation One), una sintassi per definire le strutture di dati sviluppate per l'industria delle telecomunicazioni. A ogni voce di Genbank è assegnato un identificatore univoco dal NCBI. Agli aggiornamenti viene assegnato un nuovo identificatore, mentre l'identificatore dell'entità originale rimane invariato per motivi di archivio. I riferimenti più vecchi a un'entità, quindi, non devono indicare per errore un nuovo e forse inappropriato valore. Per questo motivo i dati più recenti ricevono anche una seconda serie di identificatori univoci (UID: Unique IDentifiers) che mantengono la forma più aggiornata di un concetto, permettendo però ancora l'accesso alle versioni precedenti attraverso i loro identificatori originali.

L'utente medio della base di dati non è in grado di accedere direttamente alla struttura dei dati per eseguire interrogazioni o altre funzioni, sebbene istanze complete della base di dati possano essere esportate in molti formati, tra cui ASN.1. Il meccanismo di interrogazione fornito avviene attraverso l'applicazione Entrez (o la sua versione World Wide Web) che consente la ricerca tramite parole chiave, sequenze e l'UID di Genbank attraverso un'interfaccia statica.

La base di dati del genoma (GDB: Genome Database). Creato nel 1989, GDB è un catalogo dei dati sul gene umano che associa una specifica informazione a una specifica posizione nel genoma umano. Il grado di precisione di questa posizione sulla mappa dipende dalla fonte dei dati, che di solito non è al livello dei singoli nucleotidi. Tra i dati del GDB vi sono

⁵ Kogelnik (1998) descrive il prototipo dell'ambiente di basi di dati, chiamato GENOME, di gestione reticolare degli oggetti, dell'Istituto Tecnologico della Georgia a Emory (GENOME: Georgia Tech Emory Network Object Management Environment) per affrontare le questioni indicate precedentemente; vedere anche Kogelnik e altri (1997, 1998).

quelli che descrivono le informazioni principali della mappa genomica (compresi i limiti di confidenza e distanza) e i dati sperimentali sulla reazione a catena di polimerizzazione (PCR: *polymerase chain reaction*) (condizioni sperimentali, inneschi del PCR e reagenti usati). Più recentemente sono stati fatti degli sforzi per aggiungere dei dati sulle mutazioni collegandoli ai loci genetici, alle cellule usate negli esperimenti, alle biblioteche delle indagini sul DNA e ad alcuni dati relativi alle popolazioni e al polimorfismo fenotipico.

Il sistema GDB è costruito attorno a SYBASE, un DBMS relazionale commerciale i cui dati sono modellati usando le tecniche standard Entità-Relazione (si vedano i Capitoli 3 e 4). Gli implementatori di GDB hanno rilevato molte difficoltà nell'usare tale modello per rappresentare qualcosa di più di una semplice mappa genomica e dati d'indagine. Al fine di migliorare l'integrità dei dati e semplificare la programmazione per chi deve scrivere applicazioni, GDB distribuisce una serie di strumenti di accesso alle basi di dati (Database Access Toolkit). La maggior parte degli utenti, tuttavia, utilizza un'interfaccia Web per eseguire le interrogazioni collegandosi ai vari gestori del servizio. Ogni gestore tiene traccia dei collegamenti per una delle dieci tabelle all'interno del sistema GDB. Come con Genbank, agli utenti viene fornita solo una vista di livello molto alto dei dati al momento della ricerca, rendendo difficile l'uso della conoscenza estratta dalla struttura delle tabelle GDB. I metodi di ricerca sono più utili quando gli utenti stanno semplicemente cercando un indice nella mappa o i dati d'indagine. L'esplorazione diretta della base di dati non è incoraggiata dalle interfacce presenti. L'integrazione tra le strutture della base di dati di GDB e quelle di OMIM (si veda oltre) non è stata pienamente stabilita.

Eredità mendeliana in linea nell'uomo. L'eredità mendeliana in linea nell'uomo (OMIM: *on-line mendelian inheritance in man*) è un riepilogo elettronico delle informazioni sulle malattie genetiche umane. Iniziato su carta da Victor McCusick nel 1966 con 1500 voci, fu convertito in forma elettronica a pieno testo tra il 1987 e 1989 dal GDB. Nel 1991 la sua gestione fu trasferita dall'Università Johns Hopkins all'NCBI e l'intera base di dati fu convertita nel formato Genbank di NCBI. Oggi contiene più di 7000 voci.

OMIM contiene materiale su cinque aree patologiche suddivise per organi e sistemi. Qualsiasi proprietà morfologica, biochimica, comportamentale o di altro tipo sotto studio, viene definita **fenotipo** di un individuo (o di una cellula). Mendel si rese conto che i geni possono esistere in molte forme diverse note come **alleli**. Un **genotipo** si riferisce all'effettiva composizione allelica di un individuo.

Le voci di struttura del genotipo e del fenotipo contengono dati testuali non fortemente strutturati come descrizioni generali, nomenclatura, modi di ereditarietà, variazioni, struttura dei geni, mappaggio genomico e altre categorie. Le voci a pieno testo furono convertite nel formato strutturato ASN.1 quando OMIM fu trasferito al NCBI. Questa conversione ha migliorato molto la capacità di collegare i dati di OMIM con altre basi di dati e ha definito anche una struttura rigorosa per i dati. La forma base della base di dati, tuttavia, rimane difficile da modificare.

EcoCyc. L'Encyclopedia del Metabolismo e dei Geni di *Escherichia coli* è un esperimento recente che si propone di associare le informazioni sul genoma a quelle sul metabolismo di *E.coli* K-12. La base di dati fu creata nel 1996 grazie a una collaborazione tra l'Istituto di Ricerci Stanford, la Mayo Clinic, i laboratori della California e dei gerontologi di

E.coli, gli enzimi codificati da questi geni e le reazioni biochimiche catalizzate da ciascun enzima, nonché la loro organizzazione nei percorsi metabolici. Così facendo EcoCyc si occupa sia della sequenza sia dei domini delle funzioni delle informazioni dei genomi. Contiene 1283 composti con 965 strutture ed elenchi di legami e atomi, pesi molecolari e formule empiriche. Contiene altresì 3038 reazioni biochimiche descritte usando 269 classi di dati.

Il modello di dati orientato agli oggetti fu usato per la prima volta per implementare il sistema, con i dati memorizzati in Ocelot, un sistema di rappresentazione della conoscenza. I dati di EcoCyc sono stati ordinati in una gerarchia di classi di oggetti basandosi sull'osservazione che le proprietà di una reazione sono indipendenti dall'enzima che le catalizza e che un enzima ha molte proprietà che sono "logicamente distinte" dalle sue reazioni.

EcoCyc fornisce due metodi di interrogazione: uno diretto (attraverso interrogazioni predefinite) e uno indiretto (attraverso la navigazione ipertestuale). Le interrogazioni dirette vengono eseguite usando dei menu e delle finestre di dialogo che corrispondono a un repertorio esteso, ma finito di interrogazioni. Non è supportata la navigazione delle strutture di dati effettive. Non è documentato, inoltre, alcun meccanismo che permetta l'evoluzione dello schema.

In Tabella 14.1 sono riassunte le caratteristiche delle principali basi di dati del genoma e delle basi di dati HGMDB e ACEDB. Esistono anche altre basi di dati delle proteine che contengono informazioni sulle strutture proteiche, di cui le principali sono SWISS-PROT all'Università di Ginevra, la Banca di Dati delle Proteine (PDB: Protein Data Bank) al Brookhaven National Laboratory e la Protein Information Resource (PIR) alla University of Michigan.

Tabella 14.1 Riepilogo delle principali basi di dati relative al genoma.

Nome della base di dati	Contenuto	Tecnologia iniziale	Tecnologia attuale	Arearie di problemi di basi di dati	Tipi di dati principali
Genbank	Sequenza DNA/RNA, proteina	File testuali	File piatto/ASN.1	Consultazione degli schemi, evoluzione degli schemi, collegamento ad altri dbs	Testo, dati numerici, alcuni tipi complessi
OMIM	Genotipi e fenotipi di malattie e così via	Schede di indici/file testuali	File piatto/ASN.1	Voci in testo libero, non strutturate, che si collegano ad altri dbs	Testo
GDB	Dati di collegamento della mappa genetica	File piatto	Relazionale	Evoluzione/espansione degli schemi, oggetti complessi, collegamenti ad altri dbs	Testo, dati numerici
ACEDB	Dati di collegamento della mappa genetica, dati di sequenza (non umani)	OO	OO	Evoluzione/espansione degli schemi, collegamenti ad altri dbs	Testo, dati numerici
HGMDB	Sequenza e varianti delle sequenze	File piatto, applicazione specifica	File piatto, applicazione specifica	Evoluzione/espansione degli schemi, collegamenti ad altri dbs	Testo, dati numerici
EcoCyc	Percorsi e reazioni biochimiche	OO	OO	Gerarchia delle classi, evoluzione degli schemi	Tipi complessi, testo, dati numerici

ven National Laboratory e la Risorsa di Identificazione delle Proteine (PIR: Protein Identification Resource) alla National Biomedical Research Foundation.

Negli ultimi dieci anni vi è stato un aumento di interesse per le applicazioni delle basi di dati in biologia e medicina. Genbank, GDB e OMIM sono stati creati come archivi centrali di alcuni tipi di dati biologici, ma pur essendo estremamente utili non coprono ancora l'intero spettro dei dati del progetto del genoma umano. In tutto il mondo, però, si stanno facendo degli sforzi per progettare nuovi strumenti e tecniche che riducano i problemi di gestione dei dati per biologi e ricercatori medici.

14.5.4 Bibliografia selezionata per le basi di dati del genoma

La bioinformatica è diventata un'area di ricerca importante negli ultimi anni. Molti convegni e congressi sono stati organizzati su questo argomento. Robbins (1993) fornisce una buona panoramica, mentre Frenkel (1991) tratta in modo diffuso il progetto del genoma umano, evidenziandone il ruolo fondamentale nella bioinformatica. Cuticchia e altri (1993), Benson e altri (1996) e Pearson e altri (1994) forniscono informazioni utili su GDB, Genbank e OMIM. Wallace (1995) è stato un pioniere nella ricerca sul genoma mitocondriale che tratta di una parte specifica del genoma umano; la sequenza e i dettagli organizzativi di questa area appaiono in Anderson e altri (1981). Il lavoro recente documentato in Kogelnik e altri (1997, 1998) e in Kogelnik (1998) affronta lo sviluppo di una soluzione generica al problema della gestione dei dati nelle scienze biologiche. La base di dati MITOMAP è descritta in Kogelnik (1998) e vi si può accedere all'indirizzo <http://www.gen.emory.edu/mitomap.html>. La più grande base di dati di proteine SWISS-PROT è accessibile all'indirizzo <http://expasy.hcuge.ch/sprot/>. Le informazioni sulla base di dati ACEDB sono disponibili all'indirizzo <http://probe.nalusda.gov:8080/acedocs/>.

14.6 Biblioteche digitali

Le biblioteche digitali sono un'area di ricerca attiva e importante. Concettualmente, una biblioteca digitale è analoga a una biblioteca tradizionale – una vasta raccolta di fonti informative su diversi supporti – con in più i vantaggi delle tecnologie digitali. In realtà, le biblioteche digitali si distinguono per diversi aspetti significativi: la memorizzazione è digitale, l'accesso remoto è veloce e facile e i documenti sono copiati da una versione originale. Tenere a disposizione altre copie, inoltre, è facile e meno dipendente dai limiti di memoria e di bilancio, i principali problemi delle biblioteche tradizionali. Le tecnologie digitali, quindi, superano molte delle limitazioni economiche e fisiche delle biblioteche tradizionali.

L'introduzione al numero speciale sulle biblioteche digitali della rivista *Communications of the ACM* dell'aprile 1995 le descrive come "l'opportunità per realizzare il vecchio sogno di cogni essere umano: avere l'accesso all'intero archivio di informazioni dell'umanità". Nel Capitolo 1 una base di dati è stata definita come "una raccolta di dati correlati": a differenza dei dati correlati raccolti in una base di dati, una biblioteca digitale comprende varie fonti, molte

Tabella 14.2 Basi di dati e biblioteche digitali: affinità e differenze.

Affinità

- Entrambe riuniscono, organizzano, memorizzano, cercano, recuperano, elaborano e forniscono dati.
- Entrambe contengono e gestiscono più media.
- Entrambe contengono e gestiscono tipi di dati eterogenei e oggetti complessi.

Differenze

Biblioteche digitali	Base di dati
Nessun gestore centralizzato (la gestione è attuata attraverso interfacce concordate)	Gestore centralizzato (DBA)
Qualità dei dati meno controllata/non controllata	Qualità dei dati controllabile

delle quali non collegate tra loro. Logicamente le basi di dati possono essere viste come componenti delle biblioteche digitali (Tabella 14.2).

L'Iniziativa della Biblioteca Digitale (DLI: Digital Library Initiative), fondata congiuntamente da NSF, DARPA e NASA, è stata il principale acceleratore allo sviluppo delle biblioteche digitali. Nella sua prima fase, essa ha fornito un finanziamento significativo a sei progetti principali di sei università, che coprivano un vasto spettro di tecnologie.

Le pagine Web di questa iniziativa (si consulti dli.grainger.uiuc.edu/national.htm) definiscono lo scopo della ricerca come segue: "Migliorare in modo sostanziale i metodi per raccogliere, memorizzare e organizzare le informazioni in forma digitale e renderle disponibile per la ricerca, il recupero e l'elaborazione attraverso reti di comunicazione – e tutto in modo amichevole per l'utente finale".

Il grande volume di queste raccolte di dati così come la diversità e la molteplicità dei formati costituiscono sfide di un diverso livello. La futura progressione nello sviluppo delle biblioteche digitali probabilmente passerà dalla tecnologia attuale di ricerca tramite Internet, attraverso ricerche in rete di informazioni indicizzate raccolte in archivi comuni, a nuovi sistemi di correlazione temporale delle informazioni e analisi tramite reti intelligenti. Le tecniche per raccogliere le informazioni, memorizzarle e organizzarle per supportare requisiti informativi appresi nei decenni di progettazione e implementazione di basi di dati, costituiranno il riferimento per lo sviluppo di approcci appropriati per le biblioteche digitali. La ricerca, il recupero e l'elaborazione di diverse tipologie di dati digitali utilizzeranno le lezioni apprese dalle attività svolte nelle basi di dati su quelle tipologie di dati digitali.

14.6.1 L'Iniziativa delle Biblioteche Digitali

L'Università dell'Illinois a Urbana Champaign ha coordinato la sincronizzazione a livello nazionale di sei progetti della DLI in corso negli Stati Uniti presso le sei università che partecipano all'iniziativa. I progetti si propongono di individuare i requisiti fondamentali di sviluppo delle biblioteche digitali. I partecipanti e i temi principali sono i seguenti.

- Università della California a Berkeley: Sistemi di Informazioni Geografiche e Pianificazione Ambientale (si veda il Paragrafo 14.4 per dettagli sui sistemi GIS). Questo progetto

- implementerà una biblioteca digitale vista come una raccolta di servizi informativi distribuiti, mettendo l'accento sull'indicizzazione automatica, la ricerca intelligente, il supporto di basi di dati distribuite, un protocollo client/server migliorato per il recupero delle informazioni, una miglior acquisizione dei dati e un nuovo paradigma di interazione.
- *Università della California a Santa Barbara: Progetto Alexandria.* Informazioni sulle mappe ottenute tramite referenziazione spaziale. Questo progetto svilupperà servizi di biblioteca distribuiti per raccolte di informazioni geografiche indicizzate spazialmente (si veda il Paragrafo 14.4).
 - *Università di Carnegie Mellon: Biblioteca Digitale Video "Informedia".* Usando biblioteche di video digitali, il progetto è focalizzato sulla ricerca e l'interrogazione basata sul contenuto.
 - *Università dell'Illinois a Urbana Champaign: Archivi Federati di Letteratura Scientifica.* Questo progetto si propone di sviluppare con legge di scala una biblioteca digitale affinché possa essere utilizzata da migliaia di utenti e documenti.
 - *Università del Michigan: Agenti Intelligenti per la Localizzazione delle Informazioni.* Questo progetto verterà sulla tecnologia degli agenti, mettendo l'accento sull'utilizzo di agenti d'interfaccia utente, agenti di mediazione per coordinare le ricerche e agenti per la costituzione delle raccolte.
 - *Università di Stanford: Meccanismi di Interoperabilità tra Servizi Eterogenei.* Questo progetto svilupperà le tecnologie necessarie per una biblioteca digitale integrata, che fornisca un accesso uniforme a nuovi servizi usando raccolte di informazioni disponibili in rete.

14.6.2 Bibliografia selezionata per le biblioteche digitali

Il numero speciale della rivista *Communications of the ACM* dell'aprile 1995 è interamente dedicato alle biblioteche digitali. In quella sede Wiederhold (1995) discute il ruolo delle biblioteche digitali nell'aumento della produttività umana nel trovare nuova conoscenza. Il numero dell'aprile 1998 di *Communications of the ACM* è dedicato ad ampliare lo scopo e a fornire un accesso illimitato alle biblioteche digitali. Schatz (1995, 1997) offre un'eccellente trattazione sul recupero delle informazioni, sulla ricerca e sull'analisi e di come esse sono effettuate in Internet. Le università che partecipano all'Iniziativa delle Biblioteche Digitali possono essere raggiunte ai seguenti URL:

- Iniziativa delle Biblioteche Digitali: dli.grainger.uiuc.edu/national.htm
- Università di Berkeley: elib.cs.berkeley.edu
- Università di Santa Barbara: alexandria.sdc.ucsb.edu
- Università Carnegie-Mellon: www.informedia.cs.cmu.edu
- Università dell'Illinois a Urbana Champaign: dli.grainger.uiuc.edu/default.htm
- Università del Michigan: www.si.umich.edu/UMDL
- Università di Stanford: walrus.stanford.edu/digilib

Appendice A

Notazioni diagrammatiche alternative

In Figura A.1 sono illustrate alcune notazioni diagrammatiche usate per rappresentare i concetti del modello ER e EER. Purtroppo non esiste una notazione standard: i progettisti di basi di dati preferiscono notazioni diverse. Analogamente, vari strumenti CASE (computer-aided software engineering) e metodologie OOA (object-oriented analysis: analisi orientata a oggetti) usano notazioni differenti. Alcune sono associate a modelli che presentano concetti e vincoli aggiuntivi oltre a quelli dei modelli ER e EER descritti nei Capitoli 3 e 4, mentre altri modelli hanno meno concetti e vincoli. La notazione utilizzata nel Capitolo 3 è abbastanza vicina alla notazione originale dei diagrammi ER, che è ancora ampiamente usata. Qui discuteremo alcune notazioni alternative.

In Figura A.1(a) sono presentate diverse notazioni per la rappresentazione grafica di tipi di entità/classi, attributi e associazioni. Nei Capitoli 3 e 4 sono stati usati i simboli che in figura sono indicati con (i), ossia rettangoli, ovali e rombi. Si noti che il simbolo (ii) per i tipi di entità/classi, il simbolo (ii) per gli attributi e il simbolo (ii) per le associazioni sono tra loro simili, ma sono usati da metodologie diverse per rappresentare tre concetti diversi. Il simbolo di linea retta (iii) per rappresentare associazioni è usato da più strumenti e metodologie.

In Figura A.1(b) sono presentate alcune notazioni usate per collegare attributi a tipi di entità. Qui è stata usata la notazione (i). La notazione (ii) usa la terza notazione (iii) di Figura A.1(a) per gli attributi. Le ultime due notazioni di Figura A.1(b) – (iii) e (iv) – sono frequentemente usate nelle metodologie OOA e in qualche strumento CASE. In particolare, l'ultima notazione rappresenta sia gli attributi sia i metodi di una classe, separati da una linea orizzontale.

In Figura A.1(c) sono mostrate varie notazioni per rappresentare il rapporto di cardinalità di associazioni binarie. Nei Capitoli 3 e 4 è stata usata la notazione (i). La notazione (ii) – nota come notazione *a zampa di gallina* – è abbastanza popolare. La notazione (iv) usa la freccia come riferimento funzionale (dal lato N al lato 1) e assomiglia alla notazione qui usata per le chiavi esterne nel modello relazionale (Figura 7.7); la notazione (v) – impiegata nei dia-

Appendice B

Parametri dei dischi

Il più importante parametro di disco è il tempo richiesto per localizzare un arbitrario blocco di disco, dato il suo indirizzo, e quindi per trasferire il blocco tra il disco e un buffer in memoria centrale. Questo è il **tempo di accesso casuale (random access time)** per accedere a un blocco di disco. Le componenti temporali da considerare sono tre.

1. **Tempo di posizionamento (seek time: s).** Per dischi a testina mobile è il tempo necessario per posizionare meccanicamente la testina di lettura/scrittura sulla traccia corretta (per dischi a testina fissa, è il tempo necessario per commutare elettronicamente all'appropriata testina di lettura/scrittura). Per dischi a testina mobile questo tempo varia, dipendendo dalla distanza esistente fra la traccia al momento sottostante la testina di lettura/scrittura e la traccia specificata nell'indirizzo di blocco. Di solito il costruttore di dischi fornisce un tempo di posizionamento medio in millisecondi. La tipica gamma di valori per il tempo di posizionamento medio va da 10 a 60 msec. Il tempo di posizionamento è il principale "colpevole" del ritardo che si presenta nel trasferimento di blocchi tra disco e memoria.

2. **Ritardo di rotazione (rotational delay: rd).** Una volta che la testina di lettura/scrittura si trova sulla traccia corretta, l'utente deve attendere che l'inizio del blocco richiesto ruoti fino alla posizione che sta sotto la testina di lettura/scrittura. Ciò richiede in media il tempo necessario a effettuare un mezzo giro di disco, ma in realtà va dall'accesso immediato (se subito dopo il posizionamento l'inizio del blocco richiesto è nella posizione sottostante la testina di lettura/scrittura) fino a un giro completo di disco (se dopo il posizionamento l'inizio del blocco richiesto ha appena superato la testina di lettura/scrittura). Se la velocità di rotazione del disco è di p giri al minuto (rpm: revolutions per minute), allora il ritardo di rotazione medio rd è dato da

$$rd = (1/2)*(1/p) \text{ min} = (60*1000)/(2*p) \text{ msec}$$

Un valore tipico per p è di 10000 rpm, che dà un ritardo di rotazione di $rd = 3\text{msec}$. Per dischi a testina fissa, nei quali il tempo di posizionamento è trascurabile, questa componente causa il maggior ritardo nel trasferimento di un blocco di disco.

3. Tempo di trasferimento di blocco (block transfer time: btt). Una volta che la testina di lettura/scrittura si è portata all'inizio del blocco desiderato, è richiesto un certo tempo per trasferire i dati presenti nel blocco. Questo tempo di trasferimento di blocco dipende dalla dimensione del blocco, da quella della traccia e dalla velocità di rotazione. Se il tasso di trasferimento (transfer rate) del disco è pari a $tr \text{ byte/msec}$, e la dimensione del blocco è di $B \text{ byte}$, allora

$$btt = B/tr \text{ msec}$$

Se la dimensione della traccia è pari a 50 Kbyte e p è uguale a 3600 rpm, il tasso di trasferimento in byte/msec è

$$tr = (50*1000)/(60*1000/3600) = 3000 \text{ byte/msec}^1$$

In questo caso, $btt = B/3000 \text{ msec}$, dove B è la dimensione del blocco in byte.

Il tempo medio necessario per trovare e trasferire un blocco, dato il suo indirizzo, è stimato in

$$(s + rd + btt) \text{ msec}$$

Ciò vale sia per la lettura che per la scrittura di un blocco. Il metodo principale per ridurre questo tempo consiste nel trasferire più blocchi memorizzati su una o più tracce dello stesso cilindro: in questo modo il tempo di posizionamento è richiesto solo per il primo blocco. Per trasferire consecutivamente k blocchi *non contigui*, presenti nello *stesso cilindro*, si ha approssimativamente bisogno di

$$s + (k * (rd + btt)) \text{ msec}$$

In questo caso sono necessari due o più buffer in memoria centrale, dato che si stanno continuamente leggendo o scrivendo i k blocchi, come visto nel Paragrafo 4.3. Il tempo di trasferimento di blocco si riduce anche di più quando vengono trasferiti *blocchi consecutivi* della stessa traccia o cilindro. Ciò elimina il ritardo di rotazione per tutti i blocchi tranne il primo, e così il tempo stimato per trasferire k blocchi consecutivi diventa

$$s + rd + (k * btt) \text{ msec}$$

Una stima più accurata del trasferimento di blocchi consecutivi tiene conto dello spazio tra blocchi (si veda il Sottoparagrafo 5.2.1), che contiene le informazioni che consentono alla te-

stina di lettura/scrittura di determinare quale blocco sta per leggere. Di solito il costruttore di dischi fornisce un **tasso di trasferimento di una mole di dati (bulk transfer rate: btr)** che tiene conto della dimensione dello spazio tra blocchi quando si leggono blocchi memorizzati consecutivamente. Se la dimensione dello spazio tra blocchi è di $G \text{ byte}$, allora

$$btr = (B/(B + G)) * tr \text{ byte/msec}$$

Il tasso di trasferimento di una mole di dati è il tasso di trasferimento di *byte utili* presenti nei blocchi di dati. Quando il disco ruota, la testina di lettura/scrittura deve passare sopra tutti i byte presenti in una traccia, compresi i byte dello spazio tra blocchi, che contengono informazioni di controllo ma non dati effettivi. Quando si usa il tasso di trasferimento di una mole di dati, il tempo necessario per trasferire i dati utili di un blocco posto fra molti blocchi consecutivi è B/btr . Perciò il tempo stimato per leggere k blocchi memorizzati consecutivamente sullo stesso cilindro diventa

$$s + rd + (k * (B/btr)) \text{ msec}$$

Un altro parametro di disco è il **tempo di riscrittura (rewrite time)**. Esso si rivela utile quando si legge un blocco da disco, lo si trasferisce in un buffer in memoria centrale, si modifica il contenuto del buffer, e quindi si riscrive il contenuto del buffer nello stesso blocco di disco in cui era inizialmente memorizzato. In molti casi il tempo richiesto per modificare il buffer in memoria centrale è inferiore al tempo richiesto per una rotazione completa del disco. Se sappiamo che il buffer è pronto per la riscrittura, il sistema può tenere le testine di disco sulla stessa traccia, e durante la successiva rotazione del disco il contenuto modificato del buffer viene riscritto nel blocco di disco. Perciò si stima che il tempo di riscrittura T_{rw} sia il tempo necessario per una rotazione completa del disco:

$$T_{rw} = 2 * rd \text{ msec}$$

In sintesi, si fornisce qui di seguito un elenco dei parametri esaminati e dei relativi simboli usati:

tempo di posizionamento: $s \text{ msec}$

ritardo di rotazione: $rd \text{ msec}$

tempo di trasferimento di blocco: $btt \text{ msec}$

tempo di riscrittura: $T_{rw} \text{ msec}$

tasso di trasferimento: $tr \text{ byte/msec}$

tasso di trasferimento di una mole di dati: $btr \text{ byte/msec}$

dimensione di blocco: $B \text{ byte}$

dimensione dello spazio tra blocchi: $G \text{ byte}$

¹ Nei calcoli si è supposto, per semplicità, 1 Kbyte = 1000 byte. (N.d.T.)

[Agrawal e altri 1993a] R. Agrawal, T. Imielinski e A. Swami, "Mining Association Rules Between Sets of Items in Databases", in [SIGMOD 1993].

[Agrawal e altri 1993b] R. Agrawal, T. Imielinski e A. Swami, "Database Mining: A Performance Perspective", *IEEE TKDE*, Volume 5, Number 6 (Agosto 1993).

[Agrawal e Srikant 1994] R. Agrawal e R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", in [VLDB 1994].

[Agrawal e altri 1996] R. Agrawal, M. Mehta, J. Shafer e R. Srikant, "The Quest Data Mining System", in [KDD 1996].

[Aho e altri 1979] A. Aho, C. Beeri e J. Ullman, "The Theory of Joins in Relational Databases", *TODS*, Volume 4, Number 3 (Settembre 1979).

[Amir e altri 1997] A. Amir, R. Feldman e R. Kashi, "A New and Versatile Method for Association Generation", *Information Systems*, Volume 22, Number 6 (Settembre 1997).

[Anderson e altri 1981] S. Anderson, A. Bankier, B. Barrell, M. deBruijn, A. Coulson, J. Drouin, I. Eperon, D. Nierlich, B. Rose, F. Sanger, P. Schreier, A. Smith, R. Staden e I. Young, "Sequence and Organization of the Human Mitochondrial Genome", *Nature*, Number 290, pagg. 457-465 (1981).

[ANSI 1975] American National Standards Institute Study Group on Data Base Management Systems, Interim Report, FDT, Volume 7, Number 2, ACM (1975).

[ANSI 1986] American National Standards Institute, "The Database Language SQL", Document ANSI X3.135 (1986).

[Armstrong 1974] W. Armstrong, "Dependency Structures of Data Base Relationships", *Proceedings of the IFIP Congress* (1974).

[Atzeni e De Antonellis 1993] P. Atzeni e V. De Antonellis, *Relational Database Theory*, Benjamin/Cummings (1993).

[Atzeni e altri 1997] P. Atzeni, G. Mecca e P. Merialdo, "To Weave the Web", in [VLDB 1997].

[Baeza-Yates e Larson 1989] R. Baeza-Yates e P. A. Larson, "Performance of B1-trees with Partial Expansions", *TKDE*, Volume 1, Number 2 (Giugno 1989).

[Bayer e McCreight 1972] R. Bayer e E. McCreight, E., "Organization and Maintenance of Large Ordered Indexes", *Acta Informatica*, Volume 1, Number 3 (Febbraio 1972).

[Benson e altri 1996] D. Benson, M. Boguski, D. Lipman e J. Ostell, "GenBank", *Nucleic Acids Research*, Volume 24, Number 1 (1996).

[Berg e Roth 1989] B. Berg e J. Roth, *Software for Optical Disk*, Meckler (1989).

[Berners-Lee e altri 1992] T. Berners-Lee, R. Caillian, J. Groff e B. Pollermann, "World-Wi-

de Web: The Information Universe", *Electronic Networking: Research, Applications and Policy*, Volume 1, Number 2 (1992).

[Berners-Lee e altri 1994] T. Berners-Lee, R. Caillian, A. Lautonen, H. Nielsen e A. Secret, "The World Wide Web", *CACM*, Volume 13, Number 2 (Agosto 1994).

[Bernstein 1976] P. Bernstein, "Synthesizing Third Normal Form Relations from Functional Dependencies", *TODS*, Volume 1, Number 4 (Dicembre 1976).

[Bertino e altri 1998] F. Bertino, Rabbitti e S. Gibbs, "Query Processing in a Multimedia Environment", *TOIS*, Number 6 (1998).

[Bischoff e Alexander 1997] J. Bischoff e T. Alexander, a cura di, *Data Warehouse: Practical Advice from the Experts*, Prentice-Hall (1997).

[Biskup e altri 1979] J. Biskup, U. Dayal e P. Bernstein, "Synthesizing Independent Database Schemas", in [SIGMOD 1979].

[Blakeley e altri 1989] J. Blakeley, N. Coburn e P. Larson, "Updated Derived Relations: Detecting Irrelevant and Autonomously Computable Updates", *TODS*, Volume 14, Number 3 (Settembre 1989).

[Booch e altri 1999] G. Booch, J. Rumbaugh e I. Jacobson, *Unified Modeling Language User Guide*, Addison-Wesley (1999).

[Boyce e altri 1975] R. Boyce, D. Chamberlin, W. King e M. Hammer, "Specifying Queries as Relational Expressions", *CACM*, Volume 18, Number 11 (Novembre 1975).

[Bracchi e altri 1976] G. Bracchi, P. Paolini e G. Pelagatti, "Binary Logical Associations in Data Modelling", in [Nijssen 1976].

[Burkhard 1976] W. Burkhard, "Hashing and Trie Algorithms for Partial Match Retrieval", *TODS*, Volume 1, Number 2 (Giugno 1976), pagg. 175-187.

[Burkhard 1979] W. Burkhard, "Partial-match Hash Coding: Benefits of Redundancy", *TODS*, Volume 4, Number 2 (Giugno 1979), pagg. 228-239.

[Cammarata e altri 1989] S. Cammarata, P. Ramachandra e D. Shane, "Extending a Relational Database with Deferred Referential Integrity Checking and Intelligent Joins", in [SIGMOD 1989].

[Campbell e altri 1985] D. Campbell, D. Embley e B. Czejdo, "A Relationally Complete Query Language for the Entity-Relationship Model", in [ER Conference 1985].

[Carlis 1986] J. Carlis, "HAS, a Relational Algebra Operator or Divide Is Not Enough to Conquer", in [ICDE 1986].

[Cattell 1997] R. Cattell, a cura di, *The Object Database Standard: ODMG, Release 2.0*, Morgan Kaufmann (1997).

- [Cesarini e Soda 1991] F. Cesarini e G. Soda, "A Dynamic Hash Method with Signature", *TODS*, Volume 16, Number 2 (Giugno 1991).
- [Chamberlin e Boyce 1974] D. Chamberlin e R. Boyce, "SEQUEL: A Structured English Query Language", in [SIGMOD 1974].
- [Chamberlin e altri 1976] D. Chamberlin e altri, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control", *IBM Journal of Research and Development*, Volume 20, Number 6 (Novembre 1976).
- [Chan e altri 1992] C. Chan, B. Ooi e H. Lu, "Extensible Buffer Management of Indexes", in [VLDB 1992].
- [Chaudhuri e Dayal 1997] S. Chaudhuri e U. Dayal, "An Overview of Data Warehousing and OLAP Technology", *SIGMOD Record*, Volume 26, Number. 1 (Marzo 1997).
- [Chen 1976] P. Chen, "The Entity Relationship Mode-Toward a Unified View of Data", *TODS*, Volume 1, Number 1 (Marzo 1976).
- [Chen e Patterson 1990] P. Chen e D. Patterson, "Maximizing Performance in a Striped Disk Array", *Proceedings of Symposium on Computer Architecture*, IEEE, New York (1990).
- [Chen e altri 1994] P. Chen, E. Lee, G. Gibson, R. Katz e D. Patterson, "RAID High Performance, Reliable Secondary Storage", *ACM Computing Surveys*, Volume 26, Number 2 (1994).
- [Cheung e altri 1996] D. Cheung, J. Han, V. Ng, A. W. Fu e A. Y. Fu, "A Fast and Distributed Algorithm for Mining Association Rules", *Proceedings of International Conference on Parallel and Distributed Information Systems*, PDIS (1996).
- [Childs 1968] D. Childs, "Feasibility of a Set Theoretical Data Structure - A General Structure Based on a Reconstituted Definition of Relation", *Proceedings of the IFIP Congress* (1968).
- [Chintalapati e altri 1997] R. Chintalapati, V. Kumar e A. Datta, "An Adaptive Location Management Algorithm for Mobile Computing", *Proceedings of 22nd Annual Conference on Local Computer Networks (LCN '97)*, Minneapolis (1997).
- [Chlaybrook 1983] B. Claybrook, *File Management Techniques*, Wiley (1983).
- [Codd 1970] E. Codd, "A Relational Model for Large Shared Data Banks", *CACM*, Volume 13, Number 6 (Giugno 1970).
- [Codd 1971] E. Codd, "A Data Base Sublanguage Founded on the Relational Calculus", *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control* (Novembre 1971).
- [Codd 1972a] E. Codd, "Relational Completeness of Data Base Sublanguages", in [Rustin 1972].

- [Codd 1972b] E. Codd, "Further Normalization of the Data Base Relational Model", in [Rustin 1972].
- [Codd 1974] E. Codd, "Recent Investigations in Relational Database Systems", *Proceedings of the IFIP Congress* (1974).
- [Codd 1979] E. Codd, "Extending the Database Relational Model to Capture More Meaning", *TODS*, Volume 4, Number 4 (Dicembre 1979).
- [Codd 1985] E. Codd, "Is Your DBMS Really Relational?" e "Does Your DBMS Run By the Rules?", *COMPUTER WORLD*, (14 e 21 Ottobre 1985).
- [Codd 1990] E. Codd, *Relational Model for Data Management-Version 2*, Addison-Wesley (1990).
- [Codd e altri 1993] E. F. Codd, S. B. Codd, e C. T. Salley, "Providing OLAP (On-Line Analytical Processing) to User Analyst: An IT Mandate", <http://www.arborsoft.com/OLAP.html> (1993).
- [Comer 1979] D. Comer, "The Ubiquitous B-tree", *ACM Computing Surveys*, Volume 11, Number 2 (Giugno 1979).
- [Cuticchia e altri 1993] A. Cuticchia, K. Fasman, D. Kingsbury, R. Robbins e P. Pearson, "The GDB Human Genome Database Anno 1993", *Nucleic Acids Research*, Volume 21, Number 13 (1993).
- [Date 1983] C. Date, "The Outer Join", *Proceedings of the Second International Conference on Databases (ICOD-2)* (1983).
- [Date 1984] "A Critique of the SQL Database Language", *ACM SIGMOD Record*, Volume 14, Number 3 (Novembre 1984).
- [Date 1995] C. Date, *An Introduction Database Systems*, VI ed., Addison-Wesley (1995).
- [Date e Darwen 1993] C. J. Date, H. Darwen, *A Guide to the SQL Standard*, III ed., Addison-Wesley (1993).
- [Dayal e Bernstein 1978] U. Dayal e P. Bernstein, "On the Updatability of Relational Views", *VLDB* (1978).
- [DBGT 1971] DBGT, *Report of the codasyl Data Base Task Group*, ACM (Aprile 1971).
- [Dhawan 1997] C. Dhawan, *Mobile Computing*, McGraw-Hill (1997).
- [Dipert e Levy 1993] B. Dipert e M. Levy, *Designing with Flash Memory*, Annabooks (1993).
- [Dogac 1998] A. Dogac, "Special Section on Electronic Commerce", *ACM Sigmod Record*, Volume 27, Number 4 (Dicembre 1998).

- [Lanka e Mays 1991] S. Lanka e E. Mays, "Fully Persistent B1-Trees", in [SIGMOD 1991].
- [Larson 1978] P. Larson, "Dynamic Hashing", *BIT*, Number 18 (1978).
- [Larson 1981] P. Larson, "Analysis of Index-Sequential Files with Overflow Chaining", *TODS*, Volume 6, Number 4 (Dicembre 1981).
- [Laurini e Thompson 1992] R. Laurini e D. Thompson, *Fundamentals of Spatial Information Systems*, Academic Press (1992).
- [Lehman e Yao 1981] P. Lehman e S. Yao, "Efficient Locking for Concurrent Operations on B-Trees", *TODS*, Volume 6, Number 4 (Dicembre 1981).
- [Lenzerini e Santucci 1983] M. Lenzerini e C. Santucci, "Cardinality Constraints in the Entity Relationship Model", in [ER Conference 1983].
- [Li e altri 1998] W. Li, K. Seluk Candan, K. Hirata e Y. Hara, "Hierarchical Image Modeling for Object-based Media Retrieval", *DKE*, Volume 27, Number 2 (Settembre 1998), pagg. 139-176.
- [Lin e Dunham 1998] J. Lin e M. H. Dunham, "Mining Association Rules", in [ICDE 1998].
- [Lippman 1987] R. Lippman, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine* (Aprile 1987).
- [Litwin 1980] W. Litwin, "Linear Hashing: A New Tool for File and Table Addressing", in [VLDB 1980].
- [Livadas 1989] P. Livadas, *File Structures: Theory and Practice*, Prentice-Hall (1989).
- [Maguire e altri 1997] D. Maguire, M. Goodchild e D. Rhind, a cura di, *Geographical Information Systems: Principles and Applications*, voll. 1 e 2, Longman Scientific and Technical, New York (1997).
- [Mahajan e altri 1998] S. Mahajan, M. J. Donahoo, S. B. Navathe, M. Ammar e S. Malik, "Grouping Techniques for Update Propagation in Intermittently Connected Databases", *ICDE* (1998).
- [Maier 1983] D. Maier, *The Theory of Relational Databases*, Computer Science Press (1983).
- [Mannila e altri 1994] H. Mannila, H. Toivonen e A. Verkamo, "Efficient Algorithms for Discovering Association Rules", *KDD-94, AAAI Workshop on Knowledge Discovery in Databases*, Seattle (1994).
- [Manola 1998] F. Manola, "Towards a Richer Web Object Model", *SIGMOD Record*, Volume 27, Number 1 (Marzo 1998).
- [Markowitz e Raz 1983] V. Markowitz e Y. Raz, "ERROL: An Entity-Relationship, Role Oriented, Query Language", in [ER Conference 1983].

- [Mattison 1996] R. Mattison, *Data Warehousing: Strategies, Technologies, and Techniques*, McGraw-Hill (1996).
- [Melton e Simon 1993] J. Melton e A. R. Simon, *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann (1993).
- [Mendelzon e altri 1997] A. Mendelzon, G. Mihaila e T. Milo, "Querying the World Wide Web", *Journal of Digital Libraries*, Volume 1, Number 1 (Aprile 1997).
- [Miller 1987] N. Miller, *File Structures Using PASCAL*, Benjamin Cummings (1987).
- [Mohan e Narang 1992] C. Mohan e I. Narang, "Algorithms for Creating Indexes for Very Large Tables without Quiescing Updates", in [SIGMOD 1992].
- [Morris 1968] R. Morris, "Scatter Storage Techniques", *CACM*, Volume 11, Number 1 (Gennaio 1968).
- [Navathe e Schkolnick 1978] S. Navathe e M. Schkolnick, "View Representation in Logical Database Design", in [SIGMOD 1978].
- [Negri e altri 1991] M. Negri, S. Pelegatti e L. Sbatella, "Formal Semantics of SQL Queries", *TODS*, Volume 16, Number 3 (Settembre 1991).
- [Ng 1981] P. Ng, "Further Analysis of the Entity-Relationship Approach to Database Design", *TSE*, Volume 7, Number 1 (Gennaio 1981).
- [Nievergelt 1974] J. Nievergelt, "Binary Search Trees and File Organization", *ACM Computing Surveys*, Volume 6, Number 3 (Settembre 1974).
- [Nievergelt e altri 1984] J. Nievergelt, H. Hinterberger e K. Seveik, "The Grid File: An Adaptable Symmetric Multikey File Structure", *TODS*, Volume 9, Number 1 (Marzo 1984), pagg. 38-71.
- [Nijssen 1976] G. Nijssen, a cura di, *Modelling in Data Base Management Systems*, North-Holland (1976).
- [Nwosu e altri 1996] K. Nwosu, P. Berra e B. Thuraisingham, a cura di, *Design and Implementation of Multimedia Database Management Systems*, Kluwer Academic (1996).
- [Olle e altri 1982] T. Olle, H. Sol e A. Verrijn-Stuart, a cura di, *Information System Design Methodology*, North-Holland (1982).
- [Oracle 1997a] Oracle, *Oracle 8 Server Concepts*, voll. 1 e 2, Release 8-0, Oracle Corporation (1997).
- [Oracle 1997b] Oracle, *Oracle 8 Server Distributed Database Systems*, Release 8.0 (1997).
- [Oracle 1997c] Oracle, *PL/SQL User's Guide and Reference*, Release 8.0 (1997).

- [Oracle 1997d] Oracle, *Oracle 8 Server SQL Reference*, Release 8.0 (1997).
- [Oracle 1997e] Oracle, *Oracle 8 Parallel Server, Concepts and Administration*, Release 8.0 (1997).
- [Oracle 1997f] Oracle, *Oracle 8 Server Spatial Cartridge, User's Guide and Reference*, Release 8.0.3 (1997).
- [Osborn 1979] S. Osborn, "Towards a Universal Relation Interface", in [VLDB 1979].
- [Ozsoyoglu e altri 1985] G. Ozsoyoglu, Z. Ozsoyoglu e V. Matos, "Extending Relational Algebra and Relational Calculus with Set Valued Attributes and Aggregate Functions", *TODS*, Volume 12, Number 4 (Dicembre 1985).
- [Parent e Spaccapietra 1985] C. Parent e S. Spaccapietra, "An Algebra for a General Entity-Relationship Model", *TSE*, Volume 11, Number 7 (Luglio 1985).
- [Park e altri 1995] J. Park, M. Chen e P. Yu, "An Effective Hash Based Algorithm for Mining Association Rules", in [SIGMOD 1995].
- [Patterson e altri 1988] D. Patterson, G. Gibson e R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", in [SIGMOD 1988].
- [Pazandak e Srivastava 1995] P. Pazandak e J. Srivastava, "Evaluating Object DBMSs for Multimedia", *IEEE Multimedia*, Volume 4, Number 3 (1995), pagg. 34-49.
- [Pearson e altri 1994] P. Pearson, C. Francomano, P. Foster, C. Bocchini, P. Li e V. McKusick, "The Status of Online Mendelian inheritance in Man (OMIM) Medio 1994", *Nucleic Acids Research*, Volume 22, Number 17 (1994).
- [Piatetsky-Shapiro e Frauley 1991] G. Piatetsky-Shapiro e W. Frauley, a cura di, *Knowledge Discovery in Databases*, AAAI Press/MIT Press (1991).
- [Pitoura e Samaras 1998] E. Pitoura e G. Samaras, *Data Management for Mobile Computing*, Kluwer (1998).
- [Ramakrishnan 1997] R. Ramakrishnan, *Database Management Systems*, McGraw-Hill (1997).
- [Reisner 1977] P. Reisner, "Use of Psychological Experimentation as an Aid to Development of a Query Language", *TSE*, Volume 3, Number 3 (Maggio 1977).
- [Robbins 1993] R. Robbins, "Genome Informatics: Requirements and Challenges", *Proceedings of the Second International Conference on Bioinformatics, Supercomputing and Complex Genome Analysis*, World Scientific Publishing (1993).
- [Ruemmler e Wilkes 1994] C. Ruemmler e J. Wilkes, "An Introduction to Disk Drive Modeling", *IEEE Computer*, Volume 27, Number 3 (Marzo 1994), pagg. 17-27.

- [Rustin 1972] R. Rustin, a cura di, *Data Base Systems*, Prentice-Hall (1972).
- [Salzberg 1988] B. Salzberg, *File Structures: An Analytic Approach*, Prentice-Hall (1988).
- [Salzberg e altri 1990] B. Salzberg e altri, "FastSort: A Distributed Single-Input Single-Output External Sort", in [SIGMOD 1990].
- [Sarasua e O'Neill 1999] W. Sarasua e W. O'Neill, *GIS in Transportation*, in [Taylor e Francis 1999].
- [Sarawagi e altri 1998] S. Sarawagi, S. Thomas e R. Agrawal, "Integrating Association Rules Mining with Relational Database systems: Alternatives and Implications", in [SIGMOD 1998].
- [Savasere e altri 1995] A. Savasere, E. Omiecinski e S. Navathe, "An Efficient Algorithm for Mining Association Rules", in [VLDB 1995].
- [Schatz 1995] B. Schatz, "Information Analysis in the Net: The Interspace of the Twenty-First Century", *Keynote Plenary Lecture at American Society for Information Science (ASIS) Annual Meeting*, Chicago, (11 Ottobre 1995).
- [Schatz 1997] B. Schatz, "Information Retrieval in Digital Libraries: Bringing Search to the Net", *Science*, Volume 275 (17 Gennaio 1997).
- [Scheuermann e altri 1979] P. Scheuermann, G. Schiffner e H. Weber, "Abstraction Capabilities and Invariant Properties Modeling within the Entity-Relationship Approach", in [ER Conference 1979].
- [Schmidt e Swenson 1975] J. Schmidt e J. Swenson, "On the Semantics of the Relational Model", in [SIGMOD 1975].
- [Senko 1975] M. Senko, "Specification of Stored Data Structures and Desired Output in DIAM II with FORAL", in [VLDB 1975].
- [Senko 1980] M. Senko, "A Query Maintenance Language for the Data Independent Accessing Model II", *Information Systems*, Volume 5, number 4 (1980).
- [SIGMOD 1974] SIGMOD, *Proceedings of the ACM SIGMOD-SIGFIDET Conference on Data Description, Access, and Control*, a cura di R. Rustin (Maggio 1974).
- [SIGMOD 1975] SIGMOD, *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data*, a cura di F. King, San José, CA (Maggio 1975). ACM SIGMOD.
- [SIGMOD 1978] SIGMOD, *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*, a cura di E. Lowenthal e N. Dale, Austin, TX (Maggio-Giugno 1978).
- [SIGMOD 1979] SIGMOD, *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, a cura di P. Bernstein, Boston, MA (Maggio-Giugno 1979).

[VLDB 1994] VLDB, *Proceedings of the 20th International Conference on Very Large Data Bases*, a cura di J. Bocca, M. Jarke e C. Zaniolo, Santiago, Cile (Settembre 1994).

[VLDB 1995] VLDB, *Proceedings of the 21st International Conference on Very Large Data Bases*, a cura di U. Dayal, P. M. D. Gray e S. Nishio, Zurich, (Settembre 1995).

[VLDB 1996] VLDB, *Proceedings of the 22nd International Conference on Very Large Data Bases*, a cura di T. M. Vijayaraman, A. P. Buchman, C. Mohan e N. L. Sarda, Bombay, (Settembre 1996).

[VLDB 1997] VLDB, *Proceedings of the 23rd International Conference on Very Large Data Bases*, a cura di M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky e P. Loucopoulos, Zurich (Settembre 1997).

[Wallace 1995] D. Wallace, "1994 William Allan Award Address: Mitochondrial DNA Variation in Human Evolution, Degenerative Disease, and Aging", *American Journal of Human Genetics*, Number 57, pagg. 201-223 (1995).

[Wang 1990] K. Wang, 2Polynomial Time Designs Toward Both BCNF and Efficient Data Manipulation," in [SIGMOD 1990].

[Weiss e Indurkhy 1998] S. Weiss e N. Indurkhy, *Predictive Data Mining: A Practical Guide*, Morgan Kaufmann (1998).

[Widom 1995] J. Widom, "Research Problems in Data Warehousing", *CIKM* (Novembre 1995).

[Wiederhold 1983] G. Wiederhold, *Database Design*, II ed., McGraw-Hill (1983).

[Wiederhold 1995] G. Wiederhold, "Digital Libraries, Value, and Productivity", *CACM*, (Aprile 1995).

[Wiederhold e Elmasri 1979] G. Wiederhold e R. Elmasri, "The Structural Model for Database Design", in [ER Conference 1979].

[Wirth 1972] N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall (1972).

[Zobel e altri 1992] J. Zobel, A. Moffat e R. Sacks-Davis, "An Efficient Indexing Technique for Full-Text Database Systems", in [VLDB 1992].

Indice analitico

A

- aggregazione, 96, 105-107
- albero
 - B, 178-180, 189
 - ordine, 178
 - B+, 181-189
 - divisione, 186
 - inserimento, 184-186
 - ordine, 181-184
 - ricerca, 184
 - struttura di, 181-184
 - bilanciato, 177
- algebra relazionale, 215-230
 - altre operazioni, 231-236
 - espressione, 219
 - insieme completo, 228
 - interrogazioni esemplificative, 236-238
- algoritmi
 - di chiave, 356
 - di decomposizione, 355
 - di sintesi, 350, 355-356
 - genetici, 426-427
- ALL (SQL), 264, 268
- allocazione
 - di spazio su disco, 135
 - collegata, 135
 - contigua, 135
 - indicizzata, 135

- ALTER TABLE (SQL), 257-258
- ambiguità in interrogazioni SQL
 - in interrogazioni nidificate, 269
 - in nomi di attributi, 261
- amministratore della base di dati (DBA), 10
- analisi dei dati in linea (OLAP), 400
- anomalie di aggiornamento, 314-316
 - anomalie di cancellazione, 316
 - anomalie di inserimento, 314-315
 - anomalie di modifica, 316
- ANSI (American National Standards Institute), 27n, 40, 250
- ANY (SQL), 268
- applicazione, 6
 - di basi di dati, 7
- architettura
 - a tre livelli, 27-29
- ANSI/SPARC, 27n
- RAID 125-129
- area (su disco), 152
- AS (SQL), 262
- assegnazione di pseudonimi (SQL), 261-262
- asserzione, 289-290
- assiomi (regole di inferenza) di Armstrong, 325
- associazione, 53-59
 - binaria, 54-58, 98
- classe/sottoclasse, 77

di EQUIJOIN, 226
 di INTERSEZIONE, 222
 di JOIN ESTERNO, 234-236
 di JOIN NATURALE, 226
 di JOIN, 225-228, 234-236
 di PRODOTTO CARTESIANO, 222
 di PRODOTTO INCROCIATO, 222
 di PROIEZIONE, 218-219
 di recupero (su file), 136
 di RIDENOMINAZIONE, 220-221
 di SELEZIONE, 216-218
 di THETA JOIN, 226
 di UNIONE, 222
 di UNIONE ESTERNA, 235
 outer join in sql, 275
 operazioni
 aritmetiche in SQL, 265-267
 di join in SQL, 260, 274-275
 di un modello di dati, 24
 insiemistiche (relazionali), 221-225
 Oracle
 architettura, 365-367
 dizionario dei dati, 371-372
 istanza, 366-367
 metodi, 373
 oggetti dello schema, 369-371
 organizzazione della memoria, 374-377
 PL/SQL, 378-380
 PRO*C, 382-384
 processi, 367-368
 programmazione delle applicazioni, 379-384
 segmento, 375, 376-377
 segmento dei dati, 375, 376-377
 spazi delle tabelle, 374-375
 SQL, 372
 strumenti, 385
 struttura della base di dati, 365-367
 trigger, 373
 WebServer, 437-438
 orario, tipo di dati (SQL), 252

P

PARADOX, 364
 partecipazione
 parziale, 58
 totale, 58
 PL/SQL, 378-380
 Powerbuilder, 385
 precompilatore, 35

PRO*C (Oracle), 382-384
 processo
 di generalizzazione, 87-88
 di normalizzazione, 328
 di specializzazione, 87-88
 di esecuzione della base di dati, 34
 profondità
 globale (nell'hash estendibile), 149
 locale (hash), 150
 progettazione di basi di dati relazionali
 traduzione da EER in relazionale, 303-307
 traduzione da ER in relazionale, 298-302
 progettazione di una base di dati
 concettuale 42-46
 raffinamento, 61
 fisica di una base di dati, 43, 44
 logica di una base di dati, 43, 44
 progettista di basi di dati, 11
 programma
 applicativo, 3
 di utilità, 35
 programmatore di applicazioni, 12
 proiezione di un insieme di DF, 349
 proprietà
 di classe, 103
 di conservazione degli attributi, 348
 di join senza perdita (non-additivo), 328
 proprietario identificante, 60
 protocollo per il trasferimento di ipertesti
 (http), 434
 puntatore
 a record, 147
 a un albero, 178
 ai dati, 178, 182
 punto ancora, 164

Q

qualificazione dei nomi (SQL), 261-262
 quantificatore in SQL, 271-273

R

raccolta/analisi dei requisiti, 42-44
 raggruppamento (clustering), 231-232
 in data mining, 427
 in SQL, 277-280
 rapporto di cardinalità, 57
 rappresentazione della conoscenza (KR), 102-

RDB, 364
 read (comando di accesso a file), 137
 record, 131
 a lunghezza fissa, 132-134
 a lunghezza variabile, 132-134
 ancora, 164
 formato, 131-134
 organizzazione
 di record spanned, 134
 di record unspanned, 134
 ripartizione in blocchi (su disco), 134-135
 tipo di, 131
 recuperare delle informazioni, 443
 regola
 di arricchimento per DF, 323
 di associazione, 418-423
 di inferenza per DF, 323-326
 regressione lineare, 424-425
 relazione, 201
 associazione, 302
 entità, 302
 grado, 201
 in Access388-391
 nidificata, 331
 riferita, 211
 universale, 320, 347
 report (Access), 394-395
 reset (comando di accesso a file), 137
 rete neurale, 425-426
 reticolto
 di generalizzazioni, 84-87
 di specializzazioni, 84-87
 retroflusso di dati (backflushing), 408
 ricerca
 albero di, 176-178
 binaria, 140
 campo di, 177
 lineare, 136, 137
 ridondanza, 13-14
 controllata, 14
 riga, 200, 251
 RIGHT OUTER JOIN in SQL, 275
 RIM, 364
 rinominazione in SQL, 261-262, 274
 ripristino, 17
 risoluzione delle collisioni, 145
 ritardo di rotazione, 123, 473
 ruolo (di partecipazione ad un'associazione),
 56

S

scan (comando di accesso a file), 137
 schema, 25
 a fiocco di neve, 406
 a stella, 406
 concettuale, 28
 di relazione, 201
 elementi di, 251
 ER, 62
 esterno, 28
 in SQL, 25
 interno, 27
 modifica di, 251
 oggetti dello (Oracle), 369-371
 SDL (storage definition language) si veda
 linguaggio di definizione della memorizzazione
 SELECT (SQL), 258-261
 selettività del JOIN, 228
 semantica di relazione, 311-314
 SEQUEL (Structured English Query Language), 250
 server, 23
 SET DEFAULT (SQL), 255
 SET NULL (SQL), 255
 SGML (standard generalized markup language) si veda
 linguaggio standard di marcatura generalizzata
 sintesi relazionale, 350, 355-356
 sistema
 a utente singolo, 37
 operativo, 33
 sistemi di supporto alle decisioni (DSS), 400
 sistemi informativi geografici (GIS)
 Geographic Information Systems), 452-458
 sistemi informativi per dirigenti sito Web, 434
 software di comunicazione, 36
 SOME (SQL), 268
 sottoutilizzo, 175
 specializzazione, 78-81
 definita dall'utente, 83
 definita tramite un attributo, 83
 definita tramite un predicato, 82
 parziale, 83
 vincoli, 82-84
 totale, 83
 SQL, linguaggio

concetto di catalogo 251
 concetto di schema, 251
 DDL (definition dei dati), 249-252
 encapsulato, 291
 incorporato (Oracle), 372
 interrogazione, 258-267
 tipi di dati, 252-254
 vincoli, 252-256, 289-290
 viste, 285-289
 iQL1, 250
 iQL2, 251-291
 iQL3, 250
 iQL-92, 250
 iQL_ERROR, 382
 iQL Server, 364
 tato
 della base di dati, 26
 di relazione, 201
 valido, 27
 strumenti CASE, 469
 SUM (SQL), 276
 superchiave, 206, 329
 SYBASe, 364
 SYSTEM R, 250

I

abella, 200
 base, 256
 collegata con i join (SQL), 274-275
 come un insieme in SQL, 263-265
 dei fatti, 405
 delle dimensioni, 405
 di definizione (di una vista), 286
 in SQL, 257
 virtuale, 285
 relazione, 257
 tasse di trasferimento
 di disco, 474
 di una mole di dati, 123, 475
 tempo
 di posizionamento, 123, 473
 di riscrittura, 475
 di trasferimento di blocco (di disco), 123, 474
 tipi di vincoli
 di chiave, 206-207
 sui valori nulli, 207-208
 sul dominio, 206
 tipo di associazione, 53

tipo di dati, 5, 201
 a stringhe di bit (SQL), 252
 a stringhe di caratteri in SQL, 252
 INTERVAL (SQL), 254
 nei record, 131
 numerici in SQL, 252
 numerici interi (SQL), 252
 per data in SQL, 252-254
 timestamp (SQL), 252
 traccia (di disco), 120
 traduzione
 dello schema EER in relazionale, 303-307
 dello schema ER in relazionale, 298-302
 di categorie (EER) in relazionale, 306-307
 di generalizzazioni in relazionale, 303-306
 di specializzazioni in relazionale, 303-306
 di tipi unione (EER) in relazionale, 306-307
 transazione standard, 11
 tupla, 201-205
 dangling, 357-359
 riferente, 211
 sacca, 219
 spuria, 317-319

U

UML (Universal Modeling Language), 95-98
 UNIFY, 364
 UNION (SQL), 264
 UNIQUE (SQL) 255, 271
 Universo del Discorso (UoD), 2
 UPDATE (SQL), 284-285
 URL (uniform resource locator) si veda
 localizzatore universale di risorse
 utente
 casuale, 11
 di basi di dati, 11
 finale, 11-12
 indipendente, 11
 non sofisticato, 11
 sofisticato, 11
 vista, 9
 utilità di caricamento, 35

V

valore
 atomico di attributo, 200, 204, 330
 null, 48, 204

VDL (view definition language) si veda
 linguaggio di definizione delle viste
 verifica del join senza perdita, 351
 vincoli di chiave
 nel modello ER, 50
 nel modello relazionale, 206-207, 329
 vincoli di integrità, 16
 in SQL, 252-256, 289-290
 nel modello relazionale, 206-215
 violazioni, 212-215
 vincolo
 di completezza, 83
 di disgiunzione, 83
 di partecipazione, 58-59
 di sovrapposizione (specializzazione), 83
 dichiarativo, 289-290
 min-max, 65-66, 101
 sui valori nulli, 208
 vista (relazionale)
 aggiornamento, 287-289
 definizione, 286

implementazione, 287-289
 in SQL, 285-289
 visualizzazione
 drill-down, 405
 roll-up, 404-405
 Struttura (Access), 387
 VM/CMS, 364
 voce di dato, 131
 VSAM (Virtual Storage Access Method), 194

W

WATCOM SQL, 364
 WHERE (SQL), 258-261
 WITH CHECK OPTION (SQL), 289

X

XDB, 364
 XML (extensible markup language) si veda
 linguaggio di marcatura estendibile