Capitolo 1

IL LINGUAGGIO SQL

Un *sistema di gestione di basi di dati relazionali* è un insieme di programmi che consentono di creare una base di dati relazionale, di interrogarla per ottenere informazioni, di aggiornarla nel corso del tempo e di gestire i file in cui sono fisicamente registrati i dati. Esistono diversi sistemi di gestione di basi di dati relazionali (un esempio è Oracle) ma la maggior parte fanno uso del linguaggio *SQL* (`Structured Query Language`). Un insieme di comandi dell'*SQL* (che prende il nome di `Data Definition Language`) consente di creare una base di dati relazionale; un altro insieme di comandi dell'*SQL* (che prende il nome di `Data Manipulation Language`) consente di interrogare una base di dati relazionale e aggiornare il contenuto.

1.1 *Definizione dei Dati*

Va subito detto che le variabili relazionali sono semplicemente chiamate *table* e, d'ora in poi, useremo la parola *tabella* per indicare o il contenuto di una particolare *table* per un certo stato della base di dati oppure per la risposta ad una domanda per un certo stato della base di dati (cioè ad un dato istante). Infine gli attributi che formano lo schema di una tabella sono dette le *colonne* e le ennuple presenti in una tabella sono dette le *righe*.

Supponiamo di voler creare una base di dati relazionale di nome `Cinema` formata da tre *table* (variabili relazionali) di nome `Film`, `Attore` e `Cast`. La base di dati `Cinema` viene creata con il comando

```
CREATE DATABASE Cinema;
```

e le tre *table* vengono definite con i comandi

```
CREATE TABLE Film (...);
CREATE TABLE Attore (...);
CREATE TABLE Cast (...);
```

Nella definizione di una *table*, oltre al suo nome vengono specificati sia i nomi delle colonne (cioè degli attributi che formano il suo schema) sia i vincoli semantici.

1.1.1 *Tipi elementari*

I domini degli attributi che ne formano lo schema di una *table* sono specificati per mezzo di una gamma di tipi di dati tra i quali troviamo

VARCHAR(*n*)	stringa di caratteri di lunghezza minore o uguale ad *n*, $n \leq 255$
CHAR(*n*)	stringa di caratteri di lunghezza uguale ad *n*, $n \leq 255$
INT	numeri interi, con segno o senza, compresi nell'intervallo [–2147483548, +2147483547]
SMALLINT	numeri interi, con segno o senza, compresi nell'intervallo [–32578, +32577]
DECIMAL(*s*, *d*)	numeri decimali, con segno o senza, con *s* cifre significative, di cui *d* decimali
NUMERIC(*s*, *d*)	v. DECIMAL(*s*, *d*)
YEAR(4)	anni nel formato a quattro cifre, compresi nell'intervallo [1901, 2155]
ENUM('*a*', '*b*', ...)	stringhe appartenenti alla lista ('*a*', '*b*', ...)

Così per le tre *table* della base di dati CINEMA possiamo avere

```
CREATE TABLE Film
(
titolo VARCHAR(20),
anno YEAR(4),
regista VARCHAR(30),
durata SMALLINT,
...
);
```

```
CREATE TABLE Attore
(nome VARCHAR(30) ...,
nascita YEAR(4) ...);
```

```
CREATE TABLE Cast
(film VARCHAR(20) ...,
```

```
anno YEAR(4) ...,
attore VARCHAR(30) ...,
ruolo VARCHAR(10),
...
);
```

1.1.2 *Vincoli*

I vincoli più semplici sono di tipo intrarelazionale e fra questi menzioniamo i vincoli (predefiniti) NOT NULL, UNIQUE e PRIMARY KEY.

— Il vincolo NOT NULL non ammette il *valore nullo* di uno specificato attributo.

Ad esempio nella definizione della *table* Cast possiamo avere

```
CREATE TABLE Cast
(film VARCHAR(20) NOT NULL,
anno YEAR(4) NOT NULL,
attore VARCHAR(30) NOT NULL ...,
ruolo VARCHAR(10),
...
);
```

che sta a significare dire che, nelle tabelle di nome Cast che troveremo negli stati della base di dati Cinema, i valori degli attributi film, di anno e di attore che troveremo nelle righe della tabella saranno sempre ben definiti (cioè elementi dei rispettivi domini), mentre i valori dell'attributo ruolo possono essere indefiniti, cioè possono essere assenti, nel qual caso nella tabella compare la dicitura NULL.

— Il vincolo UNIQUE serve a definire sovrachiavi, cioè identificatori delle righe. È implicito che a tutti gli attributi presenti nella sovrachiave è proibito il valore nullo. Se la sovrachiave è formata da un unico attributo, allora la specifica è accodata alla definizione del tipo di dato dell'attributo:

```
matricola CHAR(6) UNIQUE
```

Se invece la sovrachiave è formata da due o più attributi, allora la specifica segue le definizioni degli attributi:

```
nome VARCHAR(20) NOT NULL,
```

```
cognome VARCHAR(20) NOT NULL,
UNIQUE (nome, cognome)
```

— Il vincolo PRIMARY KEY designa una (sovra)chiave come chiave primaria. È implicito che a tutti gli attributi presenti nella chiave primaria è proibito il valore nullo. Come per UNIQUE si hanno le due alternative:

```
matricola CHAR(6) PRIMARY KEY
```

e

```
PRIMARY KEY (nome, cognome)
```

Nella definizione delle tre *table* della base di dati CINEMA possiamo avere

```
CREATE TABLE Film
(
titolo VARCHAR(20),
anno YEAR(4),
regista VARCHAR(30),
durata SMALLINT,
PRIMARY KEY (titolo, anno)
);
```

```
CREATE TABLE Attore
(nome VARCHAR(30) PRIMARY KEY,
nascita YEAR(4) NOT NULL);
```

```
CREATE TABLE Cast
(film VARCHAR(20) NOT NULL,
anno YEAR(4) NOT NULL,
attore VARCHAR(30) ...,
ruolo VARCHAR(10),
PRIMARY KEY (film, anno, attore),
...
);
```

Vediamo ora un tipo di vincolo interrelazionale molto comune. Si tratta dei vincoli d'integrità referenziale che si esprimono utilizzando la clausola FOREIGN KEY. Un

tale vincolo subordina i valori di un insieme di attributi *X* nello schema di una *table* (*table figlia*) ai valori assunti da un insieme di attributi *Y* di un'altra *table* (*table madre*) nel senso che, in ogni stato della base di dati, ogni valore *nonnullo* di *X* presente nella tabella figlia deve essere presente come valore di *Y* nella tabella madre. Spesso, *Y* è la chiave primaria della tabella madre, la qual cosa spiega l'espressione FOREIGN KEY; in ogni caso, ne deve essere una sovrachiave. Se *X* e *Y* sono entrambi formati da un solo attributo, allora per esprimere il vincolo referenziale basta inserire un "rimando" (REFERENCES). Ad esempio, è ragionevole supporre che la *table* Cast sia così definita

```
CREATE TABLE Cast
(
film VARCHAR(20) NOT NULL,
anno YEAR(4) NOT NULL,
attore VARCHAR(30) NOT NULL REFERENCES Attore (nome),
ruolo VARCHAR(10),
PRIMARY KEY (film, anno, attore),
FOREIGN KEY (film, anno) REFERENCES Film (titolo, anno)
);
```

Un altro esempio è dato dalla base di dati STUART formata dalle due *table* Dinastia e Regno così definite:

```
CREATE TABLE Dinastia
(
nome VARCHAR(20) PRIMARY KEY,
sesso ENUM('F', 'M') NOT NULL,
nascita NUMERIC(4) NOT NULL,
morte NUMERIC(4) NOT NULL
)

CREATE TABLE Regno
(
sovrano VARCHAR(20) PRIMARY KEY
                REFERENCES Dinastia (nome),
inizio NUMERIC(4) NOT NULL,
fine NUMERIC(4) NOT NULL,
UNIQUE (inizio, fine)
)
```

Qui Dinastia e Regno sono rispettivamente la *table* madre e la *table* figlia e l'attributo sovrano nello schema della *table* Regno rimanda all'attributo nome nello schema della *table* Dinastia.

Infine, l'*SQL* offre la possibilità di definire vincoli più complessi di quelli predefiniti visti finora (v. Capitolo 5).

1.1.3 *Descrizione delle table*

Tutti i sistemi di gestione delle basi di dati relazionali hanno una descrizione delle *table* presenti in una base di dati (*dizionario dei dati*) in forma tabellare. Dunque, abbiamo due tipi di *table*: le *table* destinate a contenere i dati e le *table* che ne descrivono la forma (*metadati*). Queste ultime formano il *catalogo* della base di dati. Così per una base di dati contenente le *table* Impiegato e Dipartimento, il catalogo della base di dati contiene una *table* Colonne con schema

nome
tabella
posizione
default
nullo

il cui contenuto è riportato qui di seguito

nome	tabella	posizione	default	nullo
matricola	Impiegato	1	NULL	No
cognome	Impiegato	2	NULL	No
nome	Impiegato	3	NULL	No
ufficio	Impiegato	4	NULL	Sì
stipendio	Impiegato	5	0	Sì
nome	Dipartimento	1	NULL	No
sede	Dipartimento	2	NULL	Sì
nome	Colonne	1	NULL	No
tabella	Colonne	2	NULL	No
posizione	Colonne	3	NULL	No
default	Colonne	4	NULL	Sì
nullo	Colonne	1	Sì	No

1.2 *Interrogazione*

Le domande sono espresse in forma dichiarativa specificando l'informazione d'interesse (cioè la tabella richiesta). In tal senso l'*SQL* si ispira ai principi del Calcolo Relazionale distinguendosi dai linguaggi d'interrogazione procedurali, che specificano i passi computazionali per costruire tabella richiesta a partire dal contenuto della base di dati. Una volta accettata la domanda, questa viene passata al programma del sistema di gestione che tende ad ottimizzare il calcolo (*query optimizer*).

Una domanda è espressa in *SQL* da un'istruzione (`statement`) strutturata che ha tre componenti o *clausole*

```
SELECT    ...
FROM      ...
WHERE     ...
```

Argomento della clausola `SELECT` è una lista di attributi che formano lo schema della tabella prodotta in risposta alla domanda.

Argomento della clausola `FROM` è una lista di *table* (tra quelle definite nella base di dati o da queste derivate).

La clausola `WHERE` è facoltativa e laddove è presente ha come argomento una condizione logica che seleziona le ennuple che entreranno a far parte della tabella fornita in risposta alla domanda. Più precisamente, l'argomento della clausola `WHERE` è un'espressione booleana che si ottiene combinando *predicati semplici* con gli operatori `AND`, `OR` e `NOT`. (La sintassi dà la precedenza a `NOT` ma non stabilisce alcuna priorità tra `AND` e `OR`.) Ciascun predicato semplice usa gli operatori per confrontare da un lato un'espressione costruita a partire da attributi e dall'altro un valore costante o un'altra espressione. Per il momento, ci limitiamo a mostrare a titolo d'esempio alcune semplici domande sulla base di dati `CINEMA`.

— Domanda: *Quali sono i titoli dei film diretti da Fellini a partire dal 1980* ?

```
SELECT titolo
FROM Film
WHERE anno >= 1980 AND regista = 'Fellini';
```

— Domanda: *Quali sono i registi dei film usciti dal 1980 in poi*?

```
SELECT regista
FROM Film
WHERE anno >= 1980;
```

Se volessimo che la risposta non contenga duplicati, la esprimeremmo

```
SELECT DISTINCT regista
FROM Film
WHERE anno >= 1980;
```

Se volessimo che la risposta alla domanda sia leggibile per una persona di lingua inglese, la esprimeremmo

```
SELECT titolo AS title, anno AS year, regista AS director, durata
AS length
FROM Film
WHERE anno >= 1980;
```

— Domanda: *Qual è lo stipendio mensile degli impiegati che hanno cognome Rossi*?

```
SELECT stipendio/12 AS stipendio_mensile
FROM Impiegato
WHERE cognome = 'Rossi';
```

— Domanda: *Dal 1980 in poi, quanti film sono stati prodotti*?

```
SELECT COUNT(*)
FROM Film
WHERE anno > 1980;
```

— Domanda: *Dal 1980 in poi, quanti film sono usciti ogni anno* ?

```
SELECT anno, COUNT(*)
FROM Film
WHERE anno >= 1980
```

```
GROUP BY anno;
```

— Domanda: *Qual è la durata media dei film diretti da Fellini* ?

```
SELECT AVG(durata)
FROM Film
WHERE regista = 'Fellini';
```

C'è da dire che, per la possibile presenza di attributi che possono assumere valori nulli (cioè di attributi che non siano stati dichiarati `NOT NULL`), la logica delle condizioni contenute nella clausola `WHERE` è "a tre valori": Vero, Falso, Sconosciuto. Pertanto una condizione è vera per una data ennupla se e solo se il suo valore non è né falso né sconosciuto. Per determinare il valore di una condizione che include i connettivi logici, basta interpretare

— i tre valori come Vero = 1, Falso = 0, Sconosciuto = ½
— `AND` = *min*, `OR` = *max*, `NOT` (*x*) = 1 – *x*.

Ad esempio

Vero `AND` (Falso `OR` `NOT` (Sconosciuto)) = *min* (1, *max* (0, 1 – ½)) = ½

Così, dal momento che nella *table* `Film` l'attributo `durata` non è stato dichiarato `NOT NULL`, la condizione

```
durata < 90 OR durata >= 90
```

è sempre vera per ogni ennupla di una relazione di nome `Film` in cui `durata` ha un valore diverso da `NULL`, ma non lo è più ad esempio per l'ennupla (`La dolce vita, 1960, Fellini, NULL`) perché il suo valore è sconosciuto: *max* (½,½) = ½.

1.3 *Aggiornamento dei Dati*

AGGIUNTA

Alla relazione di nome `Tab` viene aggiunta una singola ennupla *t*. L'operazione è sintatticamente corretta se *t* è un'ennupla su *R*, cioè $t \in DOM(R)$. L'operazione poi è efficace se l'ennupla *t* non appartiene già alla relazione di nome `Tab`.

Ad esempio, per una *table* di nome Nomi con schema {nome, cognome} l'aggiunta dell'ennupla (Andrea', 'Rossi') si esprime in *SQL* con l'istruzione

```
INSERT [INTO]   Nomi
VALUES          (nome = 'Andrea', cognome = 'Rossi');
```

CANCELLAZIONE

Dalla relazione di nome Tab vengono eliminate tutte le ennuple che soddisfano una certa condizione *C*. L'operazione è sintatticamente corretta se gli attributi coinvolti nella condizione *C* appartengono ad *R*. L'operazione poi è efficace se la relazione di nome Tab contiene almeno un'ennupla che soddisfa la condizione *C*.

Un caso tipico si ha quando la condizione *C* assume la forma $X = x$, in cui X è un sottoinsieme di R ed x è un'ennupla su X, cioè $x \in DOM(X)$. Se poi X è una sovrachiave del modello per le relazioni di nome Tab, allora il risultato della cancellazione sarà una relazione con un'ennupla in meno (sempreché l'operazione sia efficace).

Ad esempio, per una *table* di nome Indirizzi il cui schema contenga l'attributo provincia, la cancellazione degli indirizzi in provincia di Torino si esprime in *SQL* con l'istruzione

```
DELETE [FROM]   Indirizzi
WHERE           provincia = 'TO';
```

Se la *table* contenuta nella clausola DELETE è una *table* madre, l'operazione di cancellazione di un'ennupla dalla tabella madre vanno a influenzare la tabella figlia. Abbiamo ancora quattro possibilità:

con l'istruzione CASCADE vengono cancellate dalla tabella figlia tutte le ennuple in cui X assume il valore di Y che è stato cancellato;

con l'istruzione SET NULL nella tabella figlia tutte le ennuple in cui X assume il valore di Y che è stato cancellato, ad X viene assegnato il *valore nullo*;

con l'istruzione SET DEFAULT nella tabella figlia tutte le ennuple in cui X assume il valore di Y che è stato cancellato, ad X viene assegnato il *valore di default*;

con l'istruzione `NO ACTION` il valore di *X* non viene cancellato.

MODIFICA

Ogni ennupla *t* della relazione di nome `Tab` che soddisfa una certa condizione *C* viene modificata in maniera tale che per la sua versione aggiornata *t′* si abbia $t'[Y] = y$ e $t'[R–Y] = t[R–Y]$. L'operazione è sintatticamente corretta se gli attributi coinvolti nella condizione *C* appartengono ad *R* e se *Y* è un sottoinsieme di *R* ed *y* è un'ennupla su *Y*, cioè $y \in DOM(Y)$. L'operazione poi è efficace se la relazione di nome `Tab` contiene almeno un'ennupla che soddisfa la condizione *C*.

Ad esempio, per una *table* di nome `Nomi` il cui schema contenga l'attributo `nome`, la modifica dei nomi in cui il nome è stato erroneamente scritto `Tilde` invece di `Matilde` si esprime in *SQL* con il comando

```
UPDATE    Nomi
SET       nome = 'Matilde'
WHERE     nome = 'Tilde';
```

Se la *table* contenuta nella clausola `UPDATE` è una *table* madre, l'operazione di modifica di un'ennupla dalla tabella madre vanno a influenzare la tabella figlia. Supponiamo che la modifica coinvolga uno o più attributi in *Y*. Abbiamo quattro possibilità:

con l'istruzione `CASCADE` nella tabella figlia tutte le ennuple in cui *X* assume il valore di *Y* che è stato modificato, ad *X* viene assegnato il nuovo valore di *Y*;

con l'istruzione `SET NULL` nella tabella figlia tutte le ennuple in cui *X* assume il valore di *Y* che è stato modificato, ad *X* viene assegnato il *valore nullo*;

con l'istruzione `SET DEFAULT` nella tabella figlia tutte le ennuple in cui *X* assume il valore di *Y* che è stato modificato, ad *X* viene assegnato il *valore di default*;

con l'istruzione `NO ACTION` il valore di *X* non viene modificato.

Capitolo 2

Interrogazione

Il potere espressivo del linguaggio *SQL* è tale che ogni domanda che possa esprimersi nell'Algebra Relazionale o nell'equivalente versione ristretta del Calcolo Relazionale può esprimersi in *SQL*. Ad esempio, all'espressione del Calcolo Relazionale

$$\begin{aligned}\{x(\text{titolo}) \mid\ & \exists\, y(\text{titolo}, \text{anno}, \text{regista}, \text{durata}) \\ & \text{Film}(y) \wedge \\ & y(\text{titolo}) = x(\text{titolo}) \wedge \\ & y(\text{regista}) = \texttt{Fellini} \wedge\ y(\text{anno}) \geq 1980\}\end{aligned}$$

e all'espressione dell'Algebra Relazionale

$$\pi_{\text{titolo}}\,(\sigma_{\text{regista} = \texttt{Fellini} \wedge \text{anno} \geq 1980}\ (\text{Film}))$$

corrisponde l'istruzione

```
SELECT    titolo
FROM      Film
WHERE     regista = 'Fellini' AND
          anno >= 1980;
```

Come si è detto, una tipica istruzione *SQL* è formata dalle tre clausole `SELECT`, `FROM` e `WHERE`.

```
SELECT    lista_di_attributi
FROM      lista_di_Table
[WHERE    condizione]
```

dove le parentesi quadre stanno ad indicare che la clausola `WHERE` è opzionale.

In linea di massima, la risposta ad una domanda in un certo istante si calcola come segue:

– si prende il prodotto cartesiano delle tabelle che nello stato corrente della base di dati corrispondono alle *table* elencate nella clausola `FROM`;

– nella tabella che si ottiene si vanno ad esaminare una ad una le righe e si selezionano quelle che soddisfano la condizione che è argomento della clausola WHERE (sempreché sia presente); di ciascuna di queste righe si prende la restrizione sugli attributi presenti nella clausola SELECT e la si inserisce nella tabella che verrà prodotta in risposta alla domanda, cosicché la tabella avrà come schema la lista degli attributi contenuti nella clausola SELECT.

Introduciamo ora costrutti man mano sempre più complessi.

2.1 *La clausola* SELECT

SELECT *attributo* [AS] *alias* {, *attributo* [AS] *alias* }

Qui *attributo* è un attributo presente nello schema di una delle *table* elencate nella clausola FROM oppure è un'espressione, e *alias* è una possibile ridenominazione di *attributo*.

```
SELECT   titolo AS title, anno AS year, regista AS director,
         durata AS length
FROM     Film
WHERE    anno >= 1980;
```

```
SELECT   stipendio/12 AS stipendio_mensile
FROM     Impiegato
WHERE    cognome = 'Rossi';
```

Come argomento della clausola SELECT possiamo avere anche il carattere speciale * che sta per tutti gli attributi delle *table* elencate nella clausola FROM.

```
SELECT   *
FROM     Film
WHERE    anno >= 1980;
```

2.2 *La clausola* FROM

FROM *Table* [AS] *Alias* {, *Table* [AS] *Alias* }

Qui *Table* è una *table* presente nella base di dati oppure è un'espressione di *table* della base di dati, e *Alias* è una possibile ridenominazione di *Table*.

— Domanda: *Quali sono gli attori che sono stati diretti da Fellini e in che ruolo* ?

```
SELECT    attore, ruolo
FROM      Cast, Film
WHERE     Cast.film = Film.titolo AND regista = 'Fellini';
```

— Domanda: *Trovare le coppie dei film diretti dallo stesso regista senza produrre coppie come* (*a*, *a*) *ed elencarle in ordine alfabetico.*

```
SELECT    titolo, regista
FROM      Film F1, Film F2
WHERE     F1.regista = F2.regista AND F1.titolo < F2.titolo;
```

— Domanda: *Trovare i nomi e i cognomi degli impiegati e le loro sedi di lavoro*.

```
SELECT    Impiegato.nome, Impiegato.cognome, Ufficio.sede
FROM      Impiegato,  Ufficio
WHERE     Impiegato.ufficio = Ufficio.nome;
```

Si osservi che, per evitare ambiguità, i nomi degli attributi di ogni *table* sono preceduti dal nome della *table* seguito da un punto, a meno che il riferimento alla *table* non crei ambiguità.
Per evitare di ripetere per intero i nomi delle *table* si possono usare degli acronimi:

```
SELECT    Imp.nome, Imp.cognome, Uff.sede
FROM      Impiegato AS Imp,  Ufficio AS Uff
WHERE     Imp.ufficio = Uff.nome;
```

o più semplicemente

```
SELECT    Imp.nome, Imp.cognome, Uff.sede
FROM      Impiegato Imp,  Ufficio Uff
WHERE     Imp.ufficio = Uff.nome;
```

2.3 *La clausola* WHERE

La clausola WHERE ammette come argomento un'espressione booleana combinando predicati semplici con gli operatori AND, OR e NOT. Come si è detto, la sintassi dà la precedenza a NOT ma non stabilisce alcuna priorità tra AND e OR.

Ciascun predicato semplice usa gli operatori per confrontare da un lato un'espressione costruita a partire da attributi e dall'altro un valore costante o un'altra espressione.

```
SELECT    nome
FROM      Impiegato
WHERE     cognome = 'Rossi' AND
          (ufficio = 'Amministrazione' OR
          ufficio = 'Produzione');
```

Infine la clausola WHERE può contenere la condizione

attributo IS [NOT] NULL

2.4 *Duplicati*

Una significativa differenza tra le tabelle e le relazioni è nei duplicati (che sono banditi dalle relazioni). Le tabelle possono contenere ennuple uguali e questo perché l'eliminazione di duplicati da una tabella è un'operazione dispendiosa. Se l'utente che pone una domanda vuole in risposta una tabella priva di duplicati deve specificarlo aggiungendo a SELECT la parola-chiave DISTINCT.

2.5 ORDER BY

Spesso l'utente è interessato a vedere le righe di una tabella ordinate in base ai valori crescenti o decrescenti di certi attributi. L'uso della clausola ORDER BY risponde a tale richiesta.

```
SELECT    nome, cognome
FROM      Impiegati
WHERE     ufficio = 'Produzione'
ORDER BY  stipendio DESC;

SELECT    *
FROM      Auto
ORDER BY  marca DESC, modello;
```

Capitolo 3

Operatori dell'Algebra Relazionale

3.1 JOIN

Già si è visto come combinare due *table* nell'esempio

```
SELECT    Imp.nome, Imp.cognome, Uff.sede
FROM      Impiegato Imp,  Ufficio Uff
WHERE     Imp.ufficio = Uff.nome;
```

L'*SQL* prevede anche l'uso dell'operatore JOIN. La domanda precedente può essere resa con l'equijoin:

```
SELECT    Imp.nome, cognome, Uff.sede
FROM      Impiegato Imp JOIN  Ufficio Uff
          ON Imp.ufficio = Uff.nome;
```

Si hanno versioni più sofisticate del *join* che per mettono di conservare tutte le ennuple di una o dell'altra *table* o di entrambe le *table*. Ad esempio per conoscere le persone con patente di guida e le loro autovetture, date le due *table*

```
Patente   (numero, nome, cognome)
Auto      (targa, marca, modello, patente)
```

Scriveremmo

```
SELECT    nome, cognome, numero, targa, marca, modello
FROM      Patente Pat JOIN  Auto A
          ON Pat.numero = A.patente;
```

Ma se volessimo conoscere le persone munite di patente di guida proprietari o non di autovetture, scriveremmo

```
SELECT    nome, cognome, numero, targa, marca, modello
FROM      Patente Pat LEFT JOIN  Auto A
          ON Pat.numero = A.patente;
```

Ovviamente, nella risposta alla domanda c'è da aspettarsi la presenza di *valori nulli* per gli attributi targa, marca e modello.

Viceversa, se volessimo conoscere le autovetture con o senza un proprietario, scriveremmo

```
SELECT    nome, cognome, numero, targa, marca, modello
FROM      Patente Pat RIGHT JOIN  Auto A
          ON Pat.numero = A.patente;
```

Ovviamente, nella risposta alla domanda c'è da aspettarsi la presenza di *valori nulli* per gli attributi nome, cognome e numero.

Infine se volessimo tutte le informazioni sulle persone munite di patente di guida e sulle autovetture, scriveremmo

```
SELECT    nome, cognome, numero, targa, marca, modello
FROM      Patente Pat FULL JOIN  Auto A
          ON Pat.numero = A.patente;
```

Ovviamente, nella risposta alla domanda c'è da aspettarsi la presenza di *valori nulli* per gli attributi nome, cognome, numero, targa, marca e modello.

3.2 *Operatori insiemistici*

Gli operatori insiemistici dell'Algebra Relazionale si esprimono in *SQL* con UNION, INTERSECT e EXCEPT (MINUS) con la differenza che non si richiede che gli schemi delle due tabelle siano identici ma solo che gli schemi siano compatibili.

```
SELECT    nome
FROM      Impiegato
UNION
SELECT    cognome
FROM      Impiegato;
```

```
SELECT    nome
FROM      Impiegato
EXCEPT
```

```
SELECT    cognome
FROM      Impiegato;
```

In genere il risultato dell'esecuzione di istruzioni che contengono UNION, INTERSECT e EXCEPT (MINUS) è una tabella priva di duplicati. L'aggiunta dell'opzione ALL conserva i duplicati.

```
SELECT    nome
FROM      Impiegato
WHERE     ufficio <> 'Amministrazione'
UNION ALL
SELECT    cognome
FROM      Impiegato
WHERE     ufficio <> 'Amministrazione';
```

Capitolo 4

Istruzioni complesse

4.1 AS

Già si è visto l'uso di un *alias* per una *table* rende più concisa la condizione contenuta nella clausola WHERE.

```
SELECT    I1.nome, I1.cognome
FROM      Impiegati I1, Impiegati I2
WHERE     I1.cognome = I1.cognome AND
          I1.nome <> I2.nome AND
          I1.ufficio = 'Produzione';
```

Ma un alias può servire anche a definire una nuova *table* espressa in termini delle *table* della base di dati.

— Domanda: *Quali sono i film diretti dal regista del film* La dolce vita ?

```
SELECT    titolo
FROM      Film,
          (SELECT    regista
          FROM       Film
          WHERE      titolo = 'La dolce vita') R
WHERE     Film.regista = R.regista;
```

Si osservi che questa istruzione corrisponde all'espressione del Calcolo Relazionale

$$
\begin{aligned}
\{x(\text{titolo}) \mid\ & \exists\, y(\text{titolo}, \text{anno}, \text{regista}, \text{durata}) \\
& \text{Film}(y) \wedge \\
& y(\text{titolo}) = x(\text{titolo}) \wedge \\
& \qquad \exists\, u(\text{regista}) \\
& \qquad y(\text{regista}) = u(\text{regista}) \wedge \\
& \qquad\qquad \exists\, v(\text{titolo}, \text{anno}, \text{regista}, \text{durata}) \\
& \qquad\qquad \text{Film}(v) \wedge \\
& \qquad\qquad v(\text{regista}) = u(\text{regista}) \wedge \\
& \qquad\qquad v(\text{titolo}) = \text{'La dolce vita'}\}
\end{aligned}
$$

e la *table* R corrisponde all'espressione del Calcolo Relazionale

$$\{u(\text{regista}) \mid \exists\, v(\text{titolo}, \text{anno}, \text{regista}, \text{durata})$$
$$\text{Film}(v) \wedge$$
$$v(\text{regista}) = u(\text{regista}) \wedge$$
$$v(\text{titolo}) = \text{'La dolce vita'}\}$$

— Domanda: *Quali sono gli attori che hanno recitato in un film diretto dal regista del film* La dolce vita ?

```
SELECT    attore
FROM      Cast,
          (SELECT    titolo
          FROM       Film
          WHERE      regista =
                     (SELECT    regista
                     FROM       Film
                     WHERE      titolo = 'La dolce vita')) T
WHERE     Cast.film  = T.titolo;
```

$$\{x(\text{attore}) \mid \exists\, y(\text{film}, \text{anno}, \text{attore}, \text{ruolo})$$
$$\text{Cast}(y) \wedge$$
$$y(\text{attore}) = x(\text{attore}) \wedge$$
$$\exists\, u(\text{titolo})$$
$$u(\text{titolo}) = y(\text{film}) \wedge$$
$$\exists\, v(\text{titolo}, \text{anno}, \text{regista}, \text{durata})$$
$$\text{Film}(v) \wedge$$
$$v(\text{titolo}) = u(\text{titolo}) \wedge$$
$$\exists\, w(\text{titolo}, \text{anno}, \text{regista}, \text{durata})$$
$$\text{Film}(w) \wedge$$
$$w(\text{regista}) = v(\text{regista}) \wedge$$
$$w(\text{titolo}) = \text{'La dolce vita'}\}$$

Si osservi che la *table* T corrisponde all'espressione del Calcolo Relazionale

$$\{u(\text{titolo}) \mid \exists\, v(\text{titolo}, \text{anno}, \text{regista}, \text{durata})$$
$$\text{Film}(v) \wedge$$
$$v(\text{regista}) = u(\text{regista}) \wedge$$
$$v(\text{titolo}) = \text{'La dolce vita'}\}$$

4.2 *Istruzioni a struttura nidificata*

Riprendiamo la domanda

Quali sono i film diretti dal regista del film La dolce vita ?

Nel Calcolo Relazionale questa poteva esprimersi più semplicemente alla maniera seguente

$$
\begin{aligned}
\{x(\text{titolo}) \mid\ & \exists\, y(\text{titolo}, \text{anno}, \text{regista}, \text{durata}) \\
& \text{Film}(y) \wedge \\
& y(\text{titolo}) = x(\text{titolo}) \wedge \\
& \qquad \exists\, v(\text{titolo}, \text{anno}, \text{regista}, \text{durata}) \\
& \qquad \text{Film}(v) \wedge \\
& \qquad v(\text{regista}) = y(\text{regista}) \wedge \\
& \qquad v(\text{titolo}) = \text{'La dolce vita'}\}
\end{aligned}
$$

Anche in *SQL* possiamo esprimere la domanda in modo più semplice senza introdurre la *table* R nella clausola FROM.

```
SELECT    titolo
FROM      Film
WHERE     regista =
          (SELECT    regista
          FROM       Film
          WHERE      titolo = 'La dolce vita') ;
```

L'istruzione è sintatticamente corretta perché la tabella che risulta dalla valutazione della sottoistruzione

```
SELECT    regista
FROM      Film
WHERE     titolo = 'La dolce vita';
```

ha schema {regista} e contiene un'unica ennupla. Questa sottoistruzione si dice *annidata* nell'istruzione *principale*. Le istruzioni annidate possono essere *indipendenti* o *dipendenti dal contesto*. Nel primo caso, possono essere eseguite una volta per tutte e il risultato è una tabella che non dipende dal resto dell'istruzione principale. L'istruzione annidata precedente ne è un esempio. Nel secondo caso, la

loro esecuzione dipende da qualche parametro specificato nel resto dell'istruzione principale.

Riprendiamo poi la domanda

Quali sono gli attori che hanno recitato in un film diretto dal regista del film La dolce vita ?

Nel Calcolo Relazionale questa poteva esprimersi più semplicemente alla maniera seguente

$$
\begin{aligned}
\{x(\text{attore}) \mid\ & \exists\, y(\text{film}, \text{anno}, \text{attore}, \text{ruolo}) \\
& \text{Cast}(y) \wedge \\
& y(\text{attore}) = x(\text{attore}) \wedge \\
& \qquad \exists\, v(\text{titolo}, \text{anno}, \text{regista}, \text{durata}) \\
& \qquad \text{Film}(v) \wedge \\
& \qquad v(\text{titolo}) = y(\text{film}) \wedge \\
& \qquad\qquad \exists\, w(\text{titolo}, \text{anno}, \text{regista}, \text{durata}) \\
& \qquad\qquad \text{Film}(w) \wedge \\
& \qquad\qquad w(\text{regista}) = v(\text{regista}) \wedge \\
& \qquad\qquad w(\text{titolo}) = \text{'La dolce vita'}\}
\end{aligned}
$$

Anche in *SQL* possiamo esprimere la domanda in modo più semplice senza introdurre la *table* T nella clausola FROM

```
SELECT   attore
FROM     Cast
         WHERE    film  IN
         (SELECT  titolo
         FROM     Film
         WHERE    regista =
                  (SELECT   regista
                  FROM      Film
                  WHERE     titolo = 'La dolce vita'));
```

Questa istruzione è corretta perché la tabella che risulta dalla valutazione di

```
SELECT    titolo
FROM      Film
WHERE     regista =
          (SELECT    regista
          FROM       Film
          WHERE      titolo = 'La dolce vita');
```

ha schema {titolo} e l'attributo titolo è compatibile con l'attributo film (cioè titolo e film hanno lo stesso dominio) e, inoltre, può contenere più di un'ennupla. Si osservi che l'istruzione ha tre livelli e le due istruzioni annidate sono indipendenti dal contesto.

Nelle due precedenti istruzioni a struttura nidificata abbiamo visto che la condizione nella clausola WHERE è del tipo

attributo = (SELECT ...)
attributo IN (SELECT ...)

Più in generale nelle istruzioni nidificate troviamo condizioni del tipo

attributo θ (SELECT ...) θ ∈ {=, <>, >, <, >=, <=}
attributo IN (SELECT ...)
attributo NOT IN (SELECT ...)
attributo θ ANY (SELECT ...)
attributo θ ALL (SELECT ...)
EXISTS (SELECT ...)
NOT EXISTS (SELECT ...)

Le due formule *attributo* IN (SELECT ...) e *attributo* = ANY (SELECT ...) sono equivalenti.

Consideriamo ad esempio l'istruzione

```
SELECT    I1.nome
FROM      Impiegati I1, Impiegati I2
WHERE     I1.nome <> I2.nome AND
          I1.ufficio = 'Produzione';
```

Questa è equivalente all'istruzione a struttura nidificata

```
SELECT    nome
FROM      Impiegati
WHERE     nome = ANY
          (SELECT    nome
          FROM       Impiegati
          WHERE      ufficio = 'Produzione');
```

— Domanda: *Qual è l'ufficio dell'impiegato che guadagna più di tutti gli altri* ?

```
SELECT    ufficio
FROM      Impiegato
WHERE     stipendio > = ALL
          (SELECT    stipendio
          FROM       Impiegato);
```

Concludiamo con qualche esempio di sottoistruzioni dipendenti dal contesto.

— Domanda: *Quali sono i contribuenti fiscali che hanno omonimi* ?

```
SELECT    *
FROM      Persona P
WHERE     EXISTS
          (SELECT    *
          FROM       Persona P1
          WHERE      P1.nome = P.nome AND
                     P1.cognome = P.cognome AND
                     P1.codice_fiscale <> P. codice_fiscale);
```

Si osservi che l'esecuzione dell'istruzione annidata

```
SELECT    *
FROM      Persona P1
WHERE     P1.nome = P.nome AND
          P1.cognome = P.cognome AND
          P1.codice_fiscale <> P. codice_fiscale
```

dipende dal contesto cioè dal valore di P.nome, P.cognome e P. codice_fiscale. Pertanto, la sua esecuzione segue (e non precede) la costruzione della tabella P.

— Domanda: *Quali sono i contribuenti fiscali che non hanno omonimi* ?

```
SELECT    *
FROM      Persona P
WHERE     NOT EXISTS
          (SELECT    *
          FROM       Persona P1
          WHERE      P1.nome = P.nome AND
                     P1.cognome = P.cognome AND
                     P1.codice_fiscale <> P. codice_fiscale);
```

Un'altra maniera di porre la stessa domanda è

```
SELECT    *
FROM      Persona P
WHERE     (nome, cognome) NOT IN
          (SELECT    nome, cognome
          FROM       Persona Q
          WHERE      Q.codice_fiscale <> P. codice_fiscale);
```

— Domanda: *Quali sono gli attori che hanno recitato in film di cui erano anche registi* ?

```
SELECT    nome
FROM      Attore A
WHERE     nome IN
          (SELECT    attore
          FROM       Cast
          WHERE      (film, anno) IN
                     (SELECT    titolo, anno
                     FROM       Film
                     WHERE      regista = A.nome));
```

Si osservi che l'esecuzione dell'istruzione annidata

```
SELECT    titolo, anno
FROM      Film
WHERE     regista = A.nome
```

dipende dal contesto cioè dal valore di A.nome. Pertanto, la sua esecuzione segue (e non precede) la costruzione della tabella A.

Capitolo 5

Operatori di aggregazione

Gli operatori di aggregazione costituiscono una delle maggiori estensioni dell'*SQL* rispetto all'Algebra Relazionale perché "operano" non su ennuple ma su tabelle. Il primo operatore che consideriamo è COUNT del quale si hanno diverse versioni:

> COUNT (*) restituisce il numero di ennuple nella tabella selezionata dalle clausole FROM and WHERE
>
> COUNT (DISTINCT *lista di attributi X*) restituisce il numero dei valori distinti di *X* nella tabella selezionata dalle clausole FROM and WHERE
>
> COUNT (ALL *lista di attributi X*) restituisce il numero dei valori di *X* diversi dal valore nullo nella tabella selezionata dalle clausole FROM and WHERE.

Eccone un esempio.

```
SELECT    COUNT (*)
FROM      Film
WHERE     anno >= 1980;

SELECT    COUNT (DISTINCT regista)
FROM      Film
WHERE     anno >= 1980;
```

Spesso ha interesse avere una distribuzione del risultato dell'operatore COUNT per classi di ennuple. A questo scopo, l'*SQL* offre la possibilità di usare la clausola GROUP BY come nel seguente esempio

```
SELECT    anno, COUNT (*)
FROM      Film
WHERE     anno >= 1980
GROUP BY  anno;
```

In tal caso viene prima costruita la tabella che si otterrebbe eseguendo l'istruzione

```
SELECT    *
FROM      Film
WHERE     anno >= 1980;
```

Quindi, vengono raggruppate le righe con lo stesso valore di anno e a ciascuno di questi raggruppamenti viene applicato l'operatore COUNT.

Oltre all'operatore COUNT, sono disponibili i quattro operatori di aggregazione

```
SUM  MAX MIN  AVG
```

che ammettono come argomento solo espressioni che rappresentano valori numerici o intervalli di tempo.

```
SELECT    SUM(stipendio), MAX(stipendio), AVG(stipendio)
FROM      Impiegato
WHERE     ufficio = 'Produzione';

SELECT    AVG(durata)
FROM      Film
WHERE     regista = 'Fellini';

SELECT    AVG(durata)
FROM      Film
WHERE     regista = 'Fellini'
HAVING    AVG(durata) > 100;

SELECT    AVG(stipendio)
FROM      Impiegato I,  Ufficio U
WHERE     I.ufficio = U.nome;
```

Riprendiamo la domanda

Qual è l'ufficio dell'impiegato che guadagna più di tutti gli altri ?

Per la quale abbiamo usato l'istruzione

```
SELECT    ufficio
FROM      Impiegato
```

```
WHERE     stipendio > = ALL
          (SELECT    stipendio
          FROM       Impiegato);
```

Possiamo anche esprimerla utilizzando MAX

```
SELECT    ufficio
FROM      Impiegato
WHERE     stipendio =
          (SELECT    MAX(stipendio)
          FROM       Impiegato);
```

Consideriamo ora il seguente esempio

```
SELECT    ufficio, SUM (stipendio)
FROM      Impiegato
GROUP BY  ufficio;
```

L'esecuzione di questa istruzione consiste nel costruire innanzitutto la tabella che risulta dall'esecuzione dell'istruzione

```
SELECT    ufficio, stipendio
FROM      Impiegato;
```

quindi, vengono raggruppate le righe con lo stesso valore di ufficio e a ciascuno di questi raggruppamenti viene applicato l'operatore SUM (stipendio).

Infine, abbiamo anche la possibilità di selezionare i raggruppamenti derivanti dalla clausola GROUP BY utilizzando la clausola HAVING come illustrato dal seguente esempio

```
SELECT    ufficio, SUM (stipendio) AS totale
FROM      Impiegato
GROUP BY  ufficio
HAVING    SUM (stipendio) > 100;
```

Capitolo 6

Vincoli

Oltre ai vincoli predefiniti visti al Paragrafo 1.1.2, l'*SQL* offre la possibilità di definire vincoli più complessi.
Per specificare vincoli di tipo intrarelazionale, si fa ricorso alla clausola CHECK. Come semplice esempio, la clausola

```
CHECK (...)
```

Ad esempio, per esprimere la condizione che matricola è una sovrachiave della *table* Impiegato si scrive

```
CHECK     (matricola IS NOT NULL AND
           1 = (  SELECT COUNT(*)
                  FROM Impiegato I
                  WHERE matricola = I.matricola)
          )
```

La condizione deve essere sempre soddisfatta, cioè da ogni tabella di nome Impiegato che faccia parte di uno stato della base di dati. La clausola CHECK è abbastanza generale da consentire di esprimere anche i vincoli predefiniti di NOT NULL, UNIQUE e PRIMARY KEY.

In generale, per specificare vincoli generici, di tipo intrarelazionale o interrelazionale, l'*SQL* mette a disposizione la clausola

```
CREATE ASSERTION    Nome
CHECK               (Condizione)
```

Così, il vincolo che in ogni relazione di nome Impiegato sia presente almeno un'ennupla può esprimere alla maniera seguente

```
CREATE ASSERTION    AlmenoUnImpiegato

CHECK               (1 <= (SELECT COUNT(*)
                           FROM Impiegato)
                    )
```

Ogni vincolo d'integrità definito con la clausola CHECK o con la clausola ASSERTION ha una sua propria modalità di controllo che può essere *immediata* o *differita*.

La modalità di controllo immediata verifica che il vincolo sia soddisfatto dopo ogni operazione di aggiornamento, pena l'annullamento dell'operazione. A tutti i vincoli predefiniti NOT NULL, UNIQUE e PRIMARY KEY è associata la modalità di controllo immediata.

La modalità di controllo differita verifica che il vincolo sia soddisfatto dopo ogni specificata sequenza (*transazione*) di operazioni di aggiornamento, pena l'annullamento dell'intera sequenza di operazioni (rollback). A titolo illustrativo, supponiamo di aver appena creato una *table* Impiegato il cui schema contiene l'attributo nome con vincolo NOT NULL e l'attributo ufficio con vincolo NOT NULL, e una *table* Dipartimento il cui schema contiene l'attributo nome con vincolo NOT NULL e l'attributo direttore con vincolo NOT NULL; supponiamo anche di aver imposto i vincoli referenziali che rimandano ufficio a Dipartimento.nome e direttore a Impiegato.nome. A questo punto, se entrambi i vincoli referenziali avessero modalità di controllo immediata, non ci sarebbe modo di creare né una tabella Impiegato né una tabella Dipartimento per il semplice motivo che ogni operazione di aggiunta verrebbe annullata. La modalità di controllo differita consente di superare questa difficoltà.