Capitolo 3

Linguaggio SQL

Il linguaggio SQL, abbreviazione di *Structured Query Language*, è il linguaggio di definizione e manipolazione dei dati supportato dalla totalità dei DBMS relazionali oggi disponibili. SQL ha rappresentato una tappa fondamentale nello sviluppo della tecnologia delle basi di dati, in quanto è stato il primo linguaggio con caratteristiche dichiarative progettato specificatamente per l'accesso e la manipolazione di collezioni di dati. SQL è basato sui linguaggi presentati nel Capitolo 2 per l'interrogazione di basi di dati relazionali, cioè l'algebra ed il calcolo relazionale. Le operazioni dell'algebra relazionale, in particolare, sono fondamentali per comprendere i tipi di interrogazioni che possono essere poste ad una base di dati relazionale. Nell'algebra relazionale, tuttavia, un'interrogazione è formulata come una sequenza di operazioni che, una volta eseguite, producono il risultato richiesto. L'utente deve quindi specificare *in quale ordine* le operazioni devono essere eseguite. In SQL, al contrario, l'utente specifica solo *quale* deve essere il risultato, caratterizzandolo in maniera dichiarativa. Questo permette al DBMS di determinare strategie efficienti per l'esecuzione delle operazioni di accesso e manipolazione dei dati, mediante un processo detto di *ottimizzazione*, che verrà discusso nel Capitolo 7. SQL include inoltre alcune operazioni dell'algebra (in particolare, le operazioni insiemistiche di unione, differenza, intersezione) ma si basa in misura maggiore sul calcolo relazionale, anche se la sintassi di SQL è più semplice da utilizzare.

SQL differisce quindi dai linguaggi di programmazione, quali C, C++, Java, per il fatto di essere orientato alla manipolazione di collezioni di dati e di permettere all'utente di specificare quale deve essere il risultato di una data operazione, senza dover indicare come eseguire l'operazione stessa. La specifica del risultato di un'operazione è espressa tramite condizioni sul contenuto dei dati. La disponibilità di un linguaggio ad alto livello, dichiarativo, specializzato per la manipolazione dei dati, è molto importante dal punto di vista dello sviluppo delle applicazioni. SQL manca, però, dei costrutti di controllo propri dei linguaggi di programmazione di uso generale e da solo non è quindi sufficiente alla programmazione di gran parte delle applicazioni reali. Pertanto, oltre a poter essere usato in modalità stand-alone (tipicamente tramite un'interfaccia interattiva), SQL è "combinato" con linguaggi di programmazione secondo diversi approcci, che verranno discussi in dettaglio nel Capitolo 4.

Data la sua rapida diffusione come linguaggio per la gestione dei dati, SQL è stato oggetto di numerose estensioni e standardizzazioni. Il linguaggio è diventato standard ufficiale nel 1986 ed ha subito significative revisioni nel 1992 (SQL2, noto anche come SQL-92) e nel 1999 (SQL:1999, noto anche come SQL3). SQL:1999 ha introdotto numerose estensioni ad SQL tra cui i trigger (non standardizzati in SQL2) ed alcuni costrutti del paradigma ad oggetti, tanto che il modello dei dati corrispondente non è più un modello relazionale puro, ma un modello relazionale ad oggetti. Tali caratteristiche verranno discusse in dettaglio nei Capitoli 10 e 11. Lo standard più recente è lo standard SQL:2003. Nel seguito della trattazione, faremo implicitamente riferimento a tale versione dello standard. Indicheremo esplicitamente la versione solo quando vorremo evidenziare una differenza significativa rispetto ad altre versioni.

Uno dei motivi del successo sul mercato delle basi di dati relazionali è sicuramente legato all'esistenza del linguaggio standard SQL. Avere a disposizione un linguaggio standard, infatti, permette agli utenti di non doversi preoccupare troppo della migrazione delle loro applicazioni da un sistema relazionale ad un altro, nel caso non siano soddisfatti del particolare prodotto. Tale conversione non dovrebbe essere troppo costosa nel caso in cui entrambi i sistemi siano conformi allo standard. Analogamente, possiamo scrivere programmi applicativi che contengono comandi SQL, come verrà discusso nel Capitolo 4, che accedono ai dati memorizzati in due o più DBMS relazionali utilizzando gli stessi comandi di manipolazione dei dati. In realtà, del linguaggio SQL esistono diverse implementazioni e vi sono differenze anche significative tra i DBMS relazionali commerciali. Tipicamente, le funzionalità di base del linguaggio sono le stesse in ogni implementazione; esistono, invece, differenze nelle funzionalità più avanzate. A partire da SQL:1999, lo standard SQL definisce un vasto insieme di caratteristiche (*SQL Core*) che devono essere fornite da un DBMS per potersi dichiarare conforme allo standard. Nella trattazione, presenteremo le principali funzionalità di tale nucleo del linguaggio, che dovrebbero quindi essere fornite in tutte le implementazioni, evidenziando esplicitamente eventuali caratteristiche che ancora non sono diffusamente supportate nelle implementazioni.

Il documento che specifica lo standard è strutturato in nove diverse parti. Tra queste, il materiale coperto in questo capitolo è contenuto in *SQL/Foundation*, che è la parte più estesa e più importante, che specifica il "nucleo" del linguaggio (SQL Core), corrispondente al minimo livello di conformità. Nel Capitolo 4 verrà invece trattata la parte *SQL/PSM (Persistent Stored Module)*, che specifica costrutti procedurali simili a quelli che si trovano nei comuni linguaggi di programmazione.

SQL comprende istruzioni per la definizione, l'interrogazione e l'aggiornamento dei dati. Quindi, rispetto alla terminologia introdotta nel Capitolo 1, è sia un DDL che un DML. Tutte le implementazioni di SQL forniscono inoltre comandi SDL per la definizione di strutture di memorizzazione e di strutture ausiliarie di accesso (vedi Capitolo 7), ma tali comandi non sono stati standardizzati in SQL.

In questo capitolo, introdurremo dapprima il linguaggio di definizione dei dati, in seguito presenteremo il linguaggio di interrogazione e le altre componenti del

Operazione	DDL	DML
Creazione	`CREATE`	`INSERT`
Modifica	`ALTER`	`UPDATE`
Cancellazione	`DROP`	`DELETE`

Tabella 3.1: Comandi SQL di aggiornamento a livello di schema e di istanza

linguaggio di manipolazione dei dati. La specifica di vincoli di integrità e viste completerà la trattazione.

Nel modello relazionale abbiamo usato i termini di "relazione" ed "attributo"; il linguaggio SQL differisce rispetto a tale terminologia in quanto usa i termini di "tabella" (table) e "colonna" (column). Nel seguito di questo capitolo continueremo ad usare il termine "relazione" come equivalente del termine "tabella" ed useremo il termine "colonna" come equivalente del termine "attributo". Nel presentare la sintassi del linguaggio, come usuale, verranno utilizzate la parentesi quadre per racchiudere componenti la cui specifica non è obbligatoria e parentesi graffe per racchiudere componenti alternative, di cui una deve essere obbligatoriamente specificata, separate dal simbolo '|'. Un asterisco ('$*$') indicherà la ripetibilità di una componente.

3.1 Linguaggio di definizione dei dati

Le relazioni possono essere definite tramite il comando `CREATE`, modificate tramite il comando `ALTER` e cancellate tramite il comando `DROP`. Tali comandi costituiscono i comandi principali del DDL. La Tabella 3.1 riassume i principali comandi offerti da SQL, a livello di schema e di istanza, rispettivamente, per aggiornare lo schema ed i dati. In questo paragrafo discuteremo i comandi del DDL, mentre i comandi di aggiornamento del DML saranno discussi nel Paragrafo 3.3, dopo aver presentato il linguaggio di interrogazione.

Nel seguito del paragrafo, introdurremo innanzitutto i tipi di dato forniti da SQL, che corrispondono ai domini del modello relazionale, specificando quindi i possibili contenuti delle colonne delle relazioni. Presenteremo poi i comandi per la creazione e successivamente quelli per la cancellazione e la modifica di relazioni.

3.1.1 Tipi di dato

In questo paragrafo discuteremo i tipi di dato *predefiniti* di SQL. SQL prevede la possibilità di utilizzare anche tipi *definiti dall'utente*, che verranno trattati nell'ambito delle caratteristiche relazionali ad oggetti di SQL, discusse nel Capitolo 10. I tipi predefiniti sono suddivisi in tre categorie principali: *tipi numerici*, *tipi carattere* e *tipi temporali*. Ognuna di queste categorie include, a sua volta, delle sotto-categorie di seguito discusse.

Tipi numerici. I tipi numerici sono classificati in *tipi numerici esatti*, che permettono di rappresentare valori interi e valori decimali in virgola fissa, e *tipi numerici approssimati*, che permettono di rappresentare valori numerici approssimati con rappresentazione in virgola mobile, cioè tramite mantissa ed esponente. I tipi numerici esatti includono i seguenti tipi:

- `INTEGER`. Rappresenta i valori interi. La *precisione* (numero totale di cifre) di questo tipo di dato è espressa in numero di bit o cifre, a seconda della specifica implementazione di SQL.

- `SMALLINT`. Anche questo tipo di dato rappresenta valori interi. I valori di questo tipo sono usati per eventuali ottimizzazioni in quanto richiedono minore spazio di memorizzazione. L'unico requisito è che la precisione di questo tipo di dato sia non maggiore della precisione del tipo di dato `INTEGER`.

- `BIGINT`. Anche questo tipo di dato rappresenta valori interi. L'unico requisito è che la precisione di questo tipo di dato sia non minore della precisione del tipo di dato `INTEGER`.[1]

- `NUMERIC`. Questo tipo è caratterizzato da una precisione (numero totale di cifre) ed una *scala* (numero di cifre dopo la virgola decimale). La specifica di questo tipo di dato ha la forma `NUMERIC[(`p`[,`s`])]`, dove p è la precisione che viene richiesta ed s è la scala. Il valore di default per la precisione è 1, mentre il valore di default per la scala è 0.

- `DECIMAL`. È simile al tipo di dato `NUMERIC`. Analogamente a quest'ultimo, la specifica di questo tipo di dato ha la forma `DECIMAL[(`p`[,`s`])]`, dove p è la precisione che viene richiesta ed s è la scala. La differenza tra `NUMERIC` e `DECIMAL` è che nel primo la precisione con cui i valori sono implementati deve essere esattamente la precisione richiesta dalla specifica del tipo di dato (cioè p), mentre nel secondo caso la precisione con cui i valori sono implementati deve essere almeno uguale alla precisione richiesta (ma può anche essere maggiore).

I tipi numerici approssimati includono i seguenti tipi:

- `REAL`. Rappresenta valori a singola precisione in virgola mobile. La precisione dipende dalla specifica implementazione di SQL.

- `DOUBLE PRECISION`. Anche questo tipo di dato rappresenta valori in virgola mobile, di solito a doppia precisione, e la sua precisione dipende dalla specifica implementazione di SQL. La differenza tra il tipo di dato `REAL` e il tipo di dato `DOUBLE PRECISION` è che la precisione di quest'ultimo deve essere maggiore della precisione del tipo di dato `REAL`.

[1] Anche se non specificato nello standard, usualmente 64 bit invece dei 32 bit di `INTEGER`.

- `FLOAT`. Questo tipo di dato, a differenza dei due precedenti, permette di richiedere la precisione desiderata. Pertanto, la specifica di questo tipo di dato ha la forma `FLOAT[(`p`)]`, dove p è la precisione che viene eventualmente richiesta. La precisione minima che può essere specificata è 1, mentre la precisione di default e quella massima dipendono dalle specifiche implementazioni di SQL.

I valori dei tipi numerici esatti sono rappresentati dalle cifre corrispondenti, con eventualmente prefisso il segno e l'utilizzo del punto decimale (ad esempio, `4.0`, `-6`). I valori dei tipi numerici approssimati sono rappresentati in notazione mantissa ed esponente (ad esempio, `1256E-4`).

Tipi carattere. I tipi carattere includono i seguenti tipi:

- `CHARACTER`. Questo tipo di dato (spesso abbreviato in `CHAR`) permette di definire stringhe di caratteri di lunghezza predefinita. La specifica di questo tipo di dato ha la forma `CHAR[(`n`)]` dove n è la lunghezza delle stringhe. È ovviamente possibile utilizzare una stringa di lunghezza inferiore ad n come valore di tipo `CHAR(`n`)`, ma la stringa sarà completata con degli spazi fino a raggiungere la lunghezza n. Se non viene specificata alcuna lunghezza, il default è 1.

- `CHARACTER VARYING`. Questo tipo di dato (spesso abbreviato in `VARCHAR`) permette di definire stringhe di caratteri di una lunghezza massima predefinita. La specifica di questo tipo ha la forma `VARCHAR(`n`)` dove n è la lunghezza massima delle stringhe Notiamo che in questo caso la specifica della lunghezza massima è obbligatoria e può variare tra 1 ed un valore massimo che dipende dalla specifica implementazione di SQL. La differenza tra il tipo di dato `CHAR` e il tipo di dato `VARCHAR` è che nel primo per ogni stringa viene comunque allocato uno spazio la cui lunghezza in numero di caratteri è sempre pari alla lunghezza specificata n. Nel secondo tipo di dato si adottano invece strategie di memorizzazione delle stringhe differenti, che evitano sprechi di spazio.

I valori dei tipi carattere vanno racchiusi tra singoli apici, come in `'pulp fiction'`. In tali valori viene effettuata una distinzione tra caratteri maiuscoli e minuscoli.

Tipi temporali. I tipi temporali includono i seguenti tipi:

- `DATE`. Rappresenta le date espresse come anno (4 cifre), mese (2 cifre comprese tra 1 e 12), giorno (2 cifre comprese tra 1 e 31 ed ulteriori restrizioni a seconda del mese). Un valore di questo tipo viene rappresentato dalla parola chiave `DATE` seguita dalla stringa che rappresenta la data nel formato scelto. Sono possibili diversi formati di rappresentazione, in questo testo utilizzeremo il formato giorno - prime tre lettere del mese - anno. Un esempio di data in questo formato è `DATE '08-Ott-1969'`.

- `TIME`. Rappresenta i tempi espressi come ora (2 cifre comprese tra 0 e 23), minuto (2 cifre comprese tra 0 e 59) e secondo (2 cifre comprese tra 0 e 61).[2] La specifica ha la forma `TIME[(`p`)]`, dove p è l'eventuale numero di cifre frazionarie cui si è interessati, fino al microsecondo (6 cifre). Se non viene specificato p, il valore di default è 0. Un valore di questo tipo viene rappresentato dalla parola chiave `TIME` seguita dalla stringa che rappresenta il tempo, ad esempio `TIME '21:56:32.5'`.

- `TIMESTAMP`. Rappresenta una "concatenazione" dei due tipi di dato precedenti. Pertanto permette di rappresentare timestamp che consistono di: anno, mese, giorno, ora, minuto, secondo ed eventualmente microsecondo (6 cifre). Come per il tipo `TIME`, la specifica è `TIMESTAMP[(`p`)]`, dove p è l'eventuale numero di cifre frazionarie cui si è interessati. Un valore di questo tipo viene rappresentato dalla parola chiave `TIMESTAMP` seguita dalla stringa che rappresenta data e tempo, nei formati introdotti precedentemente, ad esempio `TIMESTAMP '08-Ott-1969 21:56:32.5'`.

- `INTERVAL`. Rappresenta una durata temporale in riferimento ad uno o più dei qualificatori tra `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE` e `SECOND`. I valori di questo tipo sono rappresentati dalla parola chiave `INTERVAL` seguita da una stringa che caratterizza la durata in termini di uno o due qualificatori. Se sono presenti due qualificatori, il primo è più ampio del secondo ed i due sono separati dalla parola chiave `TO`. In realtà, a seconda dei qualificatori utilizzati, distinguiamo tra intervalli *year-month* ed intervalli *day-time*. Ad esempio, `INTERVAL '3' YEAR` rappresenta un intervallo di durata 3 anni, `INTERVAL '3-11' YEAR TO MONTH` rappresenta un intervallo di durata 3 anni ed 11 mesi, mentre `INTERVAL '36 22:30' DAY TO MINUTE` rappresenta un intervallo di durata 36 giorni, 22 ore e 30 minuti.

I tipi predefiniti includono inoltre il *tipo booleano* `BOOLEAN`, i cui valori sono `TRUE`, `FALSE` ed `UNKNOWN`,[3] ed i tipi `BLOB` (Binary Large Object) e `CLOB` (Character Large Object), per la memorizzazione di stringhe di bit o di caratteri, rispettivamente, di dimensione elevata. Tali tipi sono utili, ad esempio, nella memorizzazione di immagini e testi.

Un'ultima importante questione riguarda la compatibilità tra i vari tipi di dato. Spesso, ad esempio, può essere necessario confrontare valori di colonne diverse e ovviamente i confronti possono essere eseguiti solo se le colonne hanno tipi di dato *compatibili*. Normalmente le specifiche compatibilità dipendono dal sistema e sono contenute nei manuali d'uso, insieme ad eventuali regole di conversione. In generale si può dire che tutti i tipi di dato numerici sono compatibili e tutti i tipi di dato carattere sono compatibili. Inoltre, il tipo di dato `DATE` è compatibile con i tipi di dato carattere e lo stesso vale per i tipi di dato `TIME` e `TIMESTAMP`.

[2] I secondi possono arrivare fino a 61 e non fino a 59 come ci aspetteremmo per il fenomeno noto come *leap second*: occasionalmente vengono aggiunti uno o due secondi ad un minuto per tenere il tempo "ufficiale" sincronizzato con il tempo siderale.

[3] Il valore `UNKNOWN` verrà discusso in dettaglio nel Paragrafo 3.2.6.

3.1.2 Creazione di relazioni

La creazione di relazioni avviene in SQL tramite l'uso del comando `CREATE TABLE`, che permette di specificare lo schema della relazione ed alcuni importanti vincoli di integrità, come discusso nei paragrafi seguenti.

3.1.2.1 Specifica dello schema di una relazione

La sintassi base del comando `CREATE TABLE`, che permette di creare una relazione specificandone lo schema, è la seguente:[4]

```
CREATE TABLE <nome relazione>
       (<specifica colonna> [,<specifica colonna>]*);
```

dove:

- `<nome relazione>` è il nome della relazione che viene creata;
- `<specifica colonna>` è una specifica di colonna, il cui formato è il seguente:

  `<nome colonna> <dominio> [DEFAULT <valore default>]`

  dove:

  - `<nome colonna>` è il nome della colonna (che deve essere distinto da tutti gli altri nomi di colonna della stessa relazione);
  - `<dominio>` è il dominio della colonna e deve essere uno dei tipi di dato di SQL;
  - la clausola `DEFAULT` specifica un valore di default per la colonna. Tale valore è quello assegnato ad una nuova tupla se nessun valore è specificato per la colonna al momento dell'inserimento della tupla. Il valore di default è un qualunque valore appartenente al dominio.

Esempio 3.1 La relazione `Video` della Figura 2.1 può essere definita in SQL come segue:

```
CREATE TABLE  Video
       (colloc   DECIMAL(4),
        titolo   VARCHAR(30),
        regista  VARCHAR(20),
        tipo     CHAR DEFAULT 'd');
```
□

Ad ogni relazione è associato un tipo, chiamato *tipo riga* (*row type*), che corrisponde al tipo delle tuple di tale relazione.

[4] In questo testo, per convenzione, useremo i caratteri maiuscoli per i comandi SQL. In realtà nelle parole riservate SQL è indifferente l'uso di caratteri maiuscoli o minuscoli; il linguaggio distingue tra caratteri maiuscoli e minuscoli solo nei valori di tipo stringa.

3.1.2.2 Vincoli di obbligatorietà, chiavi e chiavi esterne

Un aspetto importante nella definizione di uno schema di basi di dati riguarda la definizione dei vincoli di integrità. SQL prevede la specifica di vincoli di integrità, come parte del comando `CREATE TABLE`, e la verifica automatica di tali vincoli. Oltre alla lista delle specifiche di colonne, quindi, un comando di definizione di relazione spesso contiene anche la definizione di importanti vincoli, discussi nel seguito di questo paragrafo: obbligatorietà di colonne e specifica di chiavi e chiavi esterne. È anche possibile specificare vincoli di integrità aggiuntivi, relativi alle tuple della relazione od alle singole colonne di tali tuple, al momento della definizione di una relazione, ma, poiché tale specifica richiede l'utilizzo del linguaggio di interrogazione, rimandiamo la trattazione di questo aspetto al Paragrafo 3.4.

Un'ulteriore possibilità fornita da SQL è quella di definire in una relazione colonne derivate, il cui valore è cioè ottenuto a partire dai valori di altre colonne, e di derivare interamente il contenuto di una relazione a partire da altri dati attraverso un'interrogazione. I meccanismi di derivazione dei dati verranno discussi nel Paragrafo 3.5.

Obbligatorietà di colonne. Per la specifica dell'obbligatorietà di una colonna è sufficiente aggiungere `NOT NULL` alla specifica della colonna. In tal modo imponiamo che ogni tupla della relazione abbia necessariamente un valore diverso dal valore nullo per la colonna.

Chiavi. La specifica delle chiavi si effettua in SQL mediante le parole chiave `UNIQUE` e `PRIMARY KEY`. Come discusso nel Capitolo 2, una relazione può avere una chiave primaria (`PRIMARY KEY`) ed una o più chiavi alternative (chiavi `UNIQUE`). La parola chiave `UNIQUE` garantisce che non esistano due tuple che condividono gli stessi valori non nulli per le colonne (le colonne `UNIQUE` possono contenere valori nulli). La parola chiave `PRIMARY KEY` impone invece che per ogni tupla i valori delle colonne specificati siano non nulli e diversi da quelli di ogni altra tupla. Le colonne specificate come `PRIMARY KEY` non possono quindi assumere valori nulli per alcuna tupla della relazione. In una tabella è possibile specificare più chiavi `UNIQUE` ma una sola `PRIMARY KEY`. Per specificare una chiave composta da una sola colonna è sufficiente far seguire la specifica della colonna da `UNIQUE` o `PRIMARY KEY`, alternativamente si può far seguire la definizione della tabella dalla clausola `PRIMARY KEY (<lista nomi colonne>)` o `UNIQUE(<lista nomi colonne>)`. Questa è la sola sintassi utilizzabile nel caso di chiavi composte da più colonne.

Esempio 3.2 Le relazioni `Video` e `Noleggio` della base di dati della videoteca possono essere definite come segue, includendo la specifica di chiavi e vincoli di obbligatorietà. Anticipiamo che, come verrà discusso nel Paragrafo 3.2.2, la funzione `CURRENT_DATE` restituisce la data corrente.

```
CREATE TABLE Video
   (colloc     DECIMAL(4) PRIMARY KEY,
```

```
    titolo     VARCHAR(30) NOT NULL,
    regista    VARCHAR(20) NOT NULL,
    tipo       CHAR NOT NULL DEFAULT 'd');

CREATE TABLE Noleggio
   (colloc    DECIMAL(4),
    dataNol   DATE DEFAULT CURRENT_DATE,
    codCli    DECIMAL(4) NOT NULL,
    dataRest  DATE,
    PRIMARY KEY (colloc,dataNol),
    UNIQUE (colloc,dataRest));
```

Poiché `dataRest` può assumere valori nulli ed i valori nulli non sono distinti tra di loro (vedi Paragrafo 3.2.6), il vincolo `UNIQUE` non impedisce in realtà che esistano due noleggi correnti per lo stesso video. Per evitare questa situazione dovrà essere definita un'apposita asserzione (vedi Paragrafo 3.4). □

È importante evidenziare che una relazione SQL si differenzia dalla relazione del modello relazionale così come definita nel Capitolo 2 in quanto una relazione SQL può contenere tuple duplicate. Per questo motivo può avere senso, in SQL, definire chiavi costituite da tutte le colonne di una relazione.

Chiavi esterne. La specifica di chiavi esterne avviene mediante la clausola `FOREIGN KEY` del comando `CREATE TABLE`. Tale clausola è opzionale e ripetibile ed ha la seguente sintassi:[5]

```
FOREIGN KEY (<lista nomi colonne>)
   REFERENCES <nome relazione>
   [ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
   [ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
```

La clausola specifica:

- Una lista di una o più colonne che costituiscono una chiave esterna della relazione R (relazione *referente*) che stiamo definendo.
- La relazione *riferita* R' (specificata dalla clausola `REFERENCES`) di cui le suddette colonne sono chiave primaria.[6]
- L'azione da eseguire (specificata nella clausola `ON DELETE`) se una tupla della relazione riferita è cancellata ed esistono tuple in R che fanno riferimento a tale tupla. Le opzioni possibili sono:

[5] In realtà la clausola prevede anche una clausola opzionale `MATCH` per la specifica del *tipo di match* tra i valori degli attributi chiave e quelli degli attributi chiave esterna. Tale clausola è significativa nel caso di chiavi esterne costituite da più di un attributo ed in presenza di valori nulli e non viene considerata nella trattazione.

[6] In realtà SQL prevede la possibilità che anche chiavi `UNIQUE` possano essere riferite. Non consideriamo tale possibilità nel presente capitolo per non complicare la trattazione.

  - `NO ACTION`. La cancellazione di una tupla dalla relazione riferita è eseguita solo se non esiste alcuna tupla in R che fa riferimento a tale tupla.
  - `CASCADE`. La cancellazione della tupla della relazione riferita implica la cancellazione di tutte le tuple di R che fanno riferimento a tale tupla.
  - `SET NULL`. La cancellazione della tupla della relazione riferita implica che, in tutte le tuple di R che fanno riferimento a tale tupla, il valore della chiave esterna viene posto uguale al valore nullo.
  - `SET DEFAULT`. La cancellazione della tupla della relazione riferita implica che, in tutte le tuple di R che fanno riferimento a tale tupla, il valore della chiave esterna viene posto uguale al valore di default specificato per le colonne che costituiscono la chiave esterna.

  L'opzione di default è `NO ACTION`.

- L'azione da eseguire (specificata nella clausola `ON UPDATE`) se la chiave primaria di una tupla della relazione riferita viene modificata ed esistono tuple in R che fanno riferimento a tale tupla. Le opzioni possibili sono le stesse ed hanno lo stesso significato di quelle che abbiamo visto per la gestione della cancellazione. L'unica differenza riguarda l'opzione `CASCADE`, la quale ha l'effetto di assegnare, come valori della chiave esterna delle tuple che fanno riferimento alla tupla la cui chiave primaria è modificata, il nuovo valore della chiave primaria di quest'ultima. L'opzione di default è, anche in questo caso, `NO ACTION`.

È importante notare che la possibilità di dichiarare chiavi esterne e di poter, inoltre, specificare le azioni che il sistema deve eseguire se una tupla appartenente ad una relazione riferita è cancellata o la sua chiave modificata, permette una gestione semplice ed efficace dell'integrità referenziale. Si noti, infine, che l'integrità referenziale potrebbe essere compromessa anche da inserimenti e modifiche del valore di chiave esterna di tuple della relazione referente. In questo caso, se la modifica porta ad una violazione dell'integrità referenziale viene semplicemente rifiutata dal sistema e non è possibile specificare azioni di riparazione alternative.

Una relazione può avere più chiavi esterne; la sua definizione contiene in tal caso tante clausole `FOREIGN KEY` quante sono le chiavi esterne. Nel caso di chiave esterna costituita da un solo attributo si può far seguire la specifica della colonna da `REFERENCES` e dalla specifica delle informazioni relative alla chiave esterna.

Esempio 3.3 La definizione delle relazioni `Film`, `Video`, `Cliente` e `Noleggio` della base di dati della videoteca, con la specifica delle chiavi esterne, è la seguente.

```
CREATE TABLE Film
   (titolo   VARCHAR(30),
    regista  VARCHAR(20),
    anno     DECIMAL(4) NOT NULL,
```

```
    genere   CHAR(15) NOT NULL,
    valutaz  NUMERIC(3,2),
    PRIMARY KEY (titolo,regista));

CREATE TABLE Cliente
   (codCli    DECIMAL(4) PRIMARY KEY,
    nome      VARCHAR(20) NOT NULL,
    cognome   VARCHAR(20) NOT NULL,
    telefono  CHAR(15) NOT NULL,
    dataN     DATE NOT NULL,
    residenza VARCHAR(30) NOT NULL,
    UNIQUE (nome,cognome,dataN));

CREATE TABLE  Video
   (colloc       DECIMAL(4) PRIMARY KEY,
    titolo       VARCHAR(30) NOT NULL,
    regista      VARCHAR(20) NOT NULL,
    tipo         CHAR NOT NULL DEFAULT 'd',
    FOREIGN KEY (titolo,regista) REFERENCES Film
                                 ON DELETE RESTRICT);

CREATE TABLE Noleggio
   (colloc   DECIMAL(4) REFERENCES Video
                        ON DELETE CASCADE ON UPDATE CASCADE,
    dataNol  DATE DEFAULT CURRENT_DATE,
    codCli   DECIMAL(4) NOT NULL REFERENCES Cliente
                        ON DELETE CASCADE ON UPDATE CASCADE,
    dataRest DATE,
    PRIMARY KEY (colloc,dataNol),
    UNIQUE (colloc,dataRest));
```

Per quanto riguarda le violazioni dell'integrità referenziale:

- se viene cancellato un cliente vengono cancellati anche tutti i noleggi da lui effettuati;
- se viene cancellato un video vengono cancellati anche tutti i noleggi di tale video;
- non viene permessa la cancellazione di informazioni relative a film se vi sono video contenenti quel film;
- le modifiche di codice cliente e collocazione video vengono propagate ai relativi noleggi. □

3.1.3 Cancellazione e modifica di relazioni

La cancellazione di una relazione è eseguita tramite il comando:

```
DROP TABLE <nome relazione> {RESTRICT | CASCADE};
```

dove `<nome relazione>` è il nome della relazione da cancellare. La cancellazione di una relazione comporta la cancellazione di tutte le tuple in essa contenute. Nel richiedere la cancellazione di una relazione, è necessario specificare una tra le opzioni `RESTRICT` e `CASCADE`. Se viene specificata l'opzione `RESTRICT` la relazione viene cancellata solo se non è riferita da altri elementi dello schema della base di dati. Al contrario, se viene specificata l'opzione `CASCADE` la relazione e tutti gli elementi dello schema che la riferiscono vengono cancellati.

Esempio 3.4 Consideriamo lo schema della base di dati della videoteca definito nell'Esempio 3.3. Poiché la relazione `Film` è riferita dalla relazione `Video`, che è a sua volta riferita dalla relazione `Noleggio`, il comando:

```
DROP TABLE Film RESTRICT;
```

non altera lo schema della base di dati, mentre il comando:

```
DROP TABLE Film CASCADE;
```

causa la cancellazione delle tre relazioni. □

Uno schema di relazione può essere modificato mediante il comando `ALTER TABLE`, la cui sintassi è la seguente:

```
ALTER TABLE <nome relazione> <modifica>;
```

dove `<nome relazione>` è il nome della relazione da modificare e `<modifica>` è la specifica della modifica da effettuare. Le modifiche possibili sono:[7]

- Aggiunta di una nuova colonna, specificata come:

  ```
  ADD [COLUMN] <specifica colonna>
  ```

  dove `<specifica colonna>` è una specifica di colonna definita in accordo al formato discusso per il comando `CREATE TABLE`.

- Modifica di una colonna, specificata come:

  ```
  ALTER [COLUMN] {SET DEFAULT <valore default> | DROP DEFAULT}
  ```

  in cui `<valore default>` è il nuovo valore di default per la colonna. Tale modifica permette di aggiungere o modificare il valore di default per la colonna (mediante `SET DEFAULT`) oppure di rimuovere il valore di default (mediante `DROP DEFAULT`).

[7] Come vedremo nel Paragrafo 3.4, in realtà tra le modifiche possibili vi sono anche `ADD CONSTRAINT` e `DROP CONSTRAINT` per l'aggiunta e la rimozione di un vincolo, rispettivamente.

- Eliminazione di una colonna, specificata come:

  `DROP [COLUMN] <nome colonna> {RESTRICT | CASCADE}`

  dove `<nome colonna>` è il nome della colonna da eliminare e `RESTRICT` e `CASCADE` hanno il significato discusso per il comando `DROP TABLE`.

Esempio 3.5 Consideriamo nuovamente lo schema della base di dati della videoteca definito nell'Esempio 3.3. Il comando:

```
ALTER TABLE Film ADD COLUMN studio VARCHAR(20);
```

ha l'effetto di aggiungere come sesta ed ultima colonna della relazione `Film` la colonna `studio`. Poiché non viene specificato un valore di default, a tutte le tuple di `Film` viene assegnato il valore `NULL` per tale colonna. Il comando:

```
ALTER TABLE Video ALTER COLUMN tipo SET DEFAULT 'v';
```

ha l'effetto di modificare il valore di default per la colonna `tipo` di `Video`, ponendolo uguale al valore `'v'`. □

3.2 Interrogazioni

Questo paragrafo descrive vari aspetti relativi alla specifica di interrogazioni in SQL. Per prima cosa introdurremo il formato di base di un'interrogazione SQL ed i principali operatori e funzioni utilizzabili nelle interrogazioni. Verranno poi discusse ulteriori caratteristiche del linguaggio di interrogazione, quali ordinamento, operazione di join, funzioni di gruppo, valori nulli e sotto-interrogazioni.

3.2.1 Formato di base del comando `SELECT`

Le interrogazioni in SQL sono espresse tramite il comando `SELECT`. La forma base di un'interrogazione SQL ha la seguente struttura:[8]

`SELECT` {`DISTINCT` $R_{i_1}.C_1, R_{i_2}.C_2, \ldots, R_{i_n}.C_n$ | `*`}
`FROM` $R_1, R_2, \ldots, R_k$ `WHERE` F`;`

dove:

- R_i $(i = 1, \ldots, k)$ è un nome di relazione. La clausola `FROM` specifica pertanto le relazioni oggetto dell'interrogazione.

[8] Come discuteremo nel seguito del paragrafo, la clausola `DISTINCT` del comando `SELECT` è in realtà opzionale.