# Introduzione ad SQL

Guida a cura di **Rio Chierego**

1. Introduzione
2. Gli operatori
3. Istruzione SELECT
4. Istruzione INSERT
5. Istruzione UPDATE
6. Istruzione DELETE
7. Istruzione CREATE, ALTER e DROP

# 1 INTRODUZIONE

Sql (Structured Query Language) è il linguaggio standard per la manipolazione di banche dati, o in gergo più tecnico, di Database, un Database è una struttura più o meno complessa di dati, immaginiamo un lungo elenco cartaceo con informazioni di carattere vario su persone, a partire dai dati fondamentali come il nome ed il cognome, dall'indirizzo ed il numero di telefono fino alle più disparate informazioni.

Gestire un archivio con una simile mole di informazioni non è cosa banale, per questo, con l'avvento dell'informatica (che di per se vive sul concetto di banca dati), sono nati più o meno potenti software per la gestione di Database, i DBMS (Database Management System), oggi detti anche RDBMS (Relational ...).

In un sistema informativo (dove sistema informativo rappresenta l'interazione tra sistema informatico e l'uomo) i DBMS si frappongono tra un software di tipo gestionale (archivio per la conservazione di dati, semplice programma di contabilità, applicazione Web ecc...) e l'utente finale, i DBMS più diffusi sul mercato sono attualmente Oracle, Basis, Db2, Ms Sql Server, mySql, Ms Access, ecc...

La differenza che passa tra i primi nei confronti di Access è che quest'ultimo è un DBMS locale, mentre gli altri sono DBMS server-side, nel senso che il loro funzionamento in fase di sviluppo avviene mediante un dialogo tra un client (macchina di sviluppo) ed un server (dove risiede fisicamente il DBMS).

Tutti questi software parlano in Sql, ognuno con il suo "dialetto" e con piccolissime differenze di sintassi tra loro.

In questo manuale faremo spesso riferimento ad Access per toccare con mano come funziona e cosa si può fare con un sistema di Database, pur non essendo questo una guida all'uso del suddetto software, ma essendo il più semplice da usare, installare, e non richiedendo particolari architetture, è quello che più si adatta alle esigenze iniziali.

Il lettore non pretenderà di potersi definire un DBA (Database Administrator) alla fine dello studio del presente manuale, ma potrà star certo di sapere come e perchè servirsi di un Database e come interfacciarlo con un'applicazione Web.

Concluso questo doveroso cappelletto sui Database e sul mondo che li circonda, focalizziamo la nostra attenzione ora nello specifico su Sql.

Sql è un linguaggio di semplicissima comprensione, si avvicina in maniera impressionante al linguaggio umano, con pochissime operazioni si possono realizzare query (letteralmente "interrogazioni", ma il termine viene utilizzato per definire tutte le possibili operazioni fattibili con Sql) molto complesse a dirsi ed ancor più semplici a farsi, possiamo:

1. creare una tabella *
2. inserire dati in una tabella
3. modificare dati esistenti di una tabella
4. cancellare dati esistenti da una tabella
5. interrogare il database ed estrarre dati da una tabella.

* Una tabella è la parte più piccola e più importante del Database, è la struttura fisica contenente i dati che hanno subito le operazioni sopra elencate.
Immaginiamo una griglia con righe e colonne, in cui ogni singola colonna ha un'intestazione (nome) chiamata campo e le righe, formate da una sola voce di ogni singolo campo, vengono chiamate record.

Questo concetto, forse per qualcuno un pò astratto, potrà tornare più chiaro con un esempio:

| ID | Nome | Cognome | Città |
|---|---|---|---|
| 1 | Luca | Ruggiero | Napoli |
| 2 | Giuseppe | Verdi | Roma |
| 3 | Antonio | Bianchi | Firenze |
| 4 | Mario | Rossi | Milano |

Possiamo accorgerci che abbiamo in orizzontale informazioni relative ad una stessa persona (record), ed in verticale informazioni appartenenti ad una stessa categoria (campo), come ad esempio il nome.

Torna su

# 2 GLI OPERATORI

Prima di addentrarci nella sintassi Sql e di sperimentare alcuni esempi è necessario passare in rassegna pochi ma fondamentali operatori che lavorano con l'Sql, avrei potuto fornire la spiegazione di quanto detto direttamente nei capitoli dedicati alle operazioni di ricerca, inserimento ecc, cosa che non mancherò di fare comunque fornendo esempi, ma ho preferito scrivere un capitolo dedicato perchè si focalizzi meglio l'attenzione su questi particolari aspetti e tutto risulti più chiaro il seguito.

## Operatori di confronto

Gli operatori di confronto servono a stabilire uguaglianze e disuguaglianze tra il parametro di ricerca (o inserimento ecc...) ed il valore che si intende ricercare, ecco lo schema:

| Operatore | Descrizione |
|---|---|
| = | Esprime un'uguaglianza perfetta tra due valori |
| LIKE | Esprime 'somiglianza' tra i due valori (es: manuali & manuale) |
| < | La variabile sulla sinistra è MINORE di quella sulla destra |
| > | La variabile sulla sinistra è MAGGIORE di quella sulla destra |
| < = | Stabilisce che un valore è MINORE O UGUALE all'altro |
| > = | ... MAGGIORE O UGUALE |
| < > | Stabilisce che i due valori sono DIVERSI tra loro |
| BETWEEN | Stabilisce che il valore è compreso tra 2 valori (es: tra 1 e 5) |

## Operatori condizionali

In Sql è possibile effettuare operazioni solo su campi che rispettano le condizioni richieste grazie ad operatori condizionali:

| Operatore | Descrizione |
|---|---|
| WHERE | Letteralmente 'DOVE', stabilisce le condizioni su cui lavorare |

## Operatori logici

Quando si stabilisce una condizione, come appena visto, è necessario avere la facoltà di stabilirne più di una, o che una escluda un'altra, ecco l'utilità degli operatori logici:

| Operatore | Descrizione |
|---|---|
| AND | Specifica due o più condizioni da soddisfare |
| OR | Il verificarsi di una condizione ESCLUDE l'altra |

Torna su

# 3 ISTRUZIONE SELECT

Con questo capitolo iniziamo il percorso verso la teoria e la pratica sulla sintassi Sql iniziando dai criteri con cui è possibile effettuare ricerche, in gergo tecnico dette query.

Come già accennato nei capitoli precedenti, l'Sql somiglia in maniera impressionante al linguaggio umano, con una scarsa terminologia è possibile realizzare potenti query (il termine è utilizzato anche per indicare genericamente operazioni Sql) sui nostri database.

L'istruzione con cui possiamo inizializzare una ricerca è SELECT, Sql è *case-insensitive* nei confronti dei comandi proprietari del linguaggio, è la stessa cosa scrivere SELECT o select, ma diventa *case-sensitive* quando si specificano i nomi delle tabelle o dei campi del database, quindi prego di prestare la massima attenzione a questo aspetto, consiglio comunque di scrivere in maiuscolo i comandi ed in minuscolo i nomi, in modo leggere meglio il codice.

Si ipotizzi di lavorare sulla tabella anagrafe con i seguenti campi:


anagrafe : Tabella

| Nome campo | Tipo dati |
|---|---|
| id | Contatore |
| nome | Testo |
| cognome | Testo |
| anni | Testo |
| nato_a | Testo |
| nato_il | Testo |
| indirizzo | Testo |
| telefono | Testo |

Ipotizzando che la tabella sia stracolma di dati eseguiamo la più banale delle ricerche, il cui risultato sarà la restituzione di tutti i record presenti:

```
SELECT * FROM anagrafe;
```

letteralmente:

SELEZIONA tutti i campi DALLA TABELLA anagrafe;

Il simbolo asterisco (*) indica appunto tutti i campi, ogni operazione va conclusa con un punto e virgola (;), potremmo però decidere di selezionare ad esempio solo il campo nome:

```
SELECT nome FROM anagrafe;
```

oppure due o più campi:

```
SELECT nome,cognome,anni FROM anagrafe;
```

Utilizziamo quindi la virgola (,) per separare più valori. Creiamo ora una query che restituisca determinati record in base ad una condizione:

```
SELECT * FROM anagrafe WHERE nome = 'Luca';
```

letteralmente:

SELEZIONA tutti i campi DALLA TABELLA anagrafe DOVE IL CAMPO nome E' UGUALE A Luca;

Possiamo potenziare i criteri di ricerca stabilendo più di una condizione:

```
SELECT* FROM anagrafe WHERE nome = 'Luca' AND cognome = 'Ruggiero';
```

letteralmente:

SELEZIONA tutti i campi DALLA TABELLA anagrafe DOVE IL CAMPO nome E' UGUALE A Luca ED IL CAMPO cognome E' UGUALE A Ruggiero;

Possiamo però avere l'esigenza di escludere l'uno o l'altro criterio di ricerca, quindi:

```
SELECT * FROM anagrafe WHERE nome = 'Luca' OR nome = 'Pippo';
```

Non è finita qui, possiamo effettuare una ricerca che consenta di trovare tutti i record i cui campi utilizzati abbiano valori simili, se ad esempio scrivo:

```
SELECT * FROM anagrafe WHERE nome LIKE 'Luca';
```

tra i risultati della ricerca troverò probabilmente anche eventuali ocorrenze di Lucia e simili.

Effettuare una ricerca in questo modo comunque inutile, possiamo utilizzare l'operatore LIKE integrando nell'espressione il simbolo percentuale (%) che serve a recuperare una parte non specificata del valore di un campo, ad esempio *%luca* troverà sia i campi con l'occorrenza di Luca che di Gianluca ecc... un esempio:

```
SELECT * FROM anagrafe WHERE nome LIKE '%luca';
```

Il simbolo % più essere utilizzato sia all'inizio che alla fine di un valore, o in tutti e due i posti.

Possiamo, grazie all'operatore di confronto BETWEEN, selezionare i record dove, ad esempio nel campo *anni* l'età sia compresa tra un certo numero di anni ed un numero maggiore:

```
SELECT * FROM anagrafe WHERE anni BETWEEN '20' AND '30';
```

I risultati saranno tutti i record il cui corrispondente valore del campo *anni* andrà da 21 a 29.

Grazie al comando DISTINCT possiamo ricercare record che potrebbero essere stati ripetuti, ad esempio se ci sono due Luca Ruggiero nel database possiamo tentare di arrivare alla persona che ci interessa con una certa facilità, se ad esempio scrivessimo:

```
SELECT DISTINCT nome,cognome,nato_a FROM anagrafe WHERE cognome =

'Ruggiero' AND nato_a = 'Napoli';
```

ritroveremo un solo Luca Ruggiero nato a Napoli, ma ovviamente potrebbero essercene comunque più di uno, dobbiamo quindi "chiudere il cerchio" il più possibile specificando ulteriori condizioni, aggiungiamo quindi all'esempio precedente l'espressione:

```
... AND anni = '24';
```

e troveremo Luca Ruggiero nato a Napoli che ha 24 anni.

Possiamo ordinare i nostri risultati in ordine crescente o decrescente scegliendo come parametro un qualsiasi campo, il nome, il cognome, l'età, la città di nascita ecc, vediamo un esempio in cui ordiniamo il ordine alfabetico crescente i record in base al cognome:

```
SELECT * FROM anagrafe ORDER BY cognome;
```

ORDER BY ci permette di fare questo, ma possiamo anche, grazie al comando DESC, ordinare i risultati in ordine decrescente:

```
SELECT * FROM anagrafe ORDER BY cognome DESC;
```

Concludiamo la lezione contando i record che ci interessano col comando COUNT:

```
SELECT COUNT (*) FROM anagrafe WHERE nome = 'Luca';
```

sapremo quanti Luca ci sono nel database, ma possiamo fare ancora di più:

```
SELECT DISTINCT COUNT (*) FROM anagrafe;
```

in questo modo se ci sono dei record che si ripetono possiamo escluderli dal conteggio e sapere quanti record univoci sono presenti.

E' comunque possibile ovviare a questo inconveniente direttamente da codice, se ad esempio utilizziamo il database per un'applicazione Web in Asp, possiamo da codice Asp non permettere l'inserimento di un intero record uguale effettuando dei controlli prima di autorizzare inserimento.

Torna su

# 4 ISTRUZIONE INSERT

Conclusa la lezione su come effettuare ricerche su di un database preoccupiamoci ora di inserire dati mediante l'istruzione INSERT.

Premetto che molti linguaggi di programmazione che si integrano con Sql, come ad esempio Asp, utilizzano metodi ed oggetti proprietari del linguaggio stesso per effettuare questa operazione, molti altri invece si basano su stringhe Sql ad esempio:

```
INSERT INTO nome_tabella (nome_campo) VALUES ('valore');
```

Per quanto semplice è preferibile scrivere l'istruzione su più righe per leggere il codice in maniera più chiara, lascio comunque questo aspetto alle preferenze del lettore:

```
INSERT INTO anagrafe
(
        nome,
        cognome,
        anni,
        nato_a,
        nato_il
)
VALUES
(
        'Luca',
        'Ruggiero',
        '24',
        'Napoli',
        '12/04/1978'
);
```

Verrà in questo modo generato un nuovo record con i valori stabiliti, il comando VALUES si frappone tra i nomi dei campi scritti tra parentesi tonde sulla sinistra dello stesso, ed i valori che si intende attribuire ai rispettivi campi sulla destra del VALUES

Torna su

## 5 ISTRUZIONE UPDATE

I dati sono soggetti a continui cambiamenti, si pensi al caso dell'Anagrafe, che stiamo portando avanti come esempio, dove una persona può cambiare casa, di conseguenza sarà necessario modificare l'indirizzo sul database.

Per compiere questa operazione ci serviamo dell'istruzione UPDATE e del comando SET per identificare il record interessato, la sintassi è semplice:

UPDATE nome_tabella SET nome_campo = 'valore_nuovo' WHERE nome_campo = 'valore_vecchio';

Un esempio pratico è il seguente:

```
UPDATE anagrafe SET indirizzo =
'Via Roma, 22' WHERE indirizzo = 'Corso V. Emanuele, 215';
```

In questo modo tutti gli utenti che abitano al Corso V. Emanuele al civico 215 saranno trasferiti a Via Roma al civico 22... non è il caso di fare una sciocchezza simile :-)

E' in genere necessario potenziare la stringa Sql con delle ulteriori condizioni, stabilendo ad esempio il nome ed il cognome dell'utente, o qualsiasi altro dato personale si ritiene opportuno specificare per evitare di effettuare modifiche lì dove non richiesto:

```
UPDATE anagrafe
SET indirizzo = 'Via Roma, 22'
WHERE indirizzo = 'Corso V. Emanuele, 215'
        AND nome = 'Luca'
        AND cognome = 'Ruggiero';
```

Consiglio di specificare il massimo numero possibile di parametri (aggiungendo alla stringa Sql nuovi operatori AND) per evitare di aggiornare record non richiesti

Torna su

## 6 ISTRUZIONE DELETE

L'istruzione DELETE offre la possibilità di eliminare interi record, la sintassi è semplice:

```
DELETE * FROM nome_tabella;
```

In questo modo la tabella in questione verrà svuotata interamente, è quindi necessario stabilire delle condizioni, ad esempio solo i record dove determinati campi hanno i valori precisati:

```
DELETE* FROM anagrafe WHERE nome = 'Luca' AND cognome = 'Ruggiero';
```

Tutti i record con le occorrenze di Luca e di Ruggiero nei rispettivi campi di appartenenza verranno eliminati, è quindi necessario anche in questo caso essere precisi e specificare il maggior numero possibile di condizioni per non cancellare record non richiesti.

Torna su

# 7 ISTRUZIONE CREATE, ALTER e DROP

Oltre alla ricerca, all'inserimento, alla modifica ed alla cancellazione dei dati, in Sql è possibile agire via codice in modo da creare, modificare o cancella re una tabella. Le istruzioni che ci interessano e che spiegheremo in questa lezione sono **CREATE** (crea una tabella), **ALTER** (modifica una tabella) e **DROP** (cancella una tabella).

Iniziamo con la creazione. L'istruzione CREATE prevede la seguente forma

```
CREATE TABLE
nome_tabella (nome_campo tipo_dato obbligatorio_o_meno);
```

Il primi due parametri passati tra parentesi tonde sono obbligatori, rappresentando il nome del campo ed il suo tipo di dato; il terzo è opzionale e può assumere valori **NULL** o **NOT NULL** che indicano rispettivamente che il capo può essere lasciato vuoto o meno. Per default, se omesso, il suo valore sarà NULL.

Facciamo un esempio di codice creando una tabella di prova, i cui campi indicano i vari tipi di dato accettati in Ms Access:

```
CREATE TABLE nome_tabella
(
   campo1  AutoIncrement,
   campo2  Text (15) NOT NULL,
   campo3  Memo NOT NULL,
   campo4  Integer,
   campo5  Float,
   campo6  Double,
   campo7  Byte,
   campo8  Currency,
   campo9  DateTime,
   campo10 Bit
);
```

dove

```
AutoIncrement = tipo Contatore
Text          = tipo Testo
Memo          = tipo Memo
Integer       = tipo Numerico (Intero lungo)
Float         = tipo Numerico (Precisione doppia)
Double        = tipo Numerico (Precisione doppia)
Byte          = tipo Numerico (Byte)
Currency      = tipo Valuta
DateTime      = tipo Data/ora
Bit           = tipo Si/No
```

Eseguire questa istruzione nell'editor Sql di Access.

Vediamo ora come modificare questa tabella utilizzando l'istruzione ALTER, la quale accetta tre tipi di modifica: **ADD** (aggiunge una colonna), **MODIFY** (modifica il tipo di una colonna) e **DROP** (cancella una colonna) avvalendosi dell'istruzione opzionale **COLUMN** che, a mio avviso, è bene comunque utilizzare.

Il seguente esempio aggiunge una colonna alla tabella **nome_tabella** creata in precedenza:

```
ALTER TABLE nome_tabella ADD COLUMN altro_campo Text (20) NOT NULL;
```

E' possibile modificare questo campo con l'istruzione

```
ALTER TABLE nome_tabella MODIFY COLUMN altro_campo Text (100);
```

impostando la lungnezza da 20 a 100 caratteri come massimo consentito per il suo valore. Per cancellare questo campo utilizzeremo l'istruzione

```
ALTER TABLE nome_tabella DROP COLUMN altro_campo;
```

La cancellazione di una tabella è molto semplice; è sufficiente utilizzare l'istruzione

```
DROP TABLE nome_tabella
```

Torna su