- 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.

- $R_{i_j}.C_j$ $(j = 1, \ldots, n,\ i_j \in \{1, \ldots, k\})$ indica che l'interrogazione deve restituire la colonna C_j della relazione R_{i_j}. La relazione R_{i_j} deve essere una delle relazioni specificate nella clausola `FROM` dell'interrogazione. Se la colonna C_j appare in una sola tra le relazioni specificate nella clausola `FROM`, allora possiamo semplicemente usare la notazione C_j, invece di $R_{i_j}.C_j$. La lista $R_{i_1}.C_1, R_{i_2}.C_2, \ldots R_{i_n}.C_n$ costituisce la *clausola di proiezione* dell'interrogazione. Il simbolo '`*`' indica che l'interrogazione deve restituire tutte le colonne delle relazioni che compaiono nella clausola `FROM`, non viene quindi effettuata alcuna proiezione.

- F è la *clausola di qualificazione* dell'interrogazione in quanto qualifica, tramite predicati, le tuple di interesse. Tale clausola consiste in una combinazione booleana di predicati: in F possono essere usati i connettivi logici `AND`, `OR` e `NOT`. Le tuple che soddisfano F *verificano* l'interrogazione. Da tali tuple vengono estratte le colonne specificate nella clausola di proiezione dell'interrogazione.

Il significato di un'interrogazione SQL, specificata in accordo al formato precedente, è rappresentato tramite la seguente espressione dell'algebra relazionale:[9]

$$\Pi_{R_{i_1}.C_1, R_{i_2}.C_2, \ldots, R_{i_n}.C_n}(\sigma_F(R_1 \times \ldots \times R_k))$$

nel caso in cui la clausola di proiezione contenga una lista di colonne e tramite l'espressione:

$$\sigma_F(R_1 \times \ldots \times R_k)$$

nel caso in cui la clausola di proiezione contenga '`*`'. La valutazione dell'interrogazione inizia quindi dalla clausola `FROM`, viene poi applicata la clausola di qualificazione ed infine la clausola di proiezione.

L'ordine in cui le colonne sono elencate nella clausola di proiezione determina l'ordine da sinistra a destra secondo cui le colonne appaiono nella relazione risultato. Quando si usa '`*`', l'ordine in cui le colonne appaiono nel risultato è dato dall'ordine in cui le relazioni compaiono nella clausola `FROM` e, per ogni relazione, dall'ordine specificato nella definizione della relazione (cioè nel comando `CREATE TABLE`).

La clausola di qualificazione può contenere i connettivi booleani `AND`, `OR` e `NOT` con l'usuale precedenza. Ordini di precedenza diversi tra questi connettivi possono essere specificati utilizzando le parentesi. I predicati semplici in SQL hanno in generale la forma $e\ op\ e'$, dove e ed e' sono espressioni che denotano valori mentre op è un operatore relazionale di confronto. Molto spesso un'espressione che denota un valore è semplicemente un nome di colonna; in tal caso il valore denotato è il valore della colonna per la tupla considerata. È tuttavia possibile formulare predicati in cui i valori sono specificati tramite espressioni, uso di funzioni e sotto-interrogazioni, discussi nel prosieguo del paragrafo. È ovviamente possibile

[9] Notiamo che gli schemi di $R_1, \ldots, R_k$ sono resi disgiunti premettendo ad ogni nome di colonna il nome della relazione cui appartiene.

specificare predicati della forma C *op* C', dove C e C' sono nomi di colonne; in questo caso il predicato esprime un confronto tra i valori di due colonne. Anche nella clausola di qualificazione al nome della colonna può (e deve, in caso di colonna appartenente a più relazioni nella clausola `FROM`) essere premesso il nome della relazione cui la colonna appartiene, con la notazione $R.C$. Oltre agli operatori di confronto già visti per l'algebra relazionale, SQL offre numerosi altri operatori di confronto, alcuni dei quali verranno discussi nel Paragrafo 3.2.2.

L'esempio successivo, così come gli altri presentati in questo capitolo, è basato sulle relazioni `Film`, `Video`, `Cliente` e `Noleggio` illustrate nelle Figure 2.1 e 2.2.

Esempio 3.6 I seguenti sono esempi di interrogazioni espresse in SQL. Per ogni interrogazione, riportiamo anche la traduzione in algebra relazionale.

Q1 Selezionare i film girati prima del 1999:

```
SELECT * FROM Film WHERE anno < 1999;
```

Traduzione in algebra relazionale: $\sigma_{\texttt{anno}<1999}(\texttt{Film})$.

Q2 Selezionare il titolo ed il regista dei film di fantascienza girati prima del 1999:

```
SELECT DISTINCT titolo, regista FROM Film
WHERE anno < 1999 AND genere = 'fantascienza';
```

Traduzione in algebra relazionale:
$\Pi_{\texttt{titolo,regista}}(\sigma_{\texttt{anno}<1999 \wedge \texttt{genere}='\texttt{fantascienza}'}(\texttt{Film}))$.

Q3 Selezionare il titolo dei film di Tim Burton di genere horror o fantascienza:

```
SELECT DISTINCT titolo FROM Film WHERE regista = 'tim burton'
AND (genere= 'horror' OR genere = 'fantascienza');
```

Traduzione in algebra relazionale:
$\Pi_{\texttt{titolo}}(\sigma_{\texttt{regista}='\texttt{tim burton}' \wedge (\texttt{genere}='\texttt{horror}' \vee \texttt{genere}='\texttt{fantascienza}')}(\texttt{Film}))$.

La Figura 3.1 riporta i risultati delle interrogazioni precedenti. □

La clausola `DISTINCT` del comando `SELECT` è in realtà opzionale. La sua inclusione ha l'effetto di richiedere l'eliminazione dei duplicati dal risultato. È importante notare che, se tale clausola è omessa, eventuali duplicati nel risultato di un'interrogazione non vengono automaticamente eliminati, perdendo quindi la corrispondenza con l'operazione di proiezione dell'algebra relazionale.

Esempio 3.7 Consideriamo l'interrogazione per determinare i generi dei film. Con la formulazione:

```
SELECT DISTINCT genere FROM Film;
```

titolo	regista	anno	genere	valutaz
underground	emir kusturica	1995	drammatico	3.20
edward mani di forbice	tim burton	1990	fantastico	3.60
nightmare before christmas	tim burton	1993	animazione	4.00
ed wood	tim burton	1994	drammatico	4.00
mars attacks	tim burton	1996	fantascienza	3.00
nirvana	gabriele salvatores	1997	fantascienza	3.00
mediterraneo	gabriele salvatores	1991	commedia	3.80
pulp fiction	quentin tarantino	1994	thriller	3.50
le iene	quentin tarantino	1992	thriller	4.00

titolo	regista
mars attacks	tim burton
nirvana	gabriele salvatores

titolo
mars attacks
il mistero di sleepy hollow

Figura 3.1: Risultati delle interrogazioni dell'Esempio 3.6

ogni genere compare nel risultato un'unica volta, viceversa con la formulazione:

```
SELECT genere FROM Film;
```

il risultato può contenere duplicati. La Figura 3.2 riporta i risultati delle interrogazioni precedenti. □

Rimarchiamo infine che il risultato di un'interrogazione, applicato ad una o più relazioni, è sempre una relazione (proprietà di chiusura). Questa proprietà permette di usare interrogazioni all'interno di altre interrogazioni, come vedremo nel Paragrafo 3.2.7.

3.2.2 Operatori e funzioni

Come discusso in precedenza, SQL fornisce numerosi operatori di confronto e funzioni predefinite, che permettono di specificare un vasto insieme di espressioni e predicati. Nel seguito discuteremo brevemente i più comuni.

3.2.2.1 Operatori di confronto

SQL fornisce operatori aggiuntivi, oltre agli usuali operatori relazionali di confronto già visti nel contesto dell'algebra relazionale. Alcuni di questi operatori sono in effetti ridondanti e sono stati introdotti per rendere più agevole l'uso del linguaggio. Altri operatori sono invece introdotti per fornire maggiori funzionalità nel trattare alcuni tipi di dato (come ad esempio le stringhe). Alcuni di questi operatori sono illustrati nel seguito. È da notare, tuttavia, che il linguaggio SQL è estremamente ricco e fornisce molti altri operatori, solo alcuni dei quali sono discussi nella trattazione successiva.

genere
drammatico
fantastico
animazione
fantascienza
horror
commedia
thriller

genere
drammatico
fantastico
animazione
drammatico
fantascienza
horror
fantastico
animazione
fantastico
drammatico
fantascienza
commedia
thriller
thriller

Figura 3.2: Risultati delle interrogazioni dell'Esempio 3.7

Condizioni su intervalli di valori. L'operatore `BETWEEN` permette di ritrovare le tuple che contengono valori di una colonna in un intervallo specificato. Un predicato di confronto con l'operatore `BETWEEN` ha il formato e `BETWEEN` v_1 `AND` v_2 dove: e è un'espressione che denota un valore, nel caso più semplice un nome di colonna;[10] v_1 e v_2 sono valori, compatibili con il tipo di e, detti rispettivamente estremo inferiore e superiore dell'intervallo. L'espressione e `BETWEEN` v_1 `AND` v_2 è semplicemente un'abbreviazione per e `>=` v_1 `AND` e `<=` v_2. L'operatore `BETWEEN` ha anche una forma negata; un predicato con la forma negata di tale operatore ha il formato e `NOT BETWEEN` v_1 `AND` v_2.

Esempio 3.8 Per determinare tutte le informazioni relative ai film girati tra il 1995 e il 2000 formuliamo la seguente interrogazione:

```
SELECT * FROM Film WHERE anno BETWEEN 1995 AND 2000;
```

il cui risultato è illustrato nella Figura 3.3. □

Ricerca di valori in un insieme. L'operatore `IN` permette di determinare le tuple che contengono uno tra i valori di un insieme specificato. Un predicato di confronto con l'operatore `IN` ha il formato e `IN` $(v_1, \ldots, v_n)$ dove: e è un'espressione che denota un valore, mentre $v_1, \ldots, v_n$ sono valori il cui tipo è compatibile con il tipo di e. L'espressione e `IN` $(v_1, \ldots, v_n)$ è semplicemente un'abbreviazione per e `=` v_1 `OR` $\ldots$ `OR` e `=` v_n. Anche per l'operatore `IN` esiste la forma negata, che ha il formato e `NOT IN` $(v_1, \ldots, v_n)$. L'operatore `IN` è particolarmente utile non tanto quando l'insieme di valori è esplicitamente enumerato, bensì quando è ottenuto tramite un'interrogazione alla base di dati. Esempi più significativi saranno pertanto discussi nella trattazione relativa alle sotto-interrogazioni.

[10] È da notare, per questo predicato come per i successivi discussi in questo paragrafo, che e potrebbe anche essere una sotto-interrogazione (vedi Paragrafo 3.2.7).

titolo	regista	anno	genere	valutaz
underground	emir kusturica	1995	drammatico	3.20
mars attacks	tim burton	1996	fantascienza	3.00
il mistero di sleepy hollow	tim burton	1999	horror	3.50
nirvana	gabriele salvatores	1997	fantascienza	3.00

titolo	regista	anno	genere	valutaz
mars attacks	tim burton	1996	fantascienza	3.00
il mistero di sleepy hollow	tim burton	1999	horror	3.50
nirvana	gabriele salvatores	1997	fantascienza	3.00

titolo	regista	anno	genere	valutaz
underground	emir kusturica	1995	drammatico	3.20
mediterraneo	gabriele salvatores	1991	commedia	3.80

Figura 3.3: Risultati delle interrogazioni degli Esempi 3.8, 3.9 e 3.10

Esempio 3.9 Per determinare tutte le informazioni relative ai film il cui genere è horror o fantascienza formuliamo la seguente interrogazione:

```
SELECT * FROM Film WHERE genere IN ('horror','fantascienza');
```

il cui risultato è illustrato nella Figura 3.3. □

Condizioni di confronto per stringhe di caratteri. L'operatore `LIKE` permette di eseguire alcune semplici operazioni di *pattern matching* su colonne di tipo stringa. Un predicato di confronto espresso con l'operatore `LIKE` ha il formato: *e* `LIKE` *pattern*, dove *e* è un'espressione che denota un valore di tipo stringa e *pattern* è una stringa di caratteri che può contenere i caratteri speciali '`%`' e '`_`'. Il simbolo '`%`' denota una sequenza di caratteri arbitrari di lunghezza qualsiasi (anche zero), mentre il simbolo '`_`' indica esattamente un carattere.

Esempio 3.10 Per determinare tutte le informazioni relative ai film che hanno il carattere '`d`' come terza lettera del titolo, formuliamo la seguente interrogazione:

```
SELECT * FROM Film WHERE titolo LIKE '__d%';
```

il cui risultato è illustrato nella Figura 3.3. □

3.2.2.2 Espressioni e funzioni

Le espressioni sono di solito formulate applicando funzioni (aritmetiche, su stringhe, su date e tempi) ai valori delle colonne delle tuple. Oltre a poter essere usate nei predicati, le espressioni possono comparire nella clausola di proiezione delle interrogazioni, nelle espressioni di assegnamento usate nel comando `UPDATE` (vedi Paragrafo 3.3) e nella specifica di colonne derivate (vedi Paragrafo 3.5).

Un'espressione usata nella clausola di proiezione dà luogo ad una relazione con una colonna non presente nella relazione su cui è effettuata l'interrogazione. Tale colonna è detta *virtuale* in quanto è derivata dalle colonne, dette *di base*, della relazione oggetto dell'interrogazione. Una colonna virtuale non è fisicamente memorizzata nella relazione, ma è derivata dinamicamente da colonne della relazione. Ad una colonna virtuale, derivata tramite un'espressione, viene assegnato nell'eventuale presentazione del risultato un nome di colonna dato dall'espressione che definisce la colonna. È possibile, tuttavia, specificare nomi alternativi per tali colonne, facendo seguire l'espressione dalla clausola `AS <nome colonna>`. Poiché tuttavia la clausola di proiezione è l'ultima clausola dell'interrogazione ad essere valutata, tali nomi non possono essere utilizzati nelle altre clausole dell'interrogazione.

Espressioni e funzioni aritmetiche. Oltre agli usuali operatori aritmetici comunemente inclusi ($+$, $-$, $*$, $/$) sono previste le seguenti funzioni:

- la funzione `ABS(`n`)` che ha come argomento un valore numerico n e ne calcola il valore assoluto;
- la funzione `MOD(`n`,`b`)` che ha come argomenti due valori interi n e b, e calcola il resto intero della divisione di n per b.

Sono inoltre previste funzioni logaritmiche, esponenziali, per il calcolo della radice quadrata, dell'elevamento a potenza, della parte intera superiore ed inferiore.

Espressioni e funzioni per stringhe. Il principale operatore tra stringhe è l'operatore di concatenazione di stringhe, denotato da '||' . Sono inoltre previste alcune funzioni per la manipolazione di strighe, tra cui citiamo:

- la funzione `LENGTH(`*str*`)` che ha come argomento una stringa *str* e ne calcola la lunghezza, intesa come numero di caratteri;
- le funzioni `UPPER(`*str*`)` e `LOWER(`*str*`)` che trasformano la stringa *str* in caratteri tutti maiuscoli o tutti minuscoli, rispettivamente;
- la funzione `SUBSTR(`*str*`,`m`[,`n`])` che ha come argomento una stringa *str* e due interi m ed n, ed estrae da *str* la sotto-stringa dal carattere di posizione m fino all'ultimo carattere o per una lunghezza n (se l'argomento n è specificato);
- la funzione `TRIM [`str_1`] FROM` str_2 che elimina dalla stringa str_2 i caratteri in str_1 (se str_1 non è specificata elimina gli spazi).

Espressioni e funzioni per date e tempi. Data l'importanza delle informazioni temporali nelle più svariate applicazioni, SQL, oltre a fornire vari tipi di dato temporali, fornisce numerose funzioni per la manipolazione di date e tempi. Le funzioni più rilevanti sono:

- Le funzioni zerarie `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP` che restituiscono, rispettivamente, la data, il tempo ed il timestamp attuale.

- `EXTRACT (`q `FROM` e`)` estrae il campo corrispondente al qualificatore temporale q dall'espressione e. `EXTRACT (DAY FROM DATE '08-Ott-1969')`, ad esempio, restituisce `8`.

Date e tempi possono essere usati in espressioni aritmetiche, per calcolare, tra gli altri, la differenza tra date, che è un intervallo temporale, come illustrato dagli esempi seguenti. In tali espressioni è inoltre possibile usare diverse *unità temporali*, tra cui: `YEAR[S]`, `MONTH[S]`, `DAY[S]`, `HOUR[S]`, `MINUTE[S]`, `SECOND[S]`.

Nell'esempio seguente la clausola di proiezione contiene un'espressione.

Esempio 3.11 Per ritrovare la collocazione del video noleggiato e la durata (in giorni) di ogni noleggio del cliente di codice 6635 formuliamo la seguente interrogazione:

```
SELECT colloc, (dataRest - dataNol) DAY
FROM Noleggio WHERE codCli = 6635;
```

Notiamo che l'espressione `(dataRest - dataNol)` restituisce un valore di tipo intervallo e SQL richiede di specificare il qualificatore rispetto a cui esprimerlo, anche se in molte implementazioni questo non è in realtà necessario e l'interrogazione viene accettata anche senza l'indicazione di `DAY`. Il risultato dell'interrogazione è illustrato nella Figura 3.4. Notiamo che per i noleggi in corso la durata assume valore nullo. Come discuteremo in dettaglio nel Paragrafo 3.2.6, infatti, un'espressione in cui un argomento è nullo assume valore nullo. □

Le espressioni possono essere utilizzate sia nella clausola di proiezione sia nella clausola di qualificazione, come illustrato dal seguente esempio.

Esempio 3.12 Vogliamo ora ritrovare la collocazione del video noleggiato e la durata (in giorni) di ogni noleggio di durata superiore a due giorni del cliente di codice 6635:

```
SELECT colloc, (dataRest - dataNol) DAY AS durata
FROM Noleggio
WHERE codCli = 6635 AND
     (dataRest - dataNol) DAY > INTERVAL '2' DAY;
```

colloc	(dataRest - dataNol) DAY
1111	1
1115	1
1117	4
1118	4
1119	2
1120	2
1121	3
1122	3
1113	3
1129	5
1127	?
1125	?

colloc	durata
1117	4
1118	4
1121	3
1122	3
1113	3
1129	5

Figura 3.4: Risultati delle interrogazioni degli Esempi 3.11 e 3.12

Nell'interrogazione viene specificato il nome `durata` per la colonna virtuale. L'espressione `(dataRest - dataNol) DAY` restituisce un intervallo, la cui durata viene confrontata con quella dell'intervallo `INTERVAL '2' DAY`, cioè due giorni. Il risultato dell'interrogazione è illustrato nella Figura 3.4. Notiamo che tale risultato non contiene le tuple relative ai noleggi in corso. Come discuteremo nel Paragrafo 3.2.6, infatti, un'espressione che assume valore nullo non soddisfa predicati di confronto con valori e non viene quindi restituita dall'interrogazione. □

Molto utile è infine la funzione di `CAST`, la cui sintassi è `CAST` e `AS` T, che permette di convertire il tipo del valore restituito dall'espressione e al tipo specificato T. L'applicazione della funzione è soggetta ad un certo numero di restrizioni, che garantiscono che la conversione sia effettivamente possibile.

3.2.3 Ordinamento del risultato di un'interrogazione

Negli esempi visti finora, l'ordine delle tuple risultato di una interrogazione è determinato dal sistema. Di solito quest'ordine dipende, oltre che dalla memorizzazione fisica delle tuple, dalla strategia scelta dal DBMS per eseguire l'interrogazione (vedi Capitolo 7). È ovviamente importante per l'applicazione poter richiedere che le tuple risultato di un'interrogazione siano ordinate in base al valore di una o più colonne. Il comando `SELECT` prevede a tale scopo la clausola addizionale `ORDER BY`, illustrata dall'esempio seguente.

Esempio 3.13 Vogliamo elencare la collocazione del video, la data di noleggio e la data di restituzione di tutti i noleggi del cliente 6635, ordinando le tuple in ordine crescente in base alla data di inizio del noleggio:

```
SELECT colloc, dataNol, dataRest FROM Noleggio
WHERE codCli = 6635
ORDER BY dataNol;
```

colloc	dataNol	dataRest
1111	01-Mar-2006	02-Mar-2006
1115	01-Mar-2006	02-Mar-2006
1117	02-Mar-2006	06-Mar-2006
1118	02-Mar-2006	06-Mar-2006
1119	08-Mar-2006	10-Mar-2006
1120	08-Mar-2006	10-Mar-2006
1121	15-Mar-2006	18-Mar-2006
1122	15-Mar-2006	18-Mar-2006
1113	15-Mar-2006	18-Mar-2006
1129	15-Mar-2006	20-Mar-2006
1127	22-Mar-2006	?
1125	22-Mar-2006	?

colloc	dataNol	dataRest
1125	22-Mar-2006	?
1127	22-Mar-2006	?
1113	15-Mar-2006	18-Mar-2006
1121	15-Mar-2006	18-Mar-2006
1122	15-Mar-2006	18-Mar-2006
1129	15-Mar-2006	20-Mar-2006
1119	08-Mar-2006	10-Mar-2006
1120	08-Mar-2006	10-Mar-2006
1117	02-Mar-2006	06-Mar-2006
1118	02-Mar-2006	06-Mar-2006
1111	01-Mar-2006	02-Mar-2006
1115	01-Mar-2006	02-Mar-2006

Figura 3.5: Risultati delle interrogazioni degli Esempi 3.13 e 3.14

Il risultato dell'interrogazione è illustrato nella Figura 3.5. □

L'ordinamento non è limitato ad una sola colonna, né ad un ordine crescente. Se più colonne sono specificate, le tuple sono ordinate prima in base ai valori della prima colonna specificata nella clausola `ORDER BY`, poi in base alla seconda colonna e così via per tutte le colonne. L'opzione `DESC` associata ad una colonna specifica l'ordinamento in base a valori decrescenti. L'altra opzione è l'opzione `ASC`, che è quella di default.

Esempio 3.14 Vogliamo elencare la collocazione del video, la data di noleggio e la data di restituzione di tutti i noleggi del cliente 6635, ordinando le tuple in base alla data di inizio del noleggio in ordine decrescente ed alla collocazione in ordine crescente. Formuliamo l'interrogazione:

```
SELECT colloc, dataNol, dataRest FROM Noleggio
WHERE codCli = 6635
ORDER BY dataNol DESC, colloc;
```

il cui risultato è illustrato nella Figura 3.5. □

3.2.4 Operazione di join

Come abbiamo visto nel Capitolo 2, l'operazione di join rappresenta un'importante operazione in quanto permette di correlare dati memorizzati in relazioni diverse, quindi di "attraversare" le associazioni esistenti tra i dati. Tradizionalmente il join è espresso in SQL tramite un prodotto cartesiano a cui sono applicati uno o più *predicati di join*. I predicati di join in SQL sono del tutto simili ai predicati di selezione visti nella clausola di qualificazione ed esprimono la relazione che deve essere verificata dalle tuple risultato dell'interrogazione. SQL prevede però anche

diverse operazioni esplicite di join, che consentono formulazioni alternative delle interrogazioni. Tali operatori, poiché producono relazioni, vengono utilizzati nella clausola `FROM` di un'interrogazione, ed includono:

- `CROSS JOIN`, che è la forma di operatore di join più semplice e corrisponde al prodotto cartesiano. Ha sintassi `<nome relazione> CROSS JOIN <nome relazione>` ed equivale all'uso di `<nome relazione>, <nome relazione>` nella clausola `FROM`.
- `JOIN ON`, che corrisponde al theta-join ed ha sintassi `<nome relazione> JOIN <nome relazione> ON <predicato>`, con `<predicato>` predicato di join arbitrario.
- `JOIN USING`, in cui viene richiesta l'uguaglianza dei valori delle colonne specificate nella clausola `USING`, la cui sintassi è `<nome relazione> JOIN <nome relazione> USING (<lista nomi colonne>)`.
- `NATURAL JOIN`, che corrisponde al join naturale, in cui viene richiesta l'uguaglianza dei valori delle colonne che hanno lo stesso nome nelle due relazioni. La sintassi è `<nome relazione> NATURAL JOIN <nome relazione>`. L'operazione di `NATURAL JOIN` non corrisponde però completamente al join naturale algebrico (vedi Capitolo 2): non viene eseguita alcuna proiezione e lo schema risultante è quello del prodotto cartesiano.

Esempio 3.15 Le seguenti sono tutte formulazioni alternative dell'interrogazione per ritrovare i titoli dei video noleggiati il 15 Marzo 2006 dal cliente con codice 6635:

```
SELECT titolo FROM Video, Noleggio
WHERE codCli = 6635 AND dataNol = DATE '15-Mar-2006'
      AND Video.colloc = Noleggio.colloc;

SELECT titolo FROM Video CROSS JOIN Noleggio
WHERE codCli = 6635 AND dataNol = DATE '15-Mar-2006'
      AND Video.colloc = Noleggio.colloc;

SELECT titolo
FROM Video JOIN Noleggio
       ON Video.colloc = Noleggio.colloc
WHERE codCli = 6635 AND dataNol = DATE '15-Mar-2006';

SELECT titolo FROM Video JOIN Noleggio USING (colloc)
WHERE codCli = 6635 AND dataNol = DATE '15-Mar-2006';

SELECT titolo FROM Video NATURAL JOIN Noleggio
WHERE codCli = 6635 AND dataNol = DATE '15-Mar-2006';
```

Il predicato di join è `Video.colloc = Noleggio.colloc`. Il risultato dell'interrogazione è illustrato nella Figura 3.6. □

Dato che il risultato di un'operazione di join è una relazione, è possibile richiedere che tale relazione sia ordinata, anche in base a valori di colonne di relazioni diverse, o che eventuali duplicati siano eliminati.

Outer join. In R `JOIN` S non si ha traccia delle tuple di R che non corrispondono ad alcuna tupla di S. Questo non è sempre quello che si desidera. Per questo motivo SQL prevede un operatore di `OUTER JOIN` che aggiunge al risultato le tuple di R e/o S che non hanno partecipato al join, completandole con `NULL`. L'operatore di join originario, per contrasto, viene anche detto `INNER JOIN`. La variante `OUTER` può essere utilizzata sia per il join naturale sia per il theta-join.

Esistono diverse varianti dell'outer join, che vengono specificate premettendo il corrispondente qualificatore all'operatore `OUTER JOIN`. Consideriamo R `OUTER JOIN` S:

- `FULL`. Sia le tuple di R sia quelle di S che non partecipano al join vengono completate ed inserite nel risultato.
- `LEFT`. Le tuple di R che non partecipano al join vengono completate ed inserite nel risultato.
- `RIGHT`. Le tuple di S che non partecipano al join vengono completate ed inserite nel risultato.

Esempio 3.16 Supponiamo di voler visualizzare per ogni video contenente un film di Tim Burton di genere fantastico la sua collocazione, il titolo ed i codici dei clienti che l'hanno eventualmente noleggiato:

```
SELECT colloc, titolo, codCli
FROM Film NATURAL JOIN Video NATURAL LEFT OUTER JOIN Noleggio
WHERE regista = 'tim burton' AND genere = 'fantastico';
```

Senza l'utilizzo dell'outer join i video, quali il 1123, che non sono mai stati noleggiati non avrebbero fatto parte del risultato. Il risultato dell'interrogazione è illustrato nella Figura 3.6. □

3.2.5 Funzioni di gruppo e raggruppamento

Un'importante funzionalità che SQL aggiunge all'algebra relazionale è quella di poter estrarre dalle tuple di una relazione informazioni riassuntive, ottenute aggregando opportunamente i valori presenti in diverse tuple. I due principali costrutti che forniscono questa funzionalità sono le funzioni di gruppo e l'operatore di raggruppamento.