SELECT: interrogare MySQL

SELECT è sicuramente l'istruzione più utilizzata in SQL, in quanto è quella che ci permette di svolgere il lavoro fondamentale che deve fare un database, cioè recuperare i dati memorizzati.

Prima di analizzarne la sintassi è importante capire bene un concetto essenziale: ogni SELECT produce un risultato che è costituito, da un punto di vista logico, da **una tabella**, esattamente come quelle da cui va a leggere. Il risultato infatti è composto, come ogni tabella, da righe e colonne.

Le colonne sono quelle che indichiamo nella SELECT, mentre le righe sono selezionate dalle tabelle originali in base alle condizioni espresse nella clausola WHERE. Vediamo due esempi che rappresentano i casi opposti:

```
SELECT * FROM nome_tabella

SELECT COUNT(*) FROM nome_tabella
```

Questi due esempi sono apparentemente molto simili, ma in realtà producono effetti diametralmente opposti riguardo alla tabella risultato (o *resultset*).

Nel primo caso otterremo tutte le righe e tutte le colonne della tabella (in pratica una copia); infatti l'uso dell'**asterisco nella SELECT** significa *'voglio tutte le colonne'*, mentre l'assenza della `WHERE` fa sì che tutte le righe vengano restituite.

Nel secondo caso invece il resultset è formato da una sola riga e una sola colonna: infatti la colonna è costituita dall'espressione COUNT(*) (che significa 'voglio sapere quante sono le righe'), mentre il fatto di avere utilizzato una **funzione di colonna** (la stessa COUNT) fa sì che tutte le righe siano "concentrate" in una sola. Le funzioni di colonna svolgono proprio questo compito: quello di elaborare `n` righe (in questo caso tutte le righe della tabella, visto che di nuovo manca la WHERE) e riassumerle in un valore unico. Un altro esempio per chiarire meglio:

```
SELECT COUNT(*), MAX(colonna1) FROM nome_tabella WHERE colonna2 = valore
```

In questo caso avremo sempre una riga, visto che anche qui abbiamo usato funzioni di colonna. Le colonne però saranno due, perchè abbiamo chiesto due valori: il numero di righe e il valore massimo di `colonna1`. La presenza della clausola WHERE fa sì che vengano incluse nel conteggio solo le righe in cui il valore di `colonna2` è uguale a quello specificato.

Diamo ora un'occhiata alla sintassi della SELECT (alcune clausole sono state omesse per semplicità):

```
SELECT

    [ALL | DISTINCT | DISTINCTROW ]

    espressione, ... [INTO OUTFILE 'nome_file' opzioni | INTO DUMPFILE 'nome_file']

    [FROM tabelle

        [WHERE condizioni]

        [GROUP BY {nome_colonna | espressione | posizione}

        [ASC | DESC], ... [WITH ROLLUP]]

        [HAVING condizioni]

        [ORDER BY {nome_colonna | espressione | posizione}

        [ASC | DESC] , ...]

        [LIMIT [offset,] numero_righe]

    ]
```

Come vedete, la struttura fondamentale di una SELECT è la seguente:

- **SELECT**, seguita da una o più espressioni che saranno le colonne della tabella risultato;
- **FROM**, seguita dai nomi di una o più tabelle dalle quali devono essere estratti i dati;
- **WHERE**, che specifica le condizioni in base alle quali ogni riga sarà estratta oppure no dalle tabelle;
- **GROUP BY**, che specifica le colonne sui cui valori devono essere raggruppate le righe nel risultato: tutte le righe con valori uguali verranno ridotte a una;
- **HAVING**, che specifica ulteriori condizioni da applicare alle righe *dopo* il raggruppamento effettuato da `GROUP BY`;
- **ORDER BY**, che specifica in quale ordine figureranno le righe del resultset;
- **LIMIT**, che stabilisce il massimo numero di righe da estrarre

È possibile omettere tutta la parte da FROM in poi per effettuare query molto semplici che non fanno riferimento ad alcuna tabella, ad esempio:

che estrae la data e l'ora attuali.

Le *espressioni* che formeranno le colonne in output possono riferirsi a colonne delle tabelle referenziate, ma possono anche essere ottenute con funzioni applicate alle colonne, o con espressioni matematiche, o con funzioni che restituiscono valori indipendenti (come nel caso di NOW()).

Abbiamo già visto che se usiamo le cosiddette *funzioni di colonna* otterremo un valore unico che rappresenta *n* righe. Di conseguenza non possiamo avere un normale nome di colonna (valore scalare) di fianco ad una funzione di colonna:

```
SELECT colonna1, max(colonna2) FROM nome_tabella
```

Questa query produce un errore, perchè `colonna1` restituirebbe un valore per ogni riga della tabella, mentre `max(colonna2)` restituisce un valore unico. Di conseguenza non è possibile determinare quante righe deve avere la tabella risultato. L'unico modo per avere una `SELECT` in cui funzioni di colonna stanno accanto a valori scalari è di applicare una GROUP BY su questi ultimi: in questo modo le funzioni di colonna verranno applicate non a tutti i valori della tabella risultato, ma singolarmente ai gruppi di valori che fanno parte di ogni gruppo di righe.

Un esempio può chiarire meglio il concetto:

```
SELECT categoria, max(stipendio) FROM dipendenti GROUP BY categoria
```

Da questa query otterremo una riga per ogni diverso valore di `categoria`, ed in ogni riga il valore di `max(stipendio)` sarà riferito alle righe che hanno quel determinato valore di `categoria`. Così potremo sapere qual è lo stipendio massimo degli impiegati, dei quadri e dei dirigenti. In sostanza possiamo affermare che le funzioni di colonna vengono applicate ai gruppi di righe, e solo se non prevediamo raggruppamenti verranno applicate all'intera tabella risultato.

La clausola **FROM** indica la tabella o le tabelle da cui i dati saranno estratti. Le query più semplici estraggono dati da una sola tabella, ma è molto frequente aver bisogno di combinare più tabelle. In questo caso si effettua una JOIN.

Poiché ogni tabella appartiene a un database, la forma completa è `nome_database.nome_tabella`. Se non indichiamo un nome di database si sottintende l'uso del database corrente. Inoltre ad ogni tabella possiamo associare un alias a cui fare riferimento nel resto della query, attraverso la clausola **AS** (ad es. `FROM ordini AS o`).

La clausola **WHERE** determina le condizioni che saranno applicate ad ogni riga della

tabella di input per decidere se tale riga farà parte del risultato.

La **GROUP BY**, come abbiamo già visto, si usa per raggruppare i risultati sui valori di una o più colonne. La tabella risultato verrà ordinata in base a questi valori. Se aggiungiamo la clausola `WITH ROLLUP`, otterremo delle righe extra con i totali sui dati numerici ad ogni rottura di valore dei campi raggruppati.

La clausola **HAVING** svolge una funzione di selezione delle righe, esattamente come `WHERE`. La differenza è che WHERE viene applicata sulle righe delle tabelle originali, mentre `HAVING` viene applicata sulle righe della tabella risultato dopo i raggruppamenti richiesti dalle `GROUP BY`. In pratica, è utile per effettuare test sui valori restituiti dalle funzioni di colonna.

Con **ORDER BY** indichiamo su quali valori ordinare l'output. Se aggiungiamo DESC i valori saranno ordinati in modo discendente, mentre il default è ascendente. Se non usiamo la ORDER BY l'ordine dei risultati sarà indefinito (a meno che non usiamo GROUP BY).

La clausola **LIMIT**, infine, limita il numero di righe che saranno restituite. Se usata con un solo parametro, indica il numero massimo di righe restituite a partire dalla prima. Se usata con due parametri, il primo indica la riga di partenza (la prima riga equivale a 0), mentre il secondo è il numero massimo di righe.

SELECT DISTINCT

La clausola **DISTINCT** permette di escludere dal risultato le righe duplicate, ovvero quelle identiche ad altre righe. Se ci sono due o più righe di risultato uguali, con `DISTINCT` (o `DISTINCTROW`, che è sinonimo) ne vedremo una sola.

ALL è l'opposto di DISTINCT, cioè estrae tutto, ed è il valore applicato per default.

Ottenere un file della query

INTO OUTFILE si usa per scrivere la tabella risultato su un file di output che verrà creato sul server (non si può usare un nome di file già esistente). Le opzioni relative sono le stesse `FIELDS` e `LINES` già viste per `LOAD DATA INFILE`, di cui `SELECT INTO OUTFILE` è complementare.

Per usare questa clausola **è necessario il privilegio FILE**. Se invece di INTO OUTFILE si usa INTO DUMPFILE, il file di output conterrà una sola riga, senza delimitatori di colonne o di righe e senza escape di caratteri.