Introduzione all'SQL

L'**SQL** (Structured Query Language) è il linguaggio standard per la manipolazione dei database su qualsiasi DBMS.
Lo standard si limita alla sintassi ed al nome dei comandi, ma ogni DBMS parla un dialetto che si differenzia sia pur minimamente dagli altri; nel corso della guida vedremo i casi in cui piccole porzioni di codice sono proprietarie di Access piuttosto che di altri DBMS.

Alcuni DBMS, ad esempio, utilizzando dei comandi proprietari di quello stesso DBMS che non esistono su altri programmi.

Con l'SQL è possibile creare una tabella, modificarne la struttura o cancellarla. E' possibile effettuare ricerche più o meno precise tra i record di una tabella, inserire nuovi dati, modificare o cancellare dati esistenti.

E' possibile mettere in relazione due o più tabelle in funzione della struttura delle tabelle e delle esigenze. A pro di questo tengo a fare una precisazione: l'SQL è un linguaggio molto semplice che si avvicina in maniera impressionante al linguaggio umano e lo stesso utilizzo di un DBMS non è di particolare complessità; la difficoltà di essere un buon progettista di database è quella di avere cognizione di cosa vuol dire gestire un archivio.

Per essere un buon progettista di database, quindi, è necessario entrare nell'ottica citata; lo scopo di questa guida è quella di fornire una buona introduzione al lettore neofita e di chiarire alcuni dubbi al lettore più esperto.

SELECT - Interrogazione di una tabella

La sintassi del linguaggio SQL è abbastanza flessibile, sebbene ci siano delle regole da rispettare come in qualsiasi linguaggio di programmazione. Qui analizzeremo il comando SQL **SELECT**. Con l'istruzione SELECT di SQL possiamo creare degli script di interrogazione al database, interrogazioni dette **query**.
La sintassi base di una SELECT SQL è la seguente:

```
SELECT * FROM nome_tabella
```

dove ***** sta per *tutti i campi*. La query

```
SELECT id, autore FROM autori
```

estrae tutti i campi dalla tabella autori. Al posto di ***** posso usare i nomi dei campi che mi interessa estrarre.

Inseriamo adesso un filtro nella query utilizzando l'operatore WHERE:

```
SELECT autore FROM autori WHERE id = 1
```

Possiamo anche decidere di conoscere l'id di una tabella in funzione di un altro parametro di ricerca. Ad esempio:

```
SELECT id FROM autori WHERE autore = 'J.R.R. Tolkien'
```

Attenzione: effettuando una ricerca in funzione di un dato di tipo numerico non dobbiamo utilizzare gli apici per racchiudere il valore; in SQL i singoli apici delimitano una stringa.
Creiamo adesso una serie di query sulla tabella libri che, essendo più ampia, ci permette di giocare un po' di più.

Estraiamo tutti i titoli dalla tabella libri dove il titolo inizia per I utilizzando gli operatori AND per stabilire due condizioni entrambe vere e LIKE per effettuare una ricerca generica:

```
SELECT titolo FROM libri WHERE titolo LIKE 'I*'
```

L'operatore LIKE necessita del sotto-operatore ***** per identificare *tutto il resto della stringa*.**'I*'** vuol dire *tutto ciò che inizia per I*.
LIKE permette di effettuare ricerche su stringhe a partire dall'inizio della stringa, dalla fine della stringa o dalla fine. Rispettivamente potremmo avere:

```
SELECT * FROM libri WHERE titolo LIKE '*Anelli'
```

e/o

```
SELECT * FROM libri WHERE titolo LIKE '*Signore*'
```

impostando semplicemente ***** come conviene. In altri DBMS il simbolo ***** per il LIKE viene sostituito da **%**.
Proviamo ad effettuare quattro ricerche in cui il prezzo è: inferiore a 50 euro; superiore a 50 euro; diverso da 50 euro; compreso tra 30 e 60 euro. Avremo rispettivamente:

```
SELECT * FROM libri WHERE prezzo < 50
SELECT * FROM libri WHERE prezzo > 50
SELECT * FROM libri WHERE prezzo <> 50
SELECT * FROM libri WHERE prezzo BETWEEN 30 AND 60
```

JOIN - Relazioni tra più tabelle

Lo scopo di una database è quello di conservare i dati in maniera stabile, ma anche quello di organizzarli in forma **normalizzata**, evitando la **ridondanza** dei dati.
Normalizzare un database significa creare una struttura tale in cui i dati sono fisicamente separati tra loro ma possono essere messi insieme con le **relazioni**.
I casi di studio sono molteplici e di diversa natura, in cui si può arrivare a diverse soluzioni, magari altrettanto valide. Prendiamo il caso delle nostre due tabelle, **autori** e **libri**; la tabella libri contiene un campo di riferimento all'autore del libro specificato, il cui nome si trova fisicamente all'interno della tabella autori. Avremmo potuto inserire ogni volta il nome dell'autore nel record in cui sono specificati i dati del libro ed avremmo potuto definire una query del tipo

```
SELECT DISTINCT autore FROM libri
```

per ottenere un report descrittivo di tutti gli autori presenti, elencandoli una singola volta.
La scelta di due tabelle per un'esigenza come quella della nostra libreria informatizzata ha dei pro e dei contro. Il pro è che si ottiene un database normalizzato che evita la ridondanza dei dati; il contro è che l'implementazione di una relazione tra due o più tabelle determina un maggior dispendio di memoria in fase di esecuzione della ricerca. A seconda delle esigenze e delle tecnologie a disposizione potremmo effettuare scelte

differenti per ottenere le soluzioni ottimali in funzione delle esigenze pratiche: da qui si determinerà la nostra bravura di progettisti di database.

Esistono due sistemi differenti per implementare una relazione, ovvero utilizzando una semplice clausola WHERE, oppure utilizzando il comando **INNER JOIN**; la differenza tra i due è nel rapporto potenza/dispendio di memoria: il comando JOIN è più performante in termini di potenza ma richiede un maggior impiego di memoria in fase di esecuzione della ricerca. Facciamo un esempio con la modalità tradizionale in cui ricerchiamo tutti i libri scritti da Tolkien, visualizzando anche il nome dell'autore al posto dell'id di riferimento.

Premetto che per indicare un campo contenuto in una determinata tabella, in SQL si usa la forma *nome_tabella.noe_campo*, ad esempio **libri.titolo**.
Ecco il codice:

```
SELECT
    autori.autore,
    libri.titolo,
    libri.prezzo
FROM
    autori,
    libri
WHERE
    autori.id = libri.id_autore
AND
    autori.id = 1
```

Specifico i nomi dei campi che voglio visualizzare associandoli alla tabella di appartenenza; specifico le tabelle in cui sono contenuti i dati che mi interessano; nel WHERE indico per prima cosa i campi di relazione (detti comunque anche *campi di JOIN*), ovvero effettuo l'associazione tra il'id dell'autore con l'id di riferimento all'autore nel record del libro; con una clausola AND specifico che voglio estrarre i libri scritti dall'autore che ha id = 1, ovvero Tolkien.
Scriviamo la stessa query utilizzando l'istruzione INNER JOIN:

```
SELECT
    autori.autore,
    libri.titolo,
    libri.prezzo
FROM
    autori
INNER JOIN
    libri
ON
    autori.id = libri.id_autore
WHERE
    autori.id = 1
```

Con questo codice: specifico i campi; specifico la prima tabella e poi la seconda, associandole i campi di relazione con la clausola **ON** ed utilizzo il WHERE per stabilire il criterio (o filtro, che dir si voglia) di ricerca.

Divertitevi ad effettuare ricerche in base a dei criteri relazionando le due tabelle.

Funzioni di aggregazione

Le funzioni di aggregazioni sono funzioni standard native di SQL che permettono di ottenere valori numerici e/o effettuare calcoli in funzione di query specifiche. Di seguito l'elenco delle funzioni di aggregazione di SQL in schema tabellare:

Funzione	Descrizione
AVG()	Restituisce la media tra due valori specificati
COUNT()	Restituisce un intero che indica il numero di record trovati
MAX()	Restituisce il valore massimo tra due valori
MIN()	Restituisce il valore minimo tra due valori
SUM()	Restituisce la somma tra più record dello stesso campo

Nell'utilizzo di una funzione di aggregazione è importante (consigliato, anche se non obbligatorio) specificare un *alias* per il risultato, con l'utilizzo della clausola **AS**. I nostri alias si chiameranno, per convenzione, **temp** (che utilizzo in genere per definire un valore temporaneo).

Facciamo qualche esempio.

Restituisce la media del prezzo di tutti i libri trovati:

```
SELECT AVG(prezzo) AS temp FROM libri
```

Restituisce il numero di libri trovati:

```
SELECT COUNT(id) AS temp FROM libri
```

Restituisce il prezzo del libro più costoso:

```
SELECT MAX(prezzo) AS temp FROM libri
```

Restituisce il prezzo del libro meno costoso:

```
SELECT MIN(prezzo) AS temp FROM libri
```

Restituisce la somma dei prezzi di tutti i libri:

```
SELECT SUM(prezzo) AS temp FROM libri
```

ovviamente in funzione di una singola unità. Per sapere il guadagno totale bisognerebbe creare una tabella **vendite**, ad esempio, ed inserirci i libri venduti al relativo prezzo.