lezione 23

Reperimento dei dati: SELECT

Per estrarre dei dati dal database, il linguaggio SQL prevede il comando **SELECT**. Estrarre dati è sinonimo di effettuare una **query** o **interrogazione** sulla base di dati. Il risultato è sempre una tabella. La sintassi del comando **SELECT** è molto complessa. Vediamo una sua forma semplificata:

```
SELECT(DISTINCT) <Attributo1>, <Attributo2>, ..., <AttributoN>
FROM <TABELLA1>, <TABELLA2>, ..., <TABELLAK>
(WHERE <Condizione>);
```

Select restituisce una tabella formata dagli attributi: <Attributo1>, <Attributo2>, ..., <AttributoN> del prodotto delle tabelle: <TABELLA1>, <TABELLA2>, ..., <TABELLAK>, ristretto alle righe che soddisfano la <Condizione>.
Precisiamo che:

- se è assente la clausola **WHERE**, la condizione si assume sempre vera;
- se è presente l'opzione **DISTINCT**, il risultato è fornito privo di righe duplicate.

Se si vogliono visualizzare tutti gli attributi presenti nel prodotto delle tabelle, è possibile utilizzare il simbolo "*", il cui significato sarà: "tutti gli attributi del prodotto delle tabelle". La <Condizione>, inoltre, può essere composta da più condizioni semplici combinate con gli operatori logici **AND**, **NOT**, **OR**. Vediamo alcuni semplici esempi. Per visualizzare il titolo di tutti i film scriveremo:

```
SELECT Titolo
FROM FILM;
```

Supponendo che possano esistere film con lo stesso titolo, potremmo evitare di scrivere righe duplicate nel seguente modo:

```
SELECT DISTINCT Titolo
FROM FILM;
```

Per visualizzare tutti gli attributi dei film scriveremo:

```
SELECT *
FROM FILM;
```

Alias e calcoli

La tabella risultato di un'operazione SELECT ha (per default), come intestazione delle colonne, il nome degli attributi della tabella interrogata. Se si vuole assegnare un diverso nome a ogni colonna del risultato, cioè se si vuole assegnare un **alias**, si deve utilizzare la clausola **AS**.
Per visualizzare il nome, la città e i posti presenti in un cinema, utilizzando l'intestazione "Numero posti" per l'ultima colonna, scriveremo:

```
SELECT Nome, Città, Posti AS "Numero posti"
FROM CINEMA;
```

È possibile eseguire con il comando SELECT il calcolo di un'espressione sugli attributi. Il risultato viene visualizzato a video in una nuova colonna intestata con la clausola AS. Il calcolo viene eseguito esternamente alla tabella, quindi senza modificare i dati in essa contenuti.

Se vogliamo detrarre una quota del 20% dall'incasso di tutte le proiezioni:

```
SELECT Incasso * 0,8 AS "Incasso netto"
FROM PROGRAMMATO;
```

stop

Spesso, è utile utilizzare delle abbreviazioni per fare riferimento ai nomi delle tabelle. Per esempio, l'interrogazione vista poco fa:

```
SELECT Titolo
FROM FILM;
```

che consentiva di visualizzare il titolo di tutti i film, potrebbe essere riscritta nel seguente modo:

```
SELECT FILM.Titolo
FROM FILM;
```

o meglio:

```
SELECT F.Titolo
FROM FILM F;
```

dove F è l'abbreviazione usata per la tabella FILM. Questo modo di procedere, risulta particolarmente utile, quando si ha a che fare con query che coinvolgono tabelle con campi dello stesso nome. Per questo motivo, all'interno di questa unità di apprendimento utilizzeremo entrambe le forme, privilegiando quella con le abbreviazioni, quando analizzeremo database caratterizzati dalla presenza di attributi con lo stesso nome.

Il valore NULL

SQL prevede il valore **NULL**, che viene utilizzato per indicare diverse situazioni. Un esempio è quando il valore esiste ma è sconosciuto, oppure quando il valore non esiste (pensate all'importo dello stipendio del vostro Sindaco e allo stipendio di un disoccupato). Il valore NULL assume un ruolo significativo nel risultato di un'espressione logica o aritmetica. Ci sono alcune regole fondamentali da ricordare:

1. il valore NULL è diverso da zero (per i dati numerici) e dalla stringa vuota (" ") per i dati alfanumerici;
2. il risultato di un'espressione aritmetica è sconosciuto (**UNKNOWN**) se un operando ha valore NULL;
3. il confronto tra un valore NULL e un qualsiasi altro valore, compreso NULL, produce sempre UNKNOWN;
4. il valore NULL, che può essere un valore di un attributo, non è una costante, quindi non può apparire in un'espressione;
5. se il risultato del predicato della clausola WHERE è il valore UNKNOWN, la t-upla non viene considerata;
6. nelle funzioni di aggregazione, in generale, le righe con valore NULL dell'attributo considerato sono ignorate;
7. il valore UNKNOWN è un valore di verità come TRUE e FALSE.

Nelle interrogazioni è molto utile controllare se il valore di un attributo è presente, oppure è uguale a NULL. Possiamo effettuare questa verifica ricorrendo ai predicati **IS NULL** e **IS NOT NULL** nelle condizioni della clausola WHERE.
Per elencare tutti i clienti che non hanno numero di telefono scriveremo:

```
SELECT CodCli, Cognome, Nome
FROM CLIENTI
WHERE Telefono IS NULL;
```

Attenzione quindi:

```
WHERE Telefono = NULL
```

è errata nella sintassi, mentre:

```
WHERE Telefono IS NULL
```

è corretta. Questo perché NULL è considerato un particolare valore che indica "valore mancante".

lezione 24

Le operazioni relazionali in SQL

Le operazioni di **restrizione**, **proiezione** e **giunzione**, su una base dati relazionale, vengono realizzate attraverso il comando SELECT, secondo le diverse forme consentite dalla sintassi.
La restrizione e la proiezione sono già state utilizzate negli esempi di uso del comando SELECT, riportati nella lezione precedente. Rivediamole in dettaglio, assieme alle altre già elencate.

L'operazione di proiezione

L'operazione di **proiezione**, che permette di ottenere una relazione contenente solo alcuni attributi della relazione di partenza, si realizza indicando accanto alla parola SELECT l'elenco degli attributi richiesti.
Volendo ottenere l'elenco degli attori e visualizzarne soltanto il cognome e il nome, si deve effettuare una proiezione sulla tabella *Attore*, estraendo oltanto le colonne corrispondenti. Nell'algebra relazionale scriviamo:

$\pi_{Cognome, Nome}$ (ATTORE)

In SQL la precedente interrogazione viene riscritta:

```
SELECT Cognome, Nome
FROM ATTORE;
```

L'operazione di restrizione

L'operazione di **restrizione**, che consente di ricavare da una relazione un'altra relazione, contenente solo le righe che soddisfano una certa condizione, viene realizzata utilizzando la clausola WHERE nel comando SELECT.
Per ottenere l'elenco di tutti gli attori di sesso maschile si opera una restrizione sulla tabella *Attore*. Nell'algebra relazionale scriviamo:

$\delta_{Sesso = 'M'}$ (ATTORE)

In SQL la precedente interrogazione viene riscritta:

```
SELECT *
FROM ATTORE
WHERE Sesso = 'M';
```

L'operazione di giunzione (join)

In SQL è possibile utilizzare i vari tipi di join:

- CROSS JOIN (prodotto di relazioni);
- INNER JOIN (join interno – equi join);
- OUTER JOIN (join esterno);
  - LEFT JOIN (outer left join – join esterno sinistro);
  - RIGHT JOIN (outer right join – join esterno destro);
- SELF JOIN (join su un'unica tabella).

Analizziamo le differenze tra questi tipi di join utilizzando l'esempio degli studenti e delle classi. Consideriamo il seguente schema ER:

Traducendo questo schema nel modello logico, avremo due tabelle: *Studente* e *Classe*. Poiché l'associazione è di tipo N:1, all'interno della tabella *Studente* avremo il campo *NomeClasse*, che realizza l'associazione tra uno studente e la corrispondente classe. Sia la diretta sia l'inversa di tale associazione sono parziali, cioè può esistere uno studente per cui non è ancora stata assegnata una classe, e può esistere una classe senza studenti assegnati. Per semplicità, supponiamo che il contenuto delle tabelle sia quello mostrato nella seguente figura.

Tabella **STUDENTE**

Matricola	Nome	Cognome	NomeClasse
1111	Paolo	Rossi	5CA
2222	Gino	Verdi	4CA
3333	Luigi	Neri	NULL

Tabella **CLASSE**

NomeClasse	Piano
5CA	2
4CA	2
3CA	1

Si evince che Luigi Neri, pur essendo uno studente della scuola, non è ancora stato assegnato a una classe, e la classe 3CA non ha ancora avuto studenti assegnati.

Join o cross join

Esaminiamo l'istruzione JOIN o la sua equivalente CROSS JOIN. La sintassi è:

```
<TABELLA1> (CROSS) JOIN <TABELLA2>
```

Questa operazione corrisponde all'operazione relazionale di **prodotto**, ovvero la tabella risultato contiene tutte le combinazioni possibili tra i valori delle t-uple della <TABELLA1> e quelli della <TABELLA2>.
<TABELLA1> e <TABELLA2> potrebbero essere generate da ulteriori query.
Nell'esempio degli studenti e delle classi il risultato è mostrato nella seguente figura.

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
JOIN CLASSE C;
```

(CROSS)JOIN

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	5CA
Luigi	Neri	NULL	5CA
Paolo	Rossi	5CA	4CA
Gino	Verdi	4CA	4CA
Luigi	Neri	NULL	4CA
Paolo	Rossi	5CA	3CA
Gino	Verdi	4CA	3CA
Luigi	Neri	NULL	3CA

Per individuare le specifiche colonne di cui vogliamo la visualizzazione, poiché possono comparire con lo stesso nome in tabelle diverse, abbiamo usato la dot notation:

```
<NOMETABELLA>.<NomeColonna>
```

Useremo la dot notation anche per abbinare le tabelle in base all'attributo comune (realizzando così l'operazione di **inner join**).
Per evidenziare i differenti tipi di join, abbiamo visualizzato nella tabella risultato due colonne contenenti il nome della classe: la prima è relativa alla tabella *Studente* la seconda alla tabella *Classe*.

lezione 25

Join

Inner join

La sintassi dell'operazione di **INNER JOIN** (equivalente al NATURAL JOIN) è:

```
<TABELLA1> INNER JOIN <TABELLA2> ON <Condizione>
```

Viene restituita una tabella, in cui sono presenti solo le combinazioni delle t-uple della prima tabella che trovano una corrispondenza (per gli attributi comuni specificati nella condizione) nell'altra.
Nell'esempio degli studenti e delle classi il risultato è mostrato nella figura seguente.

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
INNER JOIN CLASSE C
ON S.NomeClasse = C.NomeClasse;
```

INNER (o NATURAL) JOIN

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA

Il risultato ottenuto corrisponde alla richiesta: "visualizzare gli studenti che sono stati iscritti alla scuola e che hanno avuto una classe assegnata".
Lo studente *Luigi Neri* non è visualizzato perché non ha ancora classi assegnate.
La classe *3CA* non è visualizzata perché non ha alcuno studente assegnato.
Il contenuto della tabella risultato è strettamente legato alla parzialità della diretta e dell'inversa. In sostanza, le t-uple di destra che non hanno una controparte a sinistra, e le t-uple di sinistra che non hanno una controparte a destra, non vengono visualizzate.

Inner join tra tabelle utilizzando SELECT

L'operazione di INNER JOIN può essere realizzata anche utilizzando il solo comando SELECT. La sintassi è:

```
SELECT <ListaAttributi>
FROM <TABELLA1>, <TABELLA2>
WHERE <TABELLA1>.<AttributoX> = <TABELLA2>.<AttributoX>;
```

Nella clausola FROM si specificano i nomi delle due tabelle, <TABELLA1>, <TABELLA2>, nella clausola WHERE si specifica la condizione sull'attributo comune: <AttributoX>.
Avremmo potuto ottenere lo stesso risultato, mostrato nella figura precedente, scrivendo:

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S, CLASSE C
WHERE S.NomeClasse = C.NomeClasse;
```

Left join

La sintassi dell'operazione di **LEFT JOIN** (o join sinistro) è:

```
<TABELLA1> LEFT JOIN <TABELLA2> ON <Condizione>
```

Il risultato è dato dalle t-uple della <TABELLA1> (quella che compare a sinistra nella sintassi) e da quelle della <TABELLA2>, che hanno un valore corrispondente per l'attributo comune.
Nell'esempio degli studenti e delle classi il risultato è mostrato nella figura che segue.

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
LEFT JOIN CLASSE C
ON S.NomeClasse = C.NomeClasse;
```

LEFT JOIN

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA
Luigi	Neri	NULL	NULL

Il risultato ottenuto corrisponde alla richiesta: "visualizzare tutti gli studenti che sono iscritti alla scuola, anche se non è stata ancora assegnata loro una classe".
Gli studenti compaiono tutti, indipendentemente dal fatto che siano, o meno, stati assegnati alle classi. Lo studente *Luigi Neri*, infatti, non è stato ancora assegnato a una classe, ma compare lo stesso.
La classe *3CA* non compare perché non è collegata a nessuna t-upla a sinistra, cioè a nessuno studente.
Il risultato è strettamente legato alla parzialità della diretta; in sostanza elimina le t-uple di destra che non hanno una controparte a sinistra.

Right join

La sintassi dell'operazione di **RIGHT JOIN** (o join **destro**) è:

```
<TABELLA1> RIGHT JOIN <TABELLA2> ON <Condizione>
```

Il risultato è dato dalle t-uple della <TABELLA2> (quella che compare a destra nella sintassi) e da quelle della <TABELLA1>, che hanno un valore corrispondente per l'attributo comune.
Nell'esempio degli studenti e delle classi il risultato è mostrato nella figura che segue.

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
RIGHT JOIN CLASSE C
ON S.NomeClasse = C.NomeClasse;
```

RIGHT JOIN

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA
NULL	NULL	NULL	3CA

Il risultato ottenuto corrisponde alla richiesta: "visualizzare le classi presenti nella scuola e solo quegli studenti che sono già assegnati a tali classi".
Le classi sono presenti tutte, indipendentemente dal fatto che abbiano o meno studenti. La classe *3CA*, infatti, non ha studenti assegnati ma compare lo stesso.
Lo studente *Luigi Neri* non compare poiché non è collegato a nessuna t-upla a destra, cioè a nessuna classe.
Questa operazione è strettamente legata alla parzialità dell'inversa. In sostanza elimina le t-uple di sinistra che non hanno una controparte a destra.