A

Il modello reticolare

A.1 I modelli dei dati basati sui puntatori

In buona parte di questo testo, abbiamo fatto riferimento al modello relazionale, che è il modello logico utilizzato nella maggioranza delle nuove realizzazioni di basi di dati. Come già detto nei capitoli 1 e 2, però, i sistemi relazionali hanno cominciato a diffondersi solo a partire dall'inizio degli anni Ottanta, mentre i DBMS sono stati utilizzati fin dalla fine degli anni Sessanta. Tuttora, una buona parte delle basi di dati operanti sono realizzate su DBMS concepiti prima dell'invenzione del modello relazionale. Esitono sostanzialmente altri due modelli logici, il modello *reticolare* e il modello *gerarchico*. Ricordiamo che, come discusso nel paragrafo 2.4, il modello relazionale è *basato su valori*, nel senso che le corrispondenze fra dati in relazioni diverse sono rappresentate per mezzo della presenza di valori comuni, mentre il modello reticolare e quello gerarchico sono basati su puntatori, utilizzati come riferimenti espliciti fra record di tipi diversi. Allo scopo di mostrare le caratteristiche essenziali di questi modelli, dedichiamo il presente capitolo al modello reticolare, che ha strutture, vincoli e linguaggio di accesso che si prestano ad una descrizione semplice e precisa. In effetti, i due modelli hanno caratteristiche simili e sarebbe eccessivo trattare entrambi in dettaglio. Peraltro, va notato che i sistemi gerarchici sono più diffusi di quelli reticolari.

È anche utile notare che il modello reticolare e quello gerarchico hanno avuto origini diverse rispetto al modello relazionale, che è stato definito in astratto, con realizzazioni effettive arrivate molti anni dopo: il modello reticolare e quello gerarchico sono stati definiti come astrazione delle caratteristiche fondamentali di sistemi esistenti. Di conseguenza, molte caratteristiche di questi due modelli ricordano da vicino tecniche di realizzazione fisica dei dati, piuttosto che caratteristiche intrinseche dei dati stessi, come avviene invece per il modello relazionale.

Un'altra importante differenza fra sistemi relazionali da una parte e sistemi reticolari e gerarchici dall'altra risiede nel fatto che nei primi le operazioni considerano globalmente insiemi di tuple, mentre nei secondi l'accesso viene effettuato considerando un record alla volta, e quindi la scansione di insiemi di record e delle loro connessioni con altri record, deve essere specificata esplicitamente dal programmatore per mezzo di cicli.

Il modello reticolare è stato recepito da un organismo di unificazione e ne esiste pertanto una versione standard (detta *CODASYL*, dal nome del comitato

preposto). Per la precisione, esistono varie versioni successive, approvate nel 1971, 1978 e 1981, rispettivamente.

I tre paragrafi di questo capitolo sono dedicati il primo a strutture e vincoli del modello reticolare, il secondo alle operazioni di accesso e manipolazione e il terzo alle modalità secondo cui, nella progettazione logica, si specifica la traduzione dal modello Entità-Relazione al modello reticolare. Poiché il nostro interesse è essenzialmente didattico, senza pretesa di completezza, presenteremo una versione semplificata e modificata del modello e del linguaggio.

A.2 Il modello reticolare: le strutture

Le strutture di dati utilizzate nel modello reticolare sono due, il *record*, con caratteristiche simili al concetto di relazione (o, più precisamente, al concetto di file di record in un linguaggio di programmazione, quale il Pascal o il Cobol), e il *set*, che permette di correlare record, per mezzo di catene di puntatori. Una base di dati reticolare è definita con riferimento ad uno schema, che contiene tipi record collegati fra loro da tipi set. Il modello reticolare è così chiamato poichè ogni suo schema può essere espressivamente rappresentato per mezzo di un grafo (o una rete, dall'inglese *network*), con i tipi record come nodi e i tipi set come archi.

A.2.1 Tipi record e record

Un *tipo record* è un tipo di dato composto, al quale è associato un *nome*. Esso è costituito da un insieme di *campi*. Ogni campo ha un nome che lo identifica all'interno del tipo record e ad esso è associato un tipo semplice,[1] detto *dominio* del campo. Un'*occorrenza di un tipo record* è una funzione che associa a ciascun campo un valore del corrispondente dominio. È evidente la somiglianza fra il concetto di tipo record e quello di schema di relazione del modello relazionale (o di tipo record in un qualsiasi linguaggio di programmazione) e fra il concetto di occorrenza di record e quello di tupla relazionale (e di record).

È possibile specificare una *chiave*[2] per ciascun tipo record, ma possono esistere tipi record senza chiave (accessibili, come vedremo, attraverso catene di puntatori o scansioni sequenziali basate su un ordinamento fisico esistente fra i record). In altri termini, possono quindi esistere occorrenze diverse di uno stesso tipo record con esattamente gli stessi valori. Quindi, a differenza di quanto accade per il modello relazionale (o meglio, per la sua formulazione astratta), per ciascun tipo record la base di dati contiene un *multiinsieme* di occorrenze, non necessariamene un *insieme*.[3]

[1] Assumiamo per semplicità che i campi abbiano solo tipi semplici, anche se in generale ciò non è vero. Peraltro, l'estensione non aggiungerebbe niente di significativo alla nostra discussione.

[2] Il concetto è qui logico e fisico al tempo stesso, perché su questa chiave si basa la struttura di memorizzazione. Alcune versioni del modello permettono, attraverso una tecnica piuttosto involuta, di definire altre chiavi, associate a cammini di accesso secondari.

[3] In effetti, si tratta poi quasi sempre di una *lista* di occorrenze, poiché esiste fra loro un ordinamento logico o fisico. Assumendo poi, come avviene in realtà nei sistemi, l'esistenza di

A.2.2 Tipi set e occorrenze di set

Il concetto di tipo record appena visto è in effetti simile a quello di relazione o di file. Ciò che è invece caratteristico del modello reticolare è la tecnica utilizzata per correlare record di vari tipi. Un *tipo set* è un legame logico fra due (o più, ma noi ci limitiamo per semplicità al caso binario) tipi record distinti, detti rispettivamente *owner* e *member* (o *membro*) del tipo set. Sia S un tipo set, avente come owner un tipo record O e come member un tipo record M. Un'*occorrenza del set* S è costituita da un'occorrenza del record O (owner) e da una lista di zero o più occorrenze del record M (member). Ciascuna occorrenza di O è owner di una e una sola occorrenza del set S, mentre ciascuna occorrenza di M partecipa al più ad una occorrenza di S. Riassumendo, le occorrenze del set S collegano ciascuna occorrenza dell'owner O a zero o più occorrenze del member M, con il vincolo che ciascuna occorrenza del member non può appartenere a due diverse occorrenze di S: quindi, S realizza una corrispondenza "uno a molti" (o più brevemente $1:n$) fra l'insieme delle occorrenze di O e l'insieme delle occorrenze di M. Questa caratteristica delle occorrenze dei tipi set ci permette di sostanziare quanto abbiamo affermato all'inizio del capitolo e cioè che l'organizzazione dei dati del modello reticolare ricorda più le strutture fisiche dei dati che le proprietà intrinseche dei dati stessi. Infatti, si può senz'altro dire che i set rappresentano corrispondenze di tipo $1:n$ perché è facile realizzarle fisicamente, ad esempio per mezzo di liste circolari; una implementazione efficiente viene ottenuta definendo, in ciascun record un campo puntatore per ciascun tipo set in cui è coinvolto. Se le relazioni fossero di tipo "molti a molti" (o $m:n$), la struttura di queste liste sarebbe molto più complessa.

Uno *schema di base di dati reticolare* è costituito da un insieme di tipi record (con nomi diversi) e un insieme di tipi set (con nomi diversi) definiti su tali tipi record. Una *base di dati reticolare*, relativa ad un certo schema reticolare, è costituita da un insieme di occorrenze di record per ciascun tipo record e da un insieme di occorrenze di set per ciascun tipo set.

È possibile rappresentare graficamente schemi reticolari in maniera espressiva: i tipi record vengono rappresentati per mezzo di simboli rettangolari contenenti all'interno i rispettivi nomi, mentre i tipi set vengono rappresentati per mezzo di linee, vicino alle quali sono riportati i rispettivi nomi e che congiungono i rispettivi owner e member, con una freccia orientata verso il member. In figura A.1 è rappresentato uno schema di base di dati che corrisponde allo schema della base di dati relazionale della figura 2.6, costituito dalle tre relazioni:

STUDENTI(Matricola, Cognome, Nome, Data di nascita)
ESAMI(Studente, Voto, Corso)
CORSI(Codice, Titolo, Docente)

Confrontando lo schema reticolare con quello relazionale, possiamo fare due osservazioni:

un indirizzo o di un identificatore interno (cioè gestito dal sistema) per ciascuna occorrenza, e considerando il record come costituito dai valori dei campi e dal valore dell'identificatore interno, si può dire che esista un insieme di occorrenze di record per ciascun tipo record.

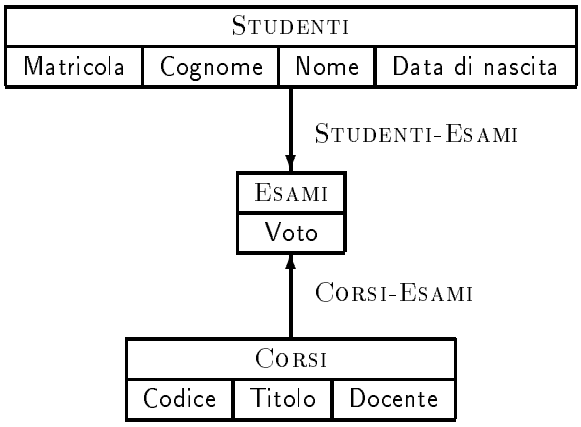

Figura A.1 Un semplice schema reticolare

- abbiamo introdotto due nomi per i tipi set, STUDENTI-ESAMI e CORSI-ESAMI, che non hanno un corrispettivo nello schema relazionale;
- non abbiamo, nello schema reticolare, campi corrispondenti agli attributi Studente e Corso della relazione ESAMI: si tratta degli attributi utilizzati per stabilire corrispondenze basate su valori, che vengono sostituite dai puntatori, come vedremo fra poco.

La figura A.2 mostra un'istanza della base di dati reticolare di figura A.1, con le stesse informazioni della base di dati relazionale di figura 2.6. Le occorrenze dei tipi set sono rappresentate in figura per mezzo di liste circolari per evidenziare le caratteristiche del concetto:

- nell'ambito di una occorrenza di un set, le occorrenze del member che partecipano sono ordinate (abbiamo appunto parlato di lista);
- la lista parte dall'occorrenza dell'owner e torna ad essa: si evidenzia così il fatto che da ciascuna occorrenza del member è possibile risalire all'owner

In effetti, la lista circolare è solo una delle possibili forme realizzative per un tipo set, ma non è necessario approfondire oltre. Esaminiamo in dettaglio la figura. Abbiamo cinque occorrenze del tipo record STUDENTI, corrispondenti alle tuple dell'omonima relazione, e per ciascuna di esse abbiamo un'occorrenza del tipo set STUDENTI-ESAMI. Delle occorrenze del tipo set, due sono vuote, cioè senza member (quelle che hanno come owner il secondo e il quarto record di STUDENTI), due hanno un solo member e una, la prima nella figura, ha due member. Al tempo stesso, ciascuna occorrenza di ESAMI appartiene ad una sola (in questo caso esattamente una) occorrenza del set: ciò conferma che le occorrenze di un tipo set stabiliscono una corrispondenza $1 : n$ fra le occorrenze dei tipi record coinvolti. Potremmo fare osservazioni analoghe per le occorrenze del tipo set CORSI-ESAMI.

Per fare ulteriori osservazioni, consideriamo un altro schema reticolare, mostrato nella figura A.3. In una base di dati su questo schema, si può senz'altro presentare la situazione in cui un'occorrenza di record member *non* partecipa ad

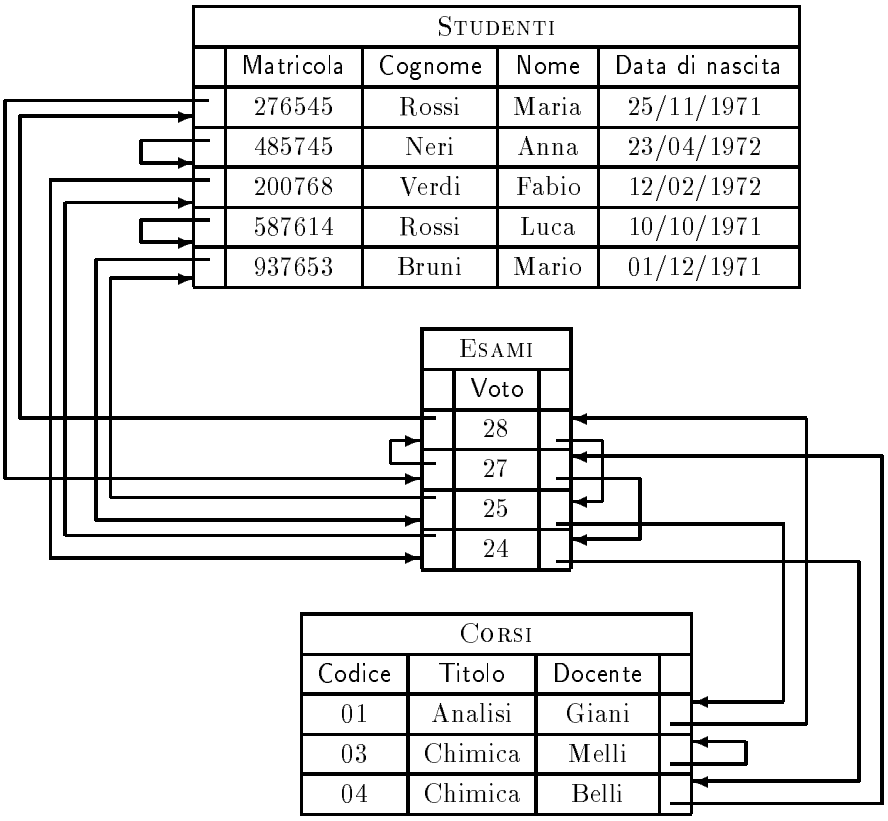

Figura A.2 Una base di dati reticolare

alcuna occorrenza di tipo set (abbiamo detto che può partecipare al massimo ad una e nell'esempio precedente partecipava sempre esattamente ad una). Il tipo set TESI stabilisce la corrispondenza fra laureandi (o laureati) e rispettivi relatori: se le occorrenze del tipo record STUDENTI includono anche studenti che non sono ancora laureandi, allora questi studenti non hanno un relatore e le occorrenze di STUDENTI ad essi relative non partecipano al tipo set TESI.

A.2.3 Vincoli di integrità

Abbiamo già visto che è possibile definire chiavi per i tipi record. Non aggiungiamo altro su questo punto, soffermandoci invece sui vincoli di integrità che possono essere definiti sui tipi set.

Per definizione, dato un tipo set S con owner O e member M, ogni occorrenza di O partecipa ad una e una sola occorrenza di S, mentre ogni occorrenza di M

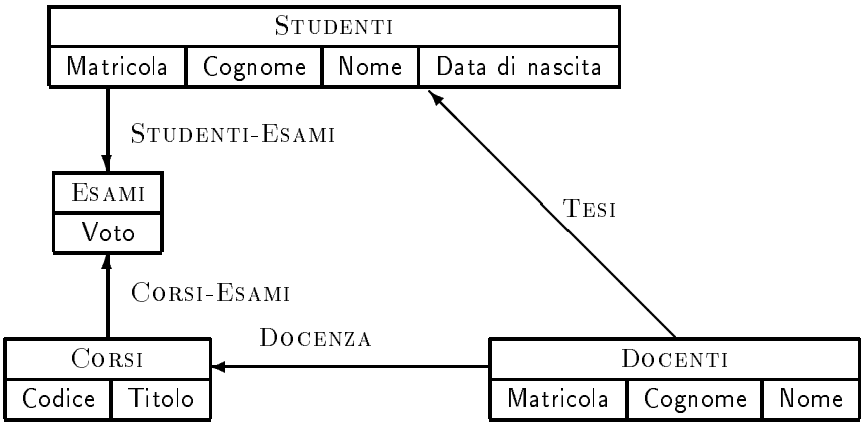

Figura A.3 Un altro schema reticolare

partecipa al più ad una occorrenza di S. Da questo punto di vista, l'unica proprietà effettivamente specificabile per mezzo di vincoli di integrità è l'*obbligatorietà* della partecipazione delle occorrenze di M ad occorrenze di S. Abbiamo visto negli esempi del paragrafo precedente come possano esistere casi in cui tutte le occorrenze del member partecipano al set (le occorrenze di ESAMI nei set STUDENTI-ESAMI e CORSI-ESAMI nelle figure A.1 e A.3) e casi in cui alcune occorrenze del member non partecipano al set (le occorrenze di STUDENTI nel set TESI in figura A.3). Concettualmente, questo vincolo dovrebbe essere specificato dichiarando, per ogni tipo set, se la partecipazione delle occorrenze del member al set è *obbligatoria* o *facoltativa*. In concreto, nella versione standard (e nelle implementazioni) del modello reticolare, esso viene specificato, in modo abbastanza involuto, attraverso due proprietà, associate ai set, l'*opzione di inserimento* e l'*opzione di ritenzione* (in inglese, *insertion* e *retention option*). Queste due proprietà descrivono il vincolo della obbligatorietà dal punto di vista dinamico, imponendo restrizioni o automatismi sulle operazioni di creazione di occorrenze del member e di connessione e disconnessione dalle occorrenze del set.

L'*opzione di inserimento* ha due possibili valori, *automatico*, che specifica che le occorrenze del record member vengono automaticamente inserite in occorrenze del set al momento in cui vengono inserite nella base di dati, e *manuale*, che specifica che ciò non accade, e quindi è necessario un inserimento "manuale," cioè esplicito, per mezzo di una specifica operazione. Nel caso di inserimento automatico, l'occorrenza del set in cui l'occorrenza del record viene inserita è selezionata secondo criteri che vedremo più avanti, quando parleremo del DDL e poi del DML del modello reticolare.

L'*opzione di ritenzione* ha tre possibili valori:

- *obbligatorio* (in inglese *mandatory*), che specifica che le occorrenze del record member, una volta inserite (automaticamente o manualmente) in un'occorrenza di set debbono sempre appartenere ad un'occorrenza di set (eventualmente diversa nel tempo), fino alla loro cancellazione dalla base di dati.

- *fisso* (*fixed*), come il precedente, con in più la restrizione di dover appartenere sempre alla stessa occorrenza di set.
- *opzionale* (*optional*), che non impone alcuna restrizione.

In linea di principio, le opzioni di inserimento e ritenzione sono indipendenti l'una dall'altra. D'altra parte, è abbastanza evidente che le combinazioni manuale-obbligatorio e manuale-fisso sono raramente significative, poiché specificano una partecipazione facoltativa all'inizio e rigidamente obbligatoria dopo la prima connessione ad un'occorrenza di set. Un discorso simile, ma meno forte, si può fare per la combinazione automatico-opzionale, poiché possono esistere casi in cui può risultare utile. Poiché anche il valore *fisso* per l'opzione di ritenzione non è di uso frequente (ad esempio, non permette di effettuare correzioni relativamente all'appartenenza ad occorrenze di set), possiamo dire che le combinazioni di gran lunga più importanti sono automatico-obbligatorio (automatic-mandatory) e manuale-opzionale (manual-optional).

A.2.4 DDL per il modello reticolare

Presentiamo ora gli aspetti essenziali del linguaggio per la definizione degli schemi nel modello reticolare. Omettiamo molti dettagli, che vanno al di là degli scopi di questo testo. Tra l'altro, pur essendo il modello e i relativi linguaggi standardizzati, alcuni aspetti variano da implementazione a implementazione. Inoltre, poiché alcune sfumature della sintassi del DDL CODASYL richiamano il Cobol, linguaggio la cui conoscenza non viene qui assunta come prerequisito, abbiamo apportato alcune modifiche volte a rendere più agevole la comprensione ad un lettore che conosca (ad esempio) il Pascal.

La sintassi è mostrata in figura A.4, dove sono omesse le produzioni relative ad alcuni simboli non terminali, ad esempio quelle degli identificatori. La dichiarazione di uno schema è costituita da una intestazione seguita dalle dichiarazioni di record e set. Supponiamo per semplicità che le dichiarazioni di record debbano tutte precedere le dichiarazioni di set.

La dichiarazione di record contiene la lista dei campi con i relativi tipi (che assumiamo semplici) e la strategia di memorizzazione. Quest'ultima prevede due alternative:

- `calc` indica una memorizzazione basata su una funzione hash sui valori del campo o dei campi indicati. La clausola `duplicates not allowed` specifica che i campi coinvolti costituiscono una chiave per il record. In generale diciamo che tali campi costituscono la *pseudochiave* del tipo record.
- `via` S `set`, dove S è un tipo set di cui il record in questione è member, specifica una memorizzazione delle occorrenze del record il più vicino possibile alle occorrenze del record owner del set, raggruppate secondo l'appartenenza alle occorrenze del set stesso.

Chiaramente, la prima alternativa privilegia gli accessi diretti alle occorrenze dei record (sulla base dei valori dei campi pseudochiave), mentre la seconda privilegia le scansioni delle occorrenze dei set. È opportuno notare come questa porzione

```
DichiarazioneSchema ::=
        schema name is NomeSchema
              { DichiarazioneRecord }
              { DichiarazioneSet }
        end
DichiarazioneRecord ::=
        record name is NomeRecord
              location mode is CriterioDiMemorizzazione
              {DichiarazioneCampo}
        end
CriterioDiMemorizzazione ::=
        calc using ListaCampi [ duplicates not allowed ] |
        via NomeSet set
DichiarazioneCampo ::= NomeCampo :   TipoSemplice
TipoSemplice ::= integer | string Lunghezza | date
DichiarazioneSet ::=
        set name is NomeSet
              owner is NomeRecord
              member is NomeRecord
                    OpzioneDiInserimento OpzioneDiRitenzione
                    order is CriterioDiOrdinamento
        end
OpzioneDiInserimento ::= manual | automatic
OpzioneDiRitenzione ::= mandatory | fixed | optional
CriterioDiOrdinamento ::= next | prior | sorted by ListaCampi
```

Figura A.4 Sintassi di un DDL didattico per il modello reticolare

della dichiarazione corrisponda a informazioni di livello più basso rispetto a quelle che dovrebbero corrispondere ad uno schema logico. Ciò è dovuto a due ordini di ragioni. In primo luogo, quando il modello reticolare fu definito e standardizzato, le idee ora accettate sull'indipendenza dei dati e quindi sulla trasparenza completa delle caratteristiche fisiche non erano state ancora comprese fino in fondo. Al tempo stesso (e come conseguenza) le operazioni di manipolazione (che vedremo nel prossimo paragrafo) fanno riferimento anche ad aspetti fisici (ad esempio l'accesso ai record sulla base dei valori dei campi viene specificato in modo diverso a seconda che sui campi interessati sia definito un metodo d'accesso diretto, oppure no).

Per quanto riguarda i tipi set, vengono specificati una serie di aspetti. Riguardo all'owner, al member e alle opzioni di inserimento e ritenzione non abbiamo niente da aggiungere a quanto già detto. La strategia di ordinamento (che presentiamo in versione semplificata) merita qualche parola. Essa specifica il criterio secondo il quale debbono essere ordinate le occorrenze del record member all'interno di ciascuna occorrenza del set (ricordiamo che esse formano una lista e quindi deve sempre esistere un ordinamento totale fra di esse). L'ordinamento viene man-

tenuto dal sistema, che all'atto dell'inserimento pone ciascuna occorrenza di record nella corretta posizione. L'opzione `sorted` specifica che le occorrenze del member all'interno di ciascuna occorrenza del set sono ordinate secondo il valore di uno o più campi. Le altre due possibilità si basano su una caratteristica fondamentale dei linguaggi di accesso a basi di dati reticolari, cioè, come vedremo meglio più avanti, il fatto che di volta in volta esista una sorta di "puntatore corrente" all'ultima occorrenza di record visitata all'interno di un set. L' opzione `next` (`prior`) specifica che il nuovo record deve essere inserito subito dopo (subito prima) il record corrente.

Mostriamo nella figura A.5 la dichiarazione nel DDL didattico dello schema reticolare della figura A.3.

A.3 Operazioni su basi di dati reticolari

Vediamo prima le operazioni di ricerca e interrogazione e poi quelle di aggiornamento, premettendo al tutto una discussione generale, valida per tutti gli aspetti dei linguaggi di manipolazione.

A.3.1 La navigazione su una base di dati

La caratteristica fondamentale dei linguaggi per il modello reticolare è quella di essere basati su una visita della base di dati, effettuata seguendo i collegamenti stabilite le occorrenze dei set. Sono ovviamente possibili anche accessi diretti o scansioni sequenziali, ma l'operazione fondamentale per la visita di una base di dati reticolare è quella di *navigazione* sulla base di dati. Questa navigazione avviene accedendo a singoli record, quindi ad un livello più basso di quanto non avvenga nel modello relazionale, in cui, come abbiamo visto, le operazioni elementari coinvolgono intere relazioni (o comunque insiemi di tuple). Per questo motivo, l'accesso a basi di dati reticolari avviene normalmente attraverso un *linguaggio ospite* cioè un linguaggio di programmazione tradizionale (per esempio COBOL o PL1) esteso in modo tale che sia possibile specificare, oltre alle istruzioni ordinarie, anche i comandi DML.

In questo paragrafo, vediamo una versione adattata e semplificata del DML reticolare e del modo in cui le relative istruzioni vengono immerse in un programma in linguaggio ospite. Fra l'altro, utilizziamo il Pascal come linguaggio ospite, anche se ciò non avviene nelle applicazioni reali. Inoltre assumiamo (e questo è molto vicino a ciò che avviene nei sistemi reali) che, per ogni tipo record definito nello schema, esista nel programma una variabile, di tipo record, con lo stesso nome e gli stessi campi: essa svolge lo stesso ruolo del buffer in una variabile Pascal di tipo file, cioè quello di permettere lo scambio di dati fra programma e base di dati. Infine, poiché le operazioni di ricerca e scansione sequenziale di record all'interno di tipi record e tipi set possono avere esito positivo o negativo (cioè portare all'accesso ad un record oppure no), assumiamo che esista una variabile globale booleana, di nome `db-status`, che assume valore *true* dopo l'esecuzione di una operazione di navigazione se e solo se essa ha portato all'individuazione di un record.

```
schema name is Universita
  record name is Studenti
    location mode is calc using Matricola duplicates not allowed
    Matricola       : integer
    Cognome         : string 20
    Nome            : string 10
    DataDiNascita   : date
  end
 name is Corsi
    location mode is calc using Codice duplicates not allowed
    Codice       : string 20
    Titolo       : string 20
  end
  record name is Docenti
    location mode is calc using Cognome
    Matricola       : integer
    Cognome         : string 20
    Nome            : string 10
  end
  record name is Esami
    location mode is via set Studenti-Esami
    Voto            : integer
  end
  set name is Studenti-Esami
    owner is Studenti
    member is Esami mandatory automatic
    order is next
  end
  set name is Corsi-Esami
    owner is Corsi
    member is Esami mandatory automatic
    order is next
  end
  set name is Docenza
    owner is Docenti
    member is Corsi optional manual
    order is next
  end
  set name is Tesi
    owner is Docenti
    member is Studenti optional manual
    order is sorted by Cognome, Nome
  end
end
```

Figura A.5 Specifica in DDL dello schema in figura A.3

Un concetto fondamentale nel linguaggio di manipolazione reticolare è quello di *record corrente*:[4] poiché la navigazione può comprendere scansioni e diramazioni, è in molti casi utile conservare traccia della posizione attuale nella visita di un certo set, o delle occorrenze di un tipo record, oppure della posizione nella quale una certa visita è stata interrotta e deve essere proseguita, e così via. I programmi in linguaggio ospite sono in grado di ricordare varie posizioni sulla base di dati, per mezzo di puntatori ad occorrenze di record, che vengono detti appunto record correnti. Esistono vari tipi di record correnti:

- Il record corrente del programma: l'ultimo record visitato, qualunque sia il suo tipo.
- Il record corrente per ciascun tipo record R: l'ultima occorrenza di R visitata, che viene quindi detta occorrenza corrente di R.
- Il record corrente per ciascun tipo set S: l'ultimo visitato fra i record che sono owner o member delle occorrenze S.

Inoltre, viene definita l'occorrenza corrente per ciascun tipo set:

- l'occorrenza del set cui appartiene il record corrente per il set stesso

All'inizio dell'esecuzione del programma, gli indicatori di correnza sono indefiniti, come variabili di un programma in linguaggio ad alto livello a cui non sia stato ancora assegnato un valore.

A.3.2 Le operazioni di ricerca

L'istruzione fondamentale per il trasferimento di dati dalla base di dati al programma (e di conseguenza all'utente) è l'istruzione `get` (la cui sintassi semplificata richiede la sola parola chiave `get` senza altri elementi), che specifica la copia del contenuto del record corrente del programma nel buffer corrispondente al suo tipo. Vedremo esempi più completi dopo aver introdotto il comando `find`, comunque, per dare una prima idea concreta, diciamo che se il record corrente del programma è una certa occorrenza del record STUDENTI, allora il contenuto (cioè i valori dei vari campi) di tale record vengono copiati nel buffer associato al tipo record STUDENTI.

La navigazione sulla base di dati viene specificata per mezzo del comando `find`, che presenta diverse forme, corrispondenti a diverse modalità di visita della base di dati, che includono quelle tradizionali di accesso sequenziale o diretto ai record di un tipo, e quelle specifiche del modello reticolare, che prevedono la navigazione attraverso le occorrenze dei set.

L'accesso sequenziale a tutti i record di un certo tipo viene specificato per mezzo delle istruzioni:[5]

[4] Più precisamente, si dovrebbe parlare di *occorrenza corrente*, ma, forse per evitare cacofonie, forse per semplicità, prevale l'altra terminologia.

[5] Esistono alcune varianti che trascuriamo.

`find first` *NomeDiRecord*

`find next` *NomeDiRecord*

che, rispettivamente, causano l'accesso al primo record del tipo indicato e a quello successivo rispetto a quello corrente. Considerando lo schema di figura A.3, la scansione sequenziale del tipo record STUDENTI, per esempio con lo scopo di stampare i nomi e i cognomi di tutti gli studenti, può venire specificata per mezzo del seguente frammento di programma:

```
find first Studenti
while db-status
do begin
      get ;
      writeln (Studenti.Nome, Studenti.Cognome) ;
      find next Studenti
     end
```

che, rispetto alla base di dati di figura A.2, produce il seguente output:

```
Maria Rossi
Anna Neri
Fabio Verdi
Luca Rossi
Mario Bruni
```

Vale la pena sottolineare che, in SQL, avremmo specificato la stessa interrogazione per mezzo di una semplice istruzione, senza cicli:

```
select Nome, Cognome
from Studenti
```

L'accesso diretto è possibile solo per record la cui strategia di memorizzazione sia `calc` e solo attraverso i valori dei campi che costituiscono la pseudochiave. Esso viene specificato per mezzo dell'istruzione:

`find any` *NomeRecord*

che permette l'accesso al primo (o all'unico, nel caso in cui sia stata specificata la clausola `duplicates not allowed`, e quindi la pseudochiave sia in effetti una chiave) dei record il cui valore della pseudochiave sia uguale al valore dei campi corrispondenti nel buffer del tipo record. Nel caso di pseudochiave, si accede ai successivi record con lo stesso valore per mezzo dell'istruzione:

`find duplicate` *NomeRecord*

da ripetersi finché necessario. Con riferimento allo schema sopra discusso, i campi Matricola e Codice sono chiave rispettivamente per STUDENTI e CORSI, mentre Cognome è una pseudochiave per IMPIEGATO. Se 200768, è il valore attuale del campo di buffer STUDENTI.Matricola, allora l'istruzione

```
find any Studenti
```

specifica un accesso al record con valore di Matricola pari a 200768 (se esiste), cioè lo fa diventare corrente per il programma, per il tipo record STUDENTI e per i tipi set STUDENTI-ESAMI e TESI. Poiché Matricola è chiave, non è possibile che vi sia più di un record con la stessa Matricola. Se invece non esiste alcun record con valore della chiave pari a 200768, allora si produce una condizione di "non trovato", che pone a *false* la variabile `db-status` e rende indefinito il record corrente di programma (gli altri indicatori di correnza rimangono inalterati). Infine, un'istruzione

```
find any Docenti
```

causa l'accesso ad uno dei record con valori del campo Cognome pari al valore assegnato al corrispondente campo di buffer e per accedere ai successivi è necessario specificare

```
find duplicate Docenti
```

all'interno di un ciclo governato da una verifica della variabile `db-status`.

È importante notare come le varie ricerche, sequenziali o dirette, su campi pseudochiave oppure no, debbano essere specificate in modo diverso, mentre in SQL il formato è unico. Ad esempio, con riferimento allo schema in figura A.3, per trovare gli studenti con un dato cognome, dobbiamo fare una scansione sequenziale, seguita da una verifica:

```
find first Studenti
while db-status
do begin
      get ;
      if Studenti.Cognome = 'Rossi'
        then writeln (Studenti.Nome, Studenti.Cognome) ;
      find next Studenti
   end
```

mentre se Cognome fosse stato la pseudochiave, avremmo potuto scrivere

```
Studenti.Cognome = 'Rossi'
find any Studenti
while db-status
do begin
      get ;
      if writeln (Studenti.Nome, Studenti.Cognome) ;
      find duplicate Studenti
   end
```

ottenendo un'esecuzione più efficiente. In SQL, invece, avremmo comunque scritto

```
select Nome, Cognome
from Studenti
where Cognome = 'Rossi'
```

e il sistema avrebbe individuato la modalità di esecuzione più conveniente, in base alle strutture fisiche.

Le operazioni di ricerca più interessanti, in quanto tipiche del modello reticolare, sono quelle effettuate seguendo i legami realizzati dalle occorrenze dei set. L'accesso all'owner dell'occorrenza corrente di un set viene specificato per mezzo di un'istruzione del tipo

`find owner within` *NomeSet*

mentre la scansione dei membri dell'occorrenza corrente di un set viene specificata per mezzo delle istruzioni

`find first` *NomeRecord* `within` *NomeSet*

`find next` *NomeRecord* `within` *NomeSet*

Vediamo alcuni esempi di navigazione sulla base di dati delle figura A.3. La lista dei professori, con l'indicazione, per ciascuno degli studenti che stanno svolgendo la tesi sotto la sua supervisione, può essere effettuata scandendo sequenzialmente i record di tipo DOCENTI e, per ciascuno di essi, visitando l'occorrenza del set TESI corrispondente, con il seguente frammento di programma:

```
find first Docenti ;
   while db-status
   do begin
            get ;
            write (Docenti.Cognome, Docenti.Nome) ;
            find first Studenti within Tesi ;
            while db-status
            do begin
                     get ;
                     write (Studenti.Cognome, Studenti.Nome) ;
                     find next Studenti within Tesi
                end ;
            writeln ;
            find next Docenti ;
       end
```

A parte l'organizzazione della stampa, in SQL (supponendo un attributo Relatore in STUDENTI che faccia riferimento al relatore) con avremmo scritto un'interrogazione molto più compatta, che tra l'altro ha il pregio di separare l'accesso ai dati dalla stampa:

```
select Docenti.Cognome, Docenti.Nome,
       Studenti.Cognome, Studenti.Nome
from Studenti, Docenti
where Studenti.Relatore = Docenti.Matricola
```

La stampa degli esami superati da uno studente (data la matricola), con il nome del corso corrispondente e con il voto riportato, può essere generata scandendo il set di cui l'occorrenza di STUDENTI è owner e, per ciascuno dei member, accedendo al record CORSI che è owner della corrispondente occorrenza di CORSI-ESAMI:

```
Studenti.Matricola =....
find any Studenti ;
   if db-status
   then begin
            get ; writeln (Studenti.Cognome) ;
            find first Esami within Studenti-Esami ;
            while db-status
            do begin
                 get ;
                 find owner within Corsi-Esami ;
                 if db-status
                 then begin
                        get ;
                        writeln ( Corsi.Titolo, Esami.Voto )
                      end ;
                 find next Esami within Studenti-Esami
        end
```

In alcune operazioni di scansione e ricerca, che richiedano più o meno contemporaneamente l'accesso a più occorrenze di uno stesso record, le funzionalità che abbiamo finora visto per la navigazione risultano inadeguate. Supponiamo ad esempio di voler stampare, sempre con riferimento allo schema in figura A.3, i cognomi di tutti i docenti, con tutti i dati relativi agli esami dei corsi da essi tenuti, con studente, voto ed eventuale relatore dello studente. Intuitivamente, il risultato desiderato si può ottenere scandendo sequenzialmente il tipo record DOCENTI, e visitando i set DOCENZA, CORSI-ESAMI, STUDENTI-ESAMI e TESI, nell'ordine. Per ciascuna di queste visite, è necessario mantenere traccia dell'ultimo record esaminato, per poter passare al successivo. Gli indicatori di correnza hanno proprio lo scopo di gestire queste informazioni; in questo caso, però, con le caratteristiche viste finora, essi non permettono di risolvere facilmente il problema. Infatti, esiste un solo record corrente per il tipo DOCENTI, mentre ne sarebbero necessari due, uno per la scansione più esterna di tutti i docenti, e l'altro per accedere ai relatori degli studenti via via considerati nella scansione più interna. Proprio per gestire situazioni di questo tipo, l'istruzione `find` ha una clausola ulteriore, che permette di inibire l'aggiornamento di alcuni indicatori di correnza. Nel caso in esame, si potrebbe usare l'occorrenza corrente del record DOCENTI per la scansione esterna e quella del set TESI per quella più interna. Tutte le versioni dell'istruzione `find` permettono una clausola finale con la seguente sintassi:

ClausolaRetainingCurrency ::=
`retaining` *ListaNomiDiSet* `currency` |
`retaining` *NomeRecord* `currency` |
`retaining all currencies`

La semantica di queste istruzioni è suggerita dalla sintassi: gli indicatori di correnza citati (o tutti, nell'ultimo caso, con eccezione del record corrente del programma) non vengono aggiornati e mantengono quindi il valore precedente. Il

frammento di programma che risolve il problema sopra descritto è il seguente (ove per brevità abbiamo omesso le operazioni di `get` e stampa):

```
find first Docenti ;
   while db-status
   do begin
        find first Corsi within Docenza ;
        while db-status
        do begin
             find first Esami within Corsi-Esami ;
             while db-status
             do begin
                  find owner within Studenti-Esami
                  if db-status
                  then begin
                         find owner within Tesi
                              retaining Docenti currency
                       end
                  find next Esami within Corsi-Esami
                end
             find next Corsi within Docenza
           end
        find next Docenti ;
      end
```

Per proseguire una scansione a partire da un record corrente per un tipo record o un tipo set (ma non per il programma) si può utilizzare l'istruzione

`find current` *NomeRecord*

`find current of` *NomeSet*

La clausola `retaining currency` permette di gestire due indicatori di correnza dello stesso tipo. Essa risulta però insufficiente per la gestione di situazioni ancora più complesse, per esempio in caso di navigazioni che richiedano più di due cicli nidificati su uno stesso tipo record o tipo set. Non scendiamo nel dettaglio, perché con gli schemi visti finora sarebbe difficile trovare un esempio realmente significativo. Comunque, per specificare navigazioni di questo tipo, i DML reticolari prevedono la possibilità di gestire direttamente gli indirizzi dei record, salvandone i valori e utilizzandoli per l'accesso. Il tutto si può realizzare tramite le due istruzioni:

`save db-key into` *Variabile*

`find` *NomeRecord* `db-key is` *Variabile*

la prima delle quali (che nei sistemi reali ha in effetti una sintassi diversa) permette di memorizzare in una variabile un puntatore al record corrente, mentre la seconda permette di accedere al record (del tipo indicato) il cui indirizzo è memorizzato nella variabile specificata.

A.3.3 Le operazioni di aggiornamento

Le operazioni di aggiornamento nei sistemi reticolari sono ancora basate sul principio della navigazione, in quanto tutti gli aggiornamenti possono essere effettuati su un solo record alla volta: il record corrente per il programma. Le operazioni fondamentali sono quelle già viste per il modello relazionale, e cioè inserimento, modifica, eliminazione, più alcune tipiche del modello stesso, relative alla partecipazione di record a set.

L'operazione di inserimento di un nuovo record viene specificata per mezzo dell'istruzione

`store` *NomeRecord*

che inserisce, fra i record relativi al tipo specificato, un nuovo record, con contenuto uguale al contenuto attuale del buffer per il tipo record stesso. Il record diviene record corrente per il programma, per il tipo e per tutti i tipi set di cui è owner. Se il tipo record interessato è member di un set con opzione di inserimento `automatic`, allora il record viene inserito nell'occorrenza corrente di tale set, diventandone record corrente.

L'operazione di modifica del contenuto di un record viene richiesta, in modo analogo, per mezzo dell'istruzione

`modify` *NomeRecord*

che assegna al record corrente del programma, che deve essere del tipo specificato (l'indicazione esplicita ha lo scopo di conferma), il contenuto del buffer del tipo stesso.

L'operazione di eliminazione è leggermente più complessa, in quanto, specificandola, si deve tener conto della partecipazione del record corrente ad occorrenze di set. La sua sintassi è la seguente:

`erase` *NomeRecord*

La cancellazione viene effettuata solo se il record corrente del programma è del tipo specificato nell'istruzione (si tratta sostanzialmente di una forma di controllo o conferma). Inoltre, la cancellazione richiede alcune condizioni riguardo alle opzioni di ritenzione e alla presenza di occorrenze di set di cui il record cancellando sia owner e che contengano occorrenze del member. Distinguiamo i vari casi, sulla base del valore dell'opzione di ritenzione:

- `fixed`: il record viene cancellato, con tutti i membri;
- `mandatory`: il record viene cancellato solo se l'occorrenza del set non ha membri; altrimenti l'operazione viene rifiutata;
- `optional`: il record viene cancellato e i membri vengono mantenuti.

Se le condizioni non sono verificate, non viene effettuato alcun aggiornamento, e la variable `db-status` assume il valore *false*. Altrimenti, il record corrente viene eliminato dalla base di dati, e, ovviamente, rimosso da tutte le eventuali occorrenze di set cui appartenga come member. Dopo la cancellazione, non esiste alcun

record corrente per il programma, per il tipo record coinvolto e per i tipi set cui partecipa. Però, per il tipo record e per i set di cui il tipo record coinvolto è member, gli indicatori di correnza, pur essendo indefiniti, "puntano" alla posizione precedentemente occupata dal record eliminato, da cui è possibile proseguire una scansione sequenziale.

L'eliminazione, dalla base di dati nella figura A.2, del record dello studente Luca Rossi, cioè quello con matricola 587614, può essere effetuata specificando direttamente

```
Studenti.Matricola := 587614
find any Studenti
erase Studenti
```

solo se l'occorrenza del set STUDENTI-ESAMI di cui tale record non ha membri. Altrimenti, poiché tale set ha opzione di ritenzione `mandatory`, è necessario eliminare (in altri casi si potrà rimuovere dal set, come vedremo più avanti, a proposito del comando `disconnect`) i membri di tale occorrenza:

```
Studenti.Matricola := 587614
find any Studenti ;
find first Esami within Studenti-Esami ;
while db-status
do begin
      erase Esami ;
      find next Esami within Studenti-Esami
   end ;
erase Studenti
```

Si noti come sia posssibile proseguire la scansione sequenziale anche dopo aver eliminato i record di tipo ESAMI. Inoltre, questi record possono essere eliminati senza ulteriori verifiche, poichè non sono owner di set.

Passiamo infine alle operazioni che coinvolgono direttamente i set. Esse permettono di specificare la connessione di un record (quello corrente del programma) alla occorrenza corrente di un set, nella posizione specificata dalla clausola `order`, e l'operazione opposta, di rimozione della connessione. Le sintassi sono ancora molto semplici:

`connect` *NomeRecord* `to` *NomeSet*

`disconnect` *NomeRecord* `from` *NomeSet*

`reconnect` *NomeRecord* `within` *NomeSet*

Per quanto riguarda la semantica, l'idea fondamentale è immediata, resta da precisare che le operazioni possono venire effettuate solo se il record corrente non è già membro del set (nel caso della `connect`) e solo se è membro di un'occorrenza del set (nel caso della `disconnect`). Inoltre la `disconnect` fa diventare indefinito il record corrente del tipo set, pur continuando a puntare alla posizione precedentemente occupata dal record rimosso. Non scendiamo in dettaglio riguardo alla `reconnect`, notando solo che, nel caso di set con valore `mandatory` per l'opzione di ritenzione, essa è indispensabile per effettuare modifiche alla connessione, in quanto non può essere simulata per mezzo di una `disconnect` seguita da una `connect`.

A.4 Progettazione logica nel modello reticolare

Nel capitolo 7 abbiamo discusso le varie attività che compongono la progettazione logica, cioè quella fase che porta a generare lo schema logico della base di dati a partire dallo schema concettuale. In effetti, abbiamo anche notato come la progettazione logica possa essere articolata in due sottofasi e come solo la seconda di esse (la "traduzione") dipenda effettivamente dal modello logico di interesse. Per completare la presentazione del modello reticolare e, in modo coordinato, della relativa progettazione, vediamo in questo paragrafo come si debba procedere per tradurre schemi E-R ristrutturati in schemi reticolari. Per certi aspetti, la situazione è ancora più semplice che per il modello relazionale, perché si può far corrispondere tipi record alle entità e tipi set alle associazioni. In effetti, nella traduzione si può sfruttare tale caratteristica, in particolare per tradurre entità in tipi record. Per le associzioni, si deve però notare che un tipo set corrisponde ad una associazione uno a molti fra tipi record diversi, per cui dovrà essere necessario prestare particolare attenzione ai casi più complessi, e cioè alle associazioni molti a molti, a quelle ternarie (o più che ternarie) e a quelle ricorsive. Nei vari casi, i vincoli di cardinalità minima pari a 1 (obbligatorietà) possono essere tradotti per mezzo delle opzioni di inserimento e ritenzione.

Vediamo i vari casi, cominciando dal più semplice.

Una associazione uno a molti può essere tradotta direttamente con un tipo set. Ad esempio, la figura A.6 mostra la traduzione dello schema già mostrato nella figura 7.20:abbiamo due tipi record e un tipo set; in DDL, potremmo utilizzare le seguenti istruzioni:

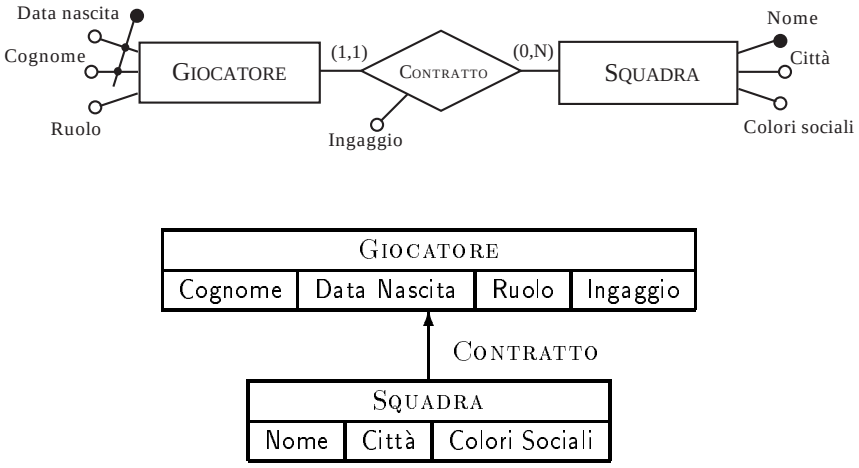

Figura A.6 Una associazione uno a molti e la sua traduzione nel modello reticolare

```
schema name is  ...
  record name is Giocatore
    location mode is calc using Cognome, DataNascita
                                        duplicates not allowed
    Cognome          : string 20
    DataNascita      : date
    Ruolo            : string 10
    Ingaggio         : integer
  end
  record name is Squadra
    location mode is calc using Nome duplicates not allowed
    Citta          : string 20
    ColoriSociali  : string 20
  end
  set name is Contratto
    owner is Squadra
    member is Giocatore mandatory automatic
    order is next
  end
```

dove peraltro le scelte relative a `location mode` e `ordering` potrebbero essere fatte in modo diverso. Notiamo che, non essendo possibile associare direttamente campi ad un tipo set, abbiamo dovuto far "migrare" Ingaggio, attributo dell'associazione, verso il record GIOCATORE, ma questo non è un problema, perché, essendo la relationship uno a molti, per ciascun giocatore esiste un solo ingaggio.

Per quanto riguarda le associazioni uno a uno, si può procedere in modo analogo, valutando, a seconda dei casi, quale possa essere il record più indicato a fungere da owner del set.

È interessante sottolineare che anche le associazioni uno a molti con identificazione esterna possono essere tradotte direttamente nello stesso modo, senza neanche la necessità di esplicitare i campi di identificazione esterna. Mostriamo un esempio nella figura A.7, dove lo schema già mostrato nella figura 7.21 viene tradotto nel modello reticolare (le opzioni dei set sono come nel caso precedente). Le occorrenze del tipo record STUDENTE non hanno una chiave identificante, ma possono essere identificate attraverso il valore del campo Matricola e l'occorrenza di UNIVERSITÀ cui sono collegate attraverso il set ISCRIZIONE.

I tre casi "difficili" sopra accennati si risolvono in effetti tutti nello stesso modo: introducendo un tipo record ausiliario, che realizza l'associazione e che, se l'associazione stessa non ha attributi propri, non ha alcun campo, ma partecipa a due o più tipi set, di solito come member.

Le figure A.8, A.9 e A.10 mostrano le traduzione degli schemi già visti nel capitolo 7 (e precisamente nelle figure 7.17, 7.18 e 7.19) ciascuno relativo ad uno dei tre casi in questione.

Nel caso di associazioni ricorsive uno a molti, è opportuno invertire uno dei due tipoi set. Ad esempio, se la associazione COMPOSIZIONE avesse avuto cardinalità massima pari a 1 per l'entità COMPONENTE (ogni componente partecipa al più ad un prodotto composto), allora la traduzione nel reticolare sarebbe stata quella mostrata in figura A.11, che impone appunto il vincolo in questione.

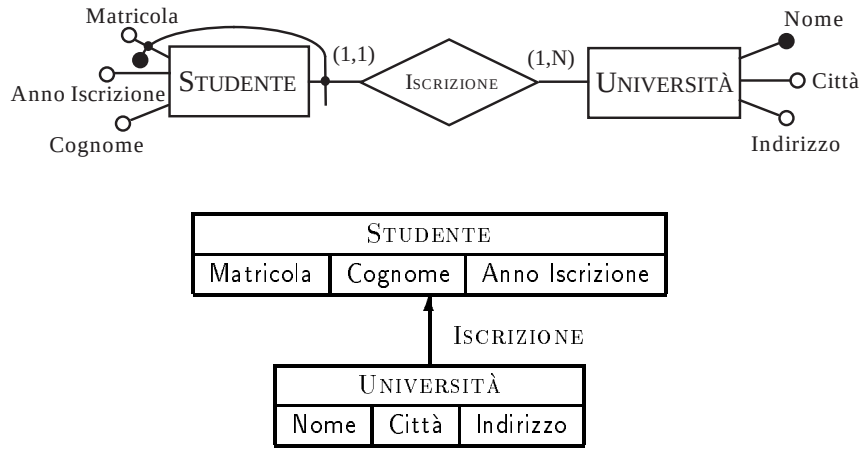

Figura A.7 Uno schema E-R con identificatore esterno e la sua traduzione nel modello reticolare

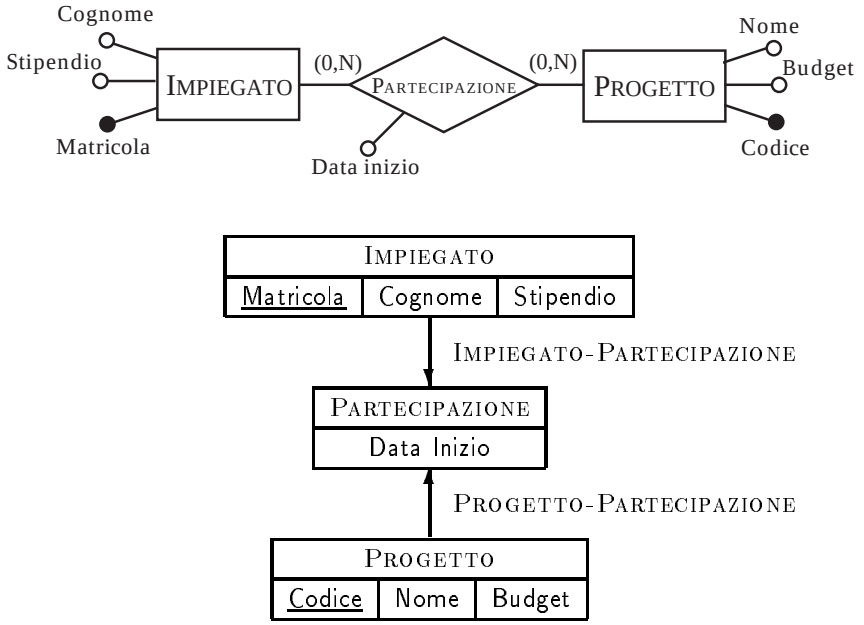

Figura A.8 Traduzione nel modello reticolare di una associazione molti a molti

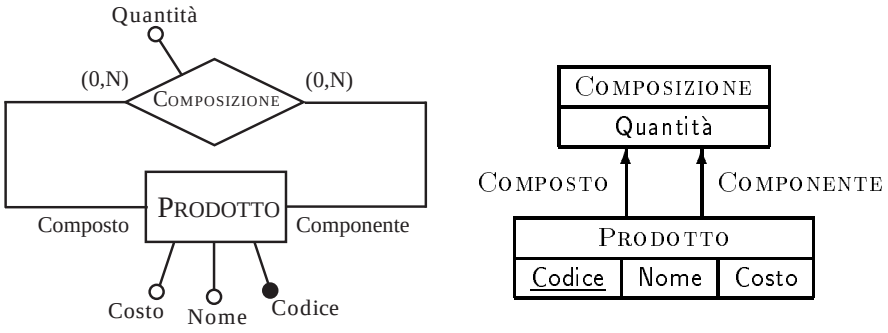

Figura A.9 Traduzione nel modello reticolare di una associazione ricorsiva

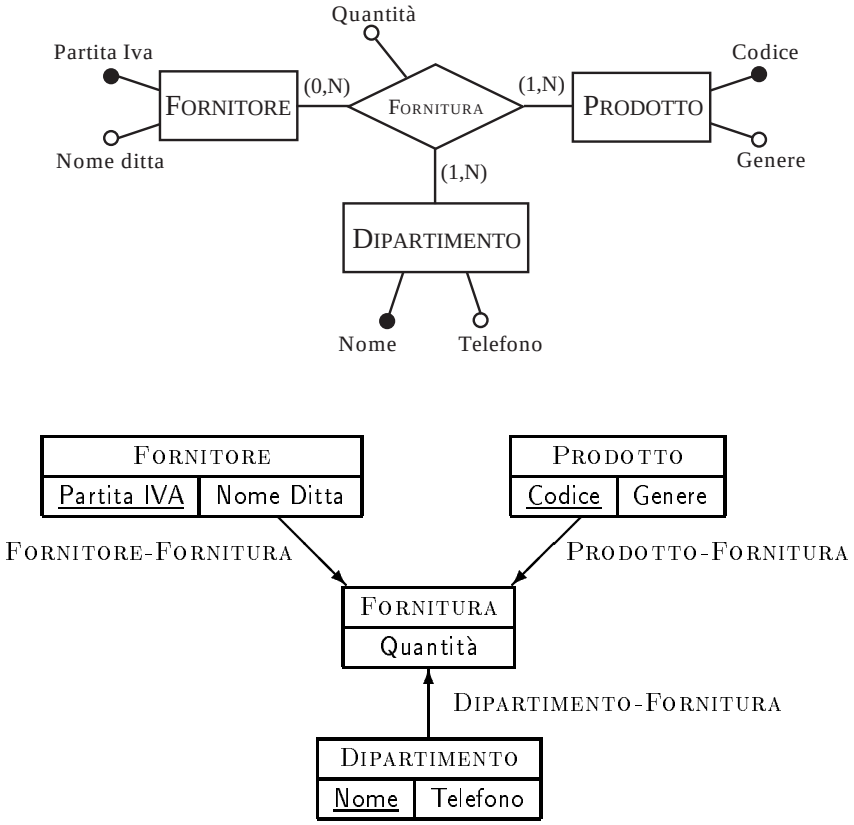

Figura A.10 Traduzione nel modello reticolare di una associazione ternaria

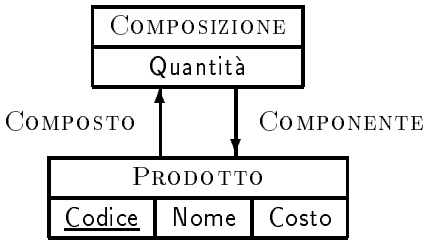

Figura A.11 Traduzione nel modello reticolare di una associazione ricorsiva uno a molti

Concludiamo mostrando nella figura A.12 lo schema reticolare ottenuto traducendo lo schema Entità-Relazione della figura 7.26, che, nel capitolo 7, abbiamo già tradotto nel modello relazionale. Lasciamo per esercizio la specifica delle opzioni di inserimento e di ritenzione, che si ottengono sulla base delle cardinalità delle associazioni.

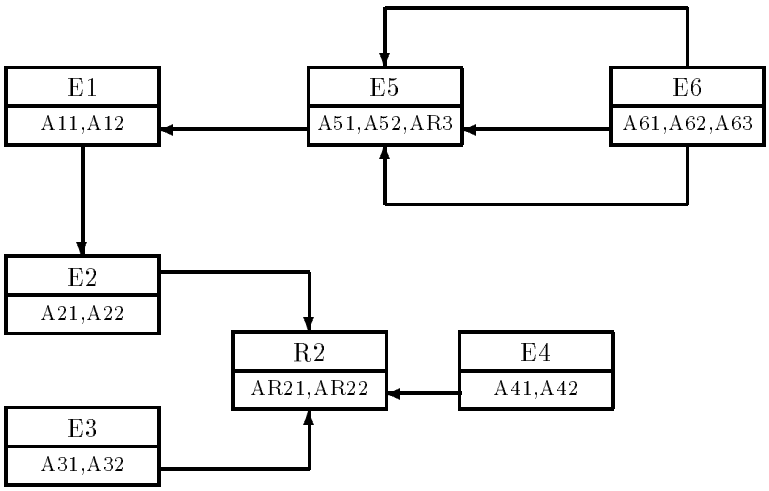

Figura A.12 Schema reticolare corrispondente allo schema E-R della figura 7.26

Note bibliografiche

I concetti relativi al modello reticolare possono essere approfonditi sui testi sulle basi di dati, in particolare quelli di Elmasri e Navathe [30] e Korth e Silberchatz [43]. A titolo di interessante curiosità si segnala l'articolo di Bachman [6],

"The programmer as a navigator," che (scritto da uno degli inventori del modello) descrive molto bene il paradigma di visita di una base di dati reticolare.

Esercizi

Esercizio A.1 Considerare le informazioni per la gestione dei prestiti di una biblioteca personale descritte nell'esercizio 2.1 e definire il corrispondente schema reticolare.

Esercizio A.2 Mostrare come possano essere realizzate nel modello reticolare strutture idonee a rappresentare un albero genealogico (vedi esercizio 2.4).

Esercizio A.3 Con riferimento alla base di dati nella figura A.2, descrivere il risultato prodotto dal seguente frammento di programma:

```
find first Corsi
while db-status
do begin
      get ;
      find first Esami within Corsi-Esami
      if db-status
        then writeln (Corsi.Codice, Corsi.Titolo) ;
      find next Corsi
   end
```

Esercizio A.4 Con riferimento allo schema reticolare nella figura A.3, mostrare i frammenti di programma che trovano:

1. per ciascuno studente, il numero di esami superati e la relativa media;

2. per ciascun professore, la lista degli studenti seguiti come tesisti e, per ciascuno di essi, la media dei voti ottenuti negli esami;

3. l'elenco degli studenti che hanno sostenuto almeno un esame con il proprio relatore.

Inoltre, mostrare i frammenti di programma che realizzano i seguenti aggiornamenti:

1. inserire un esame, ricevendo in input la matricola dello studente e il codice del corso;

2. trasferire al docente Rossi (matricola 8554) i laureandi del docente Neri (matricola 4332).

Esercizio A.5 Considerare lo schema reticolare in figura A.1 e lo schema relazionale ad esso corrispondente:

STUDENTI(Matricola, Cognome, Nome, Data di nascita)
ESAMI(Studente, Voto, Corso)
CORSI(Codice, Titolo, Docente)

Scrivere un frammento di programma che produca lo stesso risultato della seguente interrogazione SQL:

```
select Cognome, Nome, Titolo, Docente
from Studenti, Docenti, Esami
where Corso = Codice and Voto = 29 and Matricola = Studente
```

Esercizio A.6 Tradurre nel modello reticolare lo schema Entità-Relazione della figura 7.34 (esercizio 7.5).