Approfondimenti – Capitolo 2

SQL PER SISTEMI RELAZIONALI AD OGGETTI

I sistemi relazionali ad oggetti sono sistemi basati sul modello dei dati relazionale esteso con le caratteristiche tipiche del modello ad oggetti. Anche il linguaggio per la definizione e l'uso della base di dati è un'estensione dell'SQL, per mantenere il più possibile la compatibilità con i sistemi relazionali tradizionali.

Attualmente non esiste una definizione universalmente accettata del modello dei dati e del linguaggio di un sistema relazionale ad oggetti. Esistono prodotti che adottano soluzioni piuttosto diverse (quali DB2, Oracle e POSTGRES), e una proposta di un nuovo standard ISO, SQL:1999 e SQL:2003 (http://www.iso.org).

Nel seguito presenteremo prima le caratteristiche generali del modello dei dati relazionale ad oggetti e poi, come esempio di SQL ad oggetti, si presenterà la soluzione adottata da Illustra. [1]

2.1 Il modello relazionale ad oggetti

2.1.1 Il modello relazionale non in prima forma normale

L'evoluzione del modello relazionale nel modello relazionale ad oggetti può essere schematizzata come segue.

Come è noto, nel modello relazionale una relazione è un insieme di ennuple con attributi di tipo primitivo (relazioni in prima forma normale).

Una prima proposta di estensione per il modello relazionale (modelli rela-

1. Nel 1992 M. Stonebraker, G. Morganthaler, M. Ubell e P. Hawthorn crearono la società Mirò per sviluppare la versione commerciale del sistema POSTGRES. La società poi cambiò il nome prima in Montage e poi in Illustra. Nel 1996 la società fu acquistata dalla Informix, che a sua volta nel 2001 fu acquistata dalla IBM e poi del DBMS Illustra si sono perse le tracce.

zionali non in prima forma normale) è stata l'eliminazione della restrizione che gli attributi siano di tipo primitivo, ridefinendo il tipo relazione come segue:

1. int, real, bool e string sono *tipi primitivi*;
2. se $T_1, \ldots, T_n$ sono tipi primitivi o tipi insieme e $A_1, \ldots, A_n$ sono etichette distinte, dette *attributi*, allora $(A_1{:}T_1, \ldots, A_n{:}T_n)$ è un *tipo ennupla*;
3. se T è un tipo ennupla, allora {T} è un *tipo insieme*, ed il tipo delle relazioni è un tipo insieme.

Secondo questa definizione, gli elementi di una relazione sono ennuple i cui attributi sono o di tipo elementare oppure di tipo relazione (relazioni *nidificate*, o *annidate*). Un tipo relazione si ottiene quindi combinando *alternativamente* i costruttori di tipo insieme e di tipo ennupla. Del modello relazionale in prima forma normale si mantiene il fatto che le associazioni fra dati di relazioni diverse si modellano con il meccanismo delle chiavi esterne. Del modello ad oggetti si adotta però solo la possibilità di definire relazioni di ennuple con componenti non elementari, ma non la possibilità di condividere ennuple, di definire gerarchie o aspetti procedurali.

Sulle relazioni nidificate è stata poi definita un'algebra con operatori che sono una generalizzazione degli operatori dell'algebra relazionale classica [SS86, JS82, AB84a, AB84b]. Per una generalizzazione delle relazioni nidificate che elimina il vincolo sull'alternanza dei costruttori ennupla e insieme si veda [AHV95].

2.1.2 Il modello relazionale ad oggetti

Un'altra estensione del modello relazionale, proposta indipendentemente dalla precedente, sostituisce la nozione di ennupla con la nozione di oggetto con identità.

Un oggetto è una coppia (*identificatore*, *valore*), dove l'identificatore (OID) è unico e immutabile, mentre il valore è un'ennupla definita come segue:[2]

1. interi, reali, booleani, stringhe e OID sono valori primitivi. Il valore indefinito si indica con *nil*;
2. se $v_1, \ldots, v_n$ sono valori e $A_1, \ldots, A_n$ sono etichette distinte, dette *attributi*, allora $(A_1{:=}\, v_1, \ldots, A_n{:=}\, v_n)$ è un valore *ennupla*;
3. se $v_1, \ldots, v_n$ sono valori dello stesso tipo, allora $\{v_1, \ldots, v_n\}$ è un valore *tipo insieme*.

L'uso di oggetti con identità consente di costruire oggetti o_1 e o_2 che hanno come componente l'identificatore dello stesso oggetto o_3. Poiché l'identificatore di un oggetto è immutabile, modifiche dello stato di o_3 sono automaticamente riflesse in o_1 e o_2.

Uno schema relazionale ad oggetti è una collezione di definizioni di tipi oggetti e di schemi di relazioni R, ciascuna associata ad un tipo oggetto. Un

2. Di solito sono previsti altri costruttori di valori, ma per semplicità non si prenderanno in considerazione

tipo oggetto è definito come segue (la definizione dei metodi e di tipi oggetti per ereditarietà è trattata più avanti):

1. int, real, bool, string sono tipi primitivi;
2. se $T_1, \ldots, T_n$ sono tipi e $A_1, \ldots, A_n$ sono etichette distinte, dette *attributi*, allora $[A_1{:}T_1, \ldots, A_n{:}T_n]$ è un tipo *ennupla*;
3. se T è un tipo, allora {T} è un tipo insieme;
4. se T è un tipo ennupla, allora objecttype Ide := T definisce un tipo oggetto di nome Ide, il cui valore ha tipo T;
5. se T è un tipo oggetto, ref(T) è il tipo degli OID degli oggetti di tipo T.

Le associazioni fra tabelle di oggetti si possono modellare quindi con il meccanismo dell'aggregazione, ovvero inserendo negli oggetti di una tabella l'OID dell'oggetto associato nell'altra, oppure l'insieme di tali OID se l'associazione è multivalore.

In questo sistema di tipi si distingue esplicitamente tra il tipo di un oggetto (ovvero di una coppia <OID, valore>) ed il tipo di un semplice OID. Esiste quindi una funzione deref che va applicata esplicitamente ad un OID per ottenere l'oggetto relativo, e la sua inversa ref che va applicata ad un oggetto per estrarne l'OID.

Un altro approccio possibile è quello di nascondere l'esistenza di un tipo OID. In questo caso, un tipo oggetto in un campo di un tipo ennupla viene interpretato dal sistema come un tipo OID, ed è il sistema che inserisce automaticamente le operazioni deref e ref quando necessario.

Ad esempio, sia P un oggetto con un attributo q che contiene l'OID di un secondo oggetto Q, il quale ha un campo intero i. Nel primo approccio, il valore di i si estrae con l'operazione deref(P.q).i, mentre nel secondo caso basta scrivere P.q.i (supponiamo che l'estrazione di un campo a dal valore di un oggetto o si denoti come $o.a$).

Per far riferimento a questi due approcci per estendere il modello relazionale con oggetti, chiameremo il primo, adottato ad esempio in Illustra (si veda più avanti), *modello relazionale con identità di oggetto*, e il secondo, adottato in UniSQL, *modello relazionale ad oggetti*.

2.1.3 Definizione di tipi oggetto per ereditarietà

Per definire tipi oggetti per ereditarietà si aggiunge la seguente regola a quelle viste in precedenza per la definizione di tipi oggetto:

6. se T è il nome di un tipo oggetto, objecttype T2 := is T and $[B_1{:}T_1, \ldots, B_n{:}T_n]$ definisce un nuovo tipo oggetto per ereditarietà.

Sull'insieme dei tipi è definita, inoltre, una relazione di *sottotipo* con regole analoghe a quelle viste per il Galileo 97.

2.1.4 Inclusione tra relazioni

Come in Galileo 97, anche nel modello relazionale ad oggetti è possibile definire una relazione di inclusione tra relazioni, purché il tipo degli oggetti del-

la sottorelazione sia definito per ereditarietà dal tipo degli oggetti dell'altra relazione.

2.1.5 Definizione dei metodi

Per trattare in modo completo gli oggetti non basta limitarsi a considerare solo gli aspetti strutturali, ma occorre anche un meccanismo per definire metodi.

Un metodo ha tre componenti: il nome, la segnatura e l'implementazione (o corpo). Di solito nelle proposte di modelli relazionali con oggetti si adottano le seguenti alternative:

1. di un metodo si specifica nel tipo oggetto solo il nome e la segnatura. L'implementazione si specifica a parte, anche in un linguaggio diverso da quello usato per definire gli oggetti, se esso non consente di trattare gli aspetti procedurali;
2. il tipo oggetto non contiene nessuna informazione sui metodi, per cui il nome, la segnatura e l'implementazione di un metodo vengono definiti separatamente come una funzione con argomento il tipo dell'oggetto a cui si riferisce. Per consentire la ridefinizione dei metodi nei tipi definiti per ereditarietà, una funzione può essere ridefinita con lo stesso nome specializzando il tipo dell'oggetto usato come argomento (*funzione sovraccarica*).

2.1.6 L'uguaglianza tra oggetti

Due oggetti o_1 e o_2 sono uguali se hanno lo stesso OID, ovvero se sono lo stesso oggetto. Nei linguaggi ad oggetti di solito è previsto anche un altro operatore di uguaglianza, detto *uguaglianza superficiale*, per controllare se due oggetti hanno lo stesso valore ($o_1 == o_2$).

L'uguaglianza di due oggetti o_1 e o_2 si controlla scrivendo $o_1 = o_2$ nel modello relazionale ad oggetti, mentre nel modello relazionale con identità di oggetto (o almeno nel sistema Illustra, che lo incarna), è necessario confrontare esplicitamente gli OID, scrivendo ref(o_1) = ref(o_2).

2.1.7 SQL per basi di dati ad oggetti

Per interrogare basi di dati ad oggetti sono state proposte diverse estensioni dell'SQL, ma ogni sistema commerciale ha una sua particolare versione. La proposta più interessante, che mostra bene come andrebbe ripensato l'SQL relazionale per adeguarlo alle basi di dati ad oggetti, è quella del sistema O_2, sul quale si basa lo standard OQL di ODMG, anche se O_2 era un sistema oggetti ma non un sistema relazionale ad oggetti.

Un SQL ad oggetti prevede in genere le seguenti principali estensioni dell'SQL relazionale:

1. ogni costrutto che restituisce un valore di un certo tipo può essere utilizzato al posto di qualsiasi valore dello stesso tipo (*principio dell'ortogonalità*). Grazie a questo principio, ad esempio, nella parte FROM le variabili di correlazione possono essere legate non solo a ennuple di tabelle dello schema

(un caso particolare di insieme di ennuple), come accade nei sistemi relazionali, ma anche ad elementi di un qualunque insieme di ennuple ottenuto con un'altra espressione SQL oppure come risultato dell'applicazione di una funzione o di un metodo;
2. in una clausola FROM Expr_1 $\mathsf{x}_1,\ldots,$ Expr_n x_n l'espressione Expr_n può dipendere dalle variabili $\mathsf{x}_1,\ldots,\mathsf{x}_{n-1}$; questo significa che la clausola FROM può effettuare non solo un prodotto di tabelle, ma anche un prodotto dipendente, come definito ed esemplificato nella Sezione 2.3.3. Il prodotto dipendente, per la sua somiglianza con la giunzione, è chiamato, in questo contesto, *giunzione funzionale* o *implicita*;
3. nelle condizioni si possono usare i quantificatori esistenziale (predicato vero se esiste almeno un elemento di un insieme che soddisfa una certa condizione, come some del Galileo) e universale (predicato vero se ogni elemento di un insieme soddisfa una certa condizione, come each del Galileo);
4. nella selezione dei campi si possono usare cammini per selezionare componenti di campi che sono a loro volta strutturati.

Come nel caso dei sistemi relazionali, per sviluppare applicazioni si usano linguaggi che ospitano gli operatori del modello dei dati oppure interfacce API. Mancano per ora esempi di linguaggi integrati, come il PL/SQL dei sistemi relazionali.

Per dare un esempio di linguaggio per definire e usare basi di dati relazionali ad oggetti si mostra il caso del sistema Illustra che, usando la terminologia introdotta precedentemente, usa un modello relazionale con identità di oggetto.

2.2 Definizione di basi di dati

Le principali caratteristiche di un linguaggio per la definizione di basi di dati relazionali a oggetti sono le seguenti:

1. si possono definire *nuovi* tipi con il comando CREATE TYPE;
2. fra i costruttori di tipo vi è quello per definire oggetti con le seguenti caratteristiche:
   (a) un oggetto ha un'identità unica e immutabile e le componenti sono valori di qualsiasi tipo;
   (b) un tipo oggetto può essere definito per *ereditarietà*, semplice o multipla;
   (c) i metodi di un oggetto sono funzioni definite separatamente, con primo parametro il tipo dell'oggetto e corpo costituito da uno o più comandi SQL, o definito con un linguaggio tradizionale come il C;
      – una funzione può essere ridefinita specializzando il parametro di tipo oggetto e vale la proprietà del *late binding*;
      – una funzione può essere applicata a parametri attuali che sono di un sottotipo di quello formale (*ereditarietà del contesto*);
   (d) un oggetto non incapsula lo stato e quindi tutti i campi sono visibili all'esterno;

3. si possono definire collezioni modificabili di oggetti, dette *tabelle*, con il comando CREATE TABLE. Tabelle diverse possono avere elementi con lo stesso tipo. Nuovi elementi si inseriscono nelle tabelle con il comando INSERT (collezioni con inserzione esplicita);
4. si possono definire tabelle come sottoinsiemi di altre (*gerarchia di tabelle*), con la gestione automatica della relazione di sottoinsieme.

Vediamo i principali costruttori per modellare tipi di dati e tabelle, usando la soluzione di Illustra.

Tipi primitivi Oltre a quelli usuali, vi sono dei tipi temporali (abstime per istanti di tempo e reltime per intervalli) e un tipo stringa di caratteri non interpretati di lunghezza qualunque, da usare come tipo base per rappresentare dati di grande dimensione (text, large-text e large-object).

Inoltre, l'utente può definire nuovi tipi elementari (*a*) specificando la lunghezza in caratteri dei valori, (*b*) dandone la rappresentazione in C, e (*c*) definendo nello stesso linguaggio le funzioni input e output per le conversioni tra la rappresentazione esterna e interna.[3]

Tipi strutturati Fra i costruttori di tipo sono previsti i seguenti:

1. Tipi composti: sono oggetti con componenti di tipo qualsiasi. Sono ammesse definizioni di tipi composti ricorsivi solo fra elementi di tabelle usando tipi REF, mostrati più avanti. Ad esempio:

```
CREATE TYPE UnTelefono
    (Prefisso   CHAR(4),
     Numero     CHAR(8));

CREATE TYPE Studente
    (Nome       CHAR(20),
     Matricola  CHAR(8),
     Sesso      CHAR(1),
     Telefono   UnTelefono);
```

2. Array: sono collezioni ordinate di valori di numero prefissato (dichiarato con la notazione Tipo[Costante]) o variabile (ARRAYOF(Tipo)). Il tipo degli elementi può essere primitivo, REF o ARRAY;
3. SETOF(T): sono multinsiemi non ordinati di valori di tipo T;
4. REF(T): un valore di questo tipo è l'OID di un oggetto di tipo T. Questo tipo è usato in particolare per definire le associazioni fra tabelle per aggregazione: un oggetto ha un campo che contiene un riferimento ad un altro oggetto memorizzato nella tabella in associazione. Associazioni multivalore sono rappresentate con campi di tipo SETOF(REF(T)).

   Sui valori di tipo REF(T) è disponibile la funzione deref per ottenere l'oggetto riferito. Esiste anche la funzione inversa ref per passare da un oggetto

3. Questo meccanismo è usato anche per fornire nuove funzionalità al sistema, come la gestione di dati geografici, documentali, multimediali ecc. Queste estensioni sono chiamate *data blades*.

O al suo (OID); essendo l'OID memorizzato in un campo particolare dell'oggetto con nome oid, questo valore si può ottenere anche con la notazione O.oid.

Tabelle Sono un particolare valore di tipo SETOF con elementi di tipo oggetto. Una tabella è definita con il comando CREATE TABLE. Il tipo degli elementi può essere definito a parte, insieme alla tabella oppure dandone solo la struttura con il comando CREATE TABLE. Sui campi del tipo oggetto si possono definire vincoli d'integrità. Ad esempio:

```
CREATE TABLE Studenti OF TYPE Studente.

CREATE TABLE Studenti OF NEW TYPE Studente
    (Nome        CHAR(20),
     Matricola   CHAR(8),
     Sesso       CHAR(1) CHECK(Sesso IN ('F', 'M')));

CREATE TABLE Studenti
    (Nome        CHAR(20),
     Matricola   CHAR(8),
     Sesso       CHAR(1) CHECK(Sesso IN ('F', 'M')));
```

Tabelle in gerarchia Sono tabelle sottoinsieme di altre. Ad esempio:

```
CREATE TABLE Laureandi
    (TitoloTesi   CHAR(40))
    UNDER Studenti.
```

Le seguenti definizioni riguardano lo schema della base di dati di Figura 2.1.

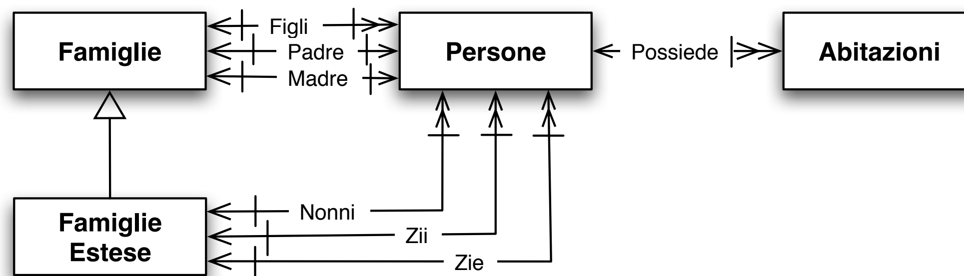

Figura 2.1. Rappresentazione grafica di una base di dati

```
CREATE TYPE Indirizzo
    (Via     CHAR(20),
     Citta   CHAR(10)).

CREATE TABLE Abitazioni
    (Categoria      CHAR(10),
     Proprietario   REF(Persona),
     Indirizzo      Indirizzo).

CREATE TABLE Persone OF NEW TYPE Persona
```

```
    (Nome          CHAR(20),
     AnnoNascita   INTEGER,
     Sesso         CHAR(1) CHECK(Sesso in ('F', 'M'))).

CREATE TABLE Famiglie OF NEW TYPE Famiglia
    (Padre           REF(Persona),
     Madre           REF(Persona),
     Figli           SETOF(REF(Persona)),
     EntrateMensili  REAL[12]).

CREATE TABLE FamiglieEstese OF NEW TYPE FamigliaEstesa
    (Nonni   REF(Persona)[4],
     Zii     SETOF (REF(Persona)),
     Zie     SETOF (REF(Persona)))
    UNDER Famiglie.
```

Questo schema mostra come, nel modello con identità, sia possibile distinguere tra un campo che contiene un riferimento ad un oggetto (Proprietario in Abitazioni) ed un campo che contiene direttamente l'oggetto (Indirizzo in Abitazioni). Questa maggior potenza espressiva si paga però con una maggior complessità nello scrivere le interrogazioni.

Uno schema può contenere definizioni di trigger (che non verranno presi in considerazione in questa breve presentazione di Illustra) e funzioni, che possono essere definite in SQL oppure esternamente nel linguaggio C. Le funzioni con corpo un comando SQL sono trattate come macro-definizioni: il loro corpo viene sostituito nel comando dove si usano, prima di iniziare la fase di ottimizzazione del comando SQL, come accade nel caso delle viste logiche. Ad esempio:

```
CREATE FUNCTION EntrateDiUnMese (INTEGER, REF(Famiglia)) RETURNS REAL
    AS SELECT UNIQUE EntrateMensili[$1]
       FROM    Famiglie f
       WHERE   ref(f) = $2.

CREATE FUNCTION tasse (REF(Famiglia)) RETURNS REAL
    AS EXTERNAL NAME /usr/local/tasse/progTasse
    LANGUAGE C.
```

Nel primo caso si definisce una funzione che ritorna uno dei dodici valori della tabella EntrateMensili di una famiglia con un certo OID. $\$i$ è l'i-esimo parametro della funzione, di cui viene specificato il tipo nell'intestazione. La specifica SELECT UNIQUE si usa quando il comando SELECT ritorna un solo valore.

Nel secondo caso si dichiara che la funzione tasse, che ritorna le tasse della famiglia in base alle entrate ed agli altri dati relativi alla famiglia, è stata definita nel linguaggio C, e il suo codice è noto al sistema operativo con il nome esterno /usr/local/tasse/progTasseFamiglia.

La funzione tasse può essere anche applicata ad un valore con il sottotipo FamigliaEstesa e può essere ridefinita come segue specializzando il parametro con un sottotipo per sfruttare il *late binding*:

```
CREATE FUNCTION tasse (REF(FamigliaEstesa)) RETURNS REAL WITH(LATE)
    AS EXTERNAL NAME /usr/local/tasse/progTasseFamigliaEstesa
    LANGUAGE C.
```

Quando si applica la funzione ad un valore, tasse(O), viene eseguita la prima funzione se O ha tipo Famiglia e la seconda se ha tipo FamigliaEstesa.

Le funzioni con parametro un tipo oggetto sono il meccanismo usato da Illustra per definire i metodi del tipo oggetto.

2.3 L'interrogazione di basi di dati

Vediamo i principali costrutti usati da Illustra.

2.3.1 Restrizioni

La clausola FROM Rel r lega la variabile r agli oggetti della relazione Rel, non ai loro OID, per cui i campi di r si possono accedere senza bisogno di usare l'operatore deref.

Ad esempio, per visualizzare tutti gli attributi delle persone nate nel 1980, basta scrivere la seguente interrogazione:

```
SELECT   p
FROM     Persone p
WHERE    p.AnnoNascita = 1980
```

Per trovare invece solo il nome di queste persone si scrive:

```
SELECT   p.Nome
FROM     Persone p
WHERE    p.AnnoNascita = 1980
```

Se una tabella ha una sotto-tabella, nella parte FROM con la funzione only si possono considerare solo gli elementi della tabella che non sono anche elementi della sotto-tabella:

```
SELECT   COUNT(*)
FROM     only(Famiglie)
```

2.3.2 Espressioni di cammino

Le espressioni di cammino $x.y$ (*Path expressions*) si usano per selezionare le componenti di un oggetto a qualunque livello. Ad esempio, per trovare la via delle abitazioni di Pisa di categoria A3 si pone:

```
SELECT   a.Indirizzo.Via
FROM     Abitazioni a
WHERE    a.Indirizzo.Citta = 'PI' AND a.Categoria = 'A3'
```

Se un componente di un oggetto è un riferimento ad un altro oggetto, come Proprietario di un'abitazione, è richiesto l'uso dell'operatore deref.

Ad esempio, per trovare la via delle abitazioni di Mario Rossi, si pone:

```
SELECT   a.Indirizzo.Via
FROM     Abitazioni a
WHERE    deref(a.Proprietario).Nome = 'Mario Rossi'
```

2.3.3 Tipi di giunzione

Giunzioni basate sull'identità degli oggetti Sono giunzioni la cui condizione confronta l'uguaglianza degli oggetti per identità. Questo tipo di uguaglianza è ottenuta confrontando gli OID degli oggetti, per cui, mentre due OID si possono confrontare direttamente scrivendo oid1 = oid2, per confrontare due oggetti è necessario scrivere REF(obj1) = REF(obj2).

Ad esempio, per trovare la via delle coppie di abitazioni diverse con lo stesso proprietario, si pone:

```
SELECT  a.Indirizzo.Via, b.Indirizzo.Via
FROM    Abitazioni a, Abitazioni b
WHERE   a.Proprietario = b.Proprietario AND NOT(REF(a) = REF(b))
```

Si confronti l'interrogazione con la seguente che usa una giunzione basata su valori per trovare la via delle coppie di abitazioni di proprietari con lo stesso nome:

```
SELECT  a.Indirizzo.Via, b.Indirizzo.Via
FROM    Abitazioni a, Abitazioni b
WHERE   deref(a.Proprietario).Nome = deref(b.Proprietario).Nome
        AND NOT(REF(a) = REF(b))
```

Supponendo che esista una funzione disgiunti, che, applicata a due insiemi di OID di persone, ritorni vero se i due insiemi non hanno elementi in comune, con la seguente interrogazione si trovano i nomi di padri di famiglie estese che hanno uno zio in comune:

```
SELECT  deref(f.Padre).Nome
FROM    FamiglieEstese f, FamiglieEstese g
WHERE   NOT disgiunti(f.Zii, g.Zii) AND NOT(REF(f) = REF(g))
```

Giunzioni con prodotti dipendenti Il prodotto dipendente consente di sfruttare le associazioni multivalore rappresentate per aggregazione per effettuare la giunzione degli elementi di una relazione con tutti quelli associati nell'altra. Se gli elementi di una relazione R hanno un campo di tipo insieme A che modella l'associazione con una relazione S, il prodotto dipendente FROM R r, r.A s concatena ogni elemento di R con ciascun valore del suo campo A, effettuando quindi la giunzione tra R ed S.

Usando questo approccio, per trovare i due nomi di ogni coppia padre-figlio, si scriverebbe:

```
SELECT  deref(fam.Padre).Nome, deref(figlio).Nome
FROM    Famiglie fam, fam.Figli figlio
```

In questo caso la giunzione verrebbe fatta fra un elemento della tabella Famiglie e tutti i corrispondenti figli (fam.Figli), che vengono legati al nome figlio, e utilizzati per fare la proiezione. Questo tipo di giunzione non è possibile in Illustra. L'esempio si deve quindi riscrivere come segue:

```
SELECT  deref(fam.Padre).Nome, deref(figlio).Nome
FROM    Famiglie fam, Persone figlio
WHERE   REF(figlio) in fam.Figli
```

2.3.4 Ortogonalità

Le operazioni su insiemi si possono comporre a piacere ed applicare indifferentemente a tabelle, insiemi e array. Un'espressione che ritorna insiemi può essere utilizzata ovunque possa apparire un insieme; ad esempio, un'operazione SELECT si può usare nella parte FROM di un'altra, o in qualunque posizione della clausola WHERE.

Ad esempio, per trovare i nomi dei padri di famiglie con più di 6 figli:

```
SELECT   deref(f.Padre).Nome
FROM     Famiglie f
WHERE    (SELECT COUNT(*) FROM f.Figli) > 6
```

Per trovare il nome del padre e il numero delle abitazioni di categoria A3 possedute da padri di famiglia, si può scrivere:

```
SELECT     deref(f.Padre).Nome AS NomePadre, COUNT(*)
FROM       Famiglie f, (SELECT *
                        FROM    Abitazioni ab
                        WHERE   ab.Categoria = 'A3') a
WHERE      f.Padre = a.Proprietario
GROUP BY   f.Padre
```

Ai valori di tipo array si possono applicare sia gli operatori su insiemi che l'indicizzazione per estrarre un campo specifico, come nel seguente esempio, che ritorna lo stipendio di gennaio per le famiglie che hanno superato i cinque milioni di reddito in più di due mesi:

```
SELECT f.EntrateMensili[1] AS StipendioGennaio
FROM   Famiglie f
WHERE  (SELECT COUNT(*)
        FROM   f.EntrateMensili em
        WHERE  em > 5000000) > 2
```

2.3.5 Funzioni

Le funzioni si possono usare nelle parti SELECT, FROM e WHERE usando la normale notazione funzionale. Ad esempio, per trovare il cognome dei padri di famiglia e le entrate del mese di gennaio (con valori ordinati per entrate), delle famiglie che hanno le entrate di gennaio maggiori di 700.000, si scrive:

```
SELECT     deref(f.Padre).Nome, EntrateDiUnMese(1, REF(f))
FROM       Famiglie f
WHERE      EntrateDiUnMese(1, REF(f)) > 700000
ORDER BY   EntrateDiUnMese(1, REF(f))
```

2.4 Conclusioni

L'esigenza di trattare oggetti complessi in molte applicazioni ha stimolato lo studio di estensioni del modello relazionale con un meccanismo ad oggetti e la definizione di un nuovo standard di SQL ad oggetti, inizialmente SQL:1999 e

poi SQL:2003. Lo standard non è adottato dai prodotti commerciali disponibili, tipo DB2, POSTGRES e Oracle, e quindi esistono alcune differenze fra di loro sui meccanismi del modello dei dati e sulla sintassi dell'SQL adottata.

Esercizi

1. Si riscrivano gli esempi usati per presentare le caratteristiche di Illustra (schema e interrogazioni) usando il Galileo 97.
2. Si consideri la seguente base di definita in Galileo 97:

```
let rec
Impiegati class
   Impiegato <->
      [Codice        :string;
       Cognome       :string;
       Residenza     :[Via :string; CAP :string; Citta :string];
       FigliACarico  :seq [Nome :string; AnnoNascita :int];
       Stipendio     :int;
       LavoraCon     :Azienda;
       Ufficio       :Ufficio
      ] key (Codice)
and
Direttori subset of Impiegati class
   Direttore <-> is Impiegato and
      [ViceDirettore :Impiegato
      ]
and
Aziende class
   Azienda <->
      [Nome           :string;
       Direttore      :Direttore;
       SedeLegale     :[Via :string; CAP :string; Citta :string];
       VolumeVendite  :int
      ] key (Nome)
and
Uffici class
Ufficio <->
      [Piano          :int;
       NumeroStanza   :int
      ] key (Piano, NumeroStanza);
```

Si definisca lo schema in Illustra e si scriva in Galileo 97 e in Illustra un'espressione select per fare le seguenti interrogazioni:

(a) trovare il cognome dei direttori delle aziende con un volume di vendite maggiore di 1 miliardo;
(b) trovare il cognome e le città di residenza degli impiegati dell'azienda con nome Mega;
(c) trovare il cognome degli impiegati che hanno un figlio nato nel 1980;
(d) trovare il cognome degli impiegati che risiedono nella stessa città della sede legale dell'azienda da cui dipendono;
(e) trovare il cognome degli impiegati che risiedono nella stessa città dove risiede il proprio direttore;

(f) trovare il cognome dei direttori che condividono l'ufficio con il proprio vice direttore;
(g) trovare il cognome degli impiegati con stipendio più alto del proprio direttore;
(h) trovare il cognome degli impiegati diretti dal signor Caio;
(i) trovare il cognome del direttore dei dipartimenti con tutti i dipendenti, diversi dal direttore, senza figli;
(j) trovare il nome delle aziende e il numero dei loro dipendenti.

Note bibliografiche

Una presentazione formale dei modelli relazionali nidificato e ad oggetti si trova in [AHV95]. Una raccolta di lavori sul modello relazionale ad oggetti si trova in [ZM90]. Per maggiori dettagli su Illustra si veda [SM96], mentre per una presentazione di UniSQL si veda [Kim95].

BIBLIOGRAFIA

[AB84a] S. Abiteboul and N. Bidoit. An algebra for non normalized relations. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, 1984.

[AB84b] S. Abiteboul and N. Bidoit. Non first normal form relations to represent hierarchically organized data. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 191–200, 1984.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Database Foundations*. Addison-Wesley, Reading, Massachusetts, 1995.

[JS82] G. Jaeschke and H. Schek. Remarks on the algebra on non first normal form relations. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 124–138, 1982.

[Kim95] W. Kim. *Modern Database Systems. The Object Model, Interoperability, and Beyond*. Addison-Wesley, Reading, Massachusetts, 1995.

[SM96] M. Stonebraker and D. Moore. *Object-Relational DBMSs: The Next Great Wave*. Morgan Kaufmann Publishers, San Mateo, California, 1996.

[SS86] H-J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):140–173, 1986.

[ZM90] S.B. Zdonik and D. Maier. Fundamentals of object-oriented databases. In S.B. Zdonik and D. Maier, editors, *Readings in Object-Oriented Database Systems*, pages 1–32. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.