SQL (Structured Query Language)

Introduzione

Cosa è SQL

SQL non identifica un prodotto commerciale, ma un linguaggio nello stesso modo in cui C, Fortran, Cobol, Basic, Pascal indicano linguaggi generali e non compilatori particolari.
È un linguaggio, che serve per eseguire varie operazioni sia sui dati che sulle strutture che li contengono. La sigla, acronimo di *Structured Query Language*, è ormai diventata sinonimo di linguaggio standard per la gestione dei database relazionali.
SQL è dunque un linguaggio per la gestione di database relazionali, quindi assolve alle funzioni di Data Description Languge (linguaggio di descrizione dei dati e delle strutture che li conterranno), di Data Manager Language (linguaggio per la manipolazione dei dati) e di linguaggio di interrogazione.
Il termine SQL può generare confusione. La lettera S, iniziale di Structured (Strutturato), e la lettera L, iniziale di Language, sono abbastanza semplici, ma la lettera Q si presta a varie interpretazioni. Ovviamente Q sta per Query che se fosse interpretata alla lettera, limiterebbe il linguaggio SQL a uno strumento per interrogare il database. In effetti SQL fa molto di più che porre delle domande.
Gli informatici chiamano questo linguaggio di alto livello o dichiarativo perché permette di svolgere operazioni dichiarando **cosa** si deve ottenere e non **come** si deve ottenere. Ricordiamo che i linguaggi di terza generazione o procedurali sono quelli dove bisogna specificare il *come si fa*, non è sufficiente dichiarare il *cosa si deve fare*. Non è così per questo linguaggio, che pur limitando le scelte del programmatore e l'efficienza del programma, libera lo sviluppatore dal gravoso compito di scrivere pagine e pagine di codice. Chi usa questo linguaggio però, non è solo chi programma, ma anche chi si avvicina all'informatica marginalmente e per riflesso. Queste persone sono gli impiegati, i professionisti, i commessi, i magazzinieri, ecc. insomma chiunque ha la necessita di manipolare o consultare basi di dati. Forse la causa del suo grande successo sta nella sua semplicità di utilizzo. Non bisogna, però farsi ingannare, perché se da un lato SQL è intuitivo e semplice, da un altro, per essere capito a fondo richiede di essere studiato con attenzione per capirne tutte le sfumature e le notevoli potenzialità.

Storia di SQL

Le origini di SQL risalgono all'inizio degli anni 70 in California, quando la società IBM sviluppa il System R, un applicativo per la gestione dei dati, il cui linguaggio veniva chiamato *Sequel*. Questo linguaggio rappresentava l'embrione di quello che sarebbe poi diventato l'attuale SQL. Infatti alla fine degli anni 70, sempre l'IBM, sviluppa un altro prodotto il DB2 (un sistema per la gestione di database relazionali o RDBMS, *Relational Database Management System*) che utilizza una primordiale versione di SQL. Da allora si sono succeduti un gran numero di prodotti che implementano questo linguaggio e ogni produttore, aggiungendo delle variazioni e estensioni proprie, ha contribuito alla creazione della miriade di dialetti che oggi vengono chiamati SQL.

Standardizzazione di SQL

I due enti che si occupano del processo di standardizzazione ANSI (*American National Standards Organization*) e ISO (*International Standards Organization*), stanno svolgendo, ormai da anni, azioni di promozione dello standard SQL.
Sebbene questi enti preparino le specifiche cui devono adeguarsi i progettisti dei vari DBMS tutti i prodotti che implementano SQL differiscono in maniera più o meno marcata dagli standard ufficiali, aggiungendo delle variazioni alla sintassi o ampliando alcune funzioni.
Nel 1986 è stato promulgato il primo standard; esso possedeva già gran parte delle primitive di formulazione di interrogazioni, ma offriva un limitato supporto per la definizione e manipolazione dei dati e delle strutture logiche che avrebbero dovuto contenerli.
Nel 1989 lo standard è stato ulteriormente esteso ma in modo limitato; l'aggiunta più significativa di questa versione è stata la definizione dell'integrità referenziale. Si fa riferimento a questa versione dello standard usando il nome *SQL-89*.
Nel 1992 è stata pubblicata un'altra versione, in gran parte compatibile con quella precedente ma arricchita da un gran numero di nuove funzionalità. Si fa riferimento a questa ulteriore versione usando i nomi *SQL-92* o *SQL-2*.
L'opera di standardizzazione sta continuando ancora, infatti si sta lavorando ad un'ennesima versione chiamata SQL-3.

Capitolo 1
Introduzione alle query

Prime elementari regole

La sintassi del linguaggio SQL è abbastanza flessibile, sebbene ci siano delle regole da rispettare come in qualsiasi linguaggio di programmazione.

```
SELECT *
FROM ELEMENTO;
```

In questo esempio tutti i caratteri sono scritti in maiuscolo, ma non deve essere necessariamente così. Avremmo potuto anche scrivere così:

```
select *
from elemento;
```

L'asterisco (*) di select * indica al database di fornire tutte le colonne associate alla tabella specificata dalla clausola FROM.

Come termina una istruzione SQL? : in alcune implementazioni si usa il punto e virgola (;) in altre lo slash (/).

Selezionare le colonne o cambiare l'ordine di apparizione

```
select numeroatomico,simbolo
from elemento;

select simbolo, numeroatomico
from elemento;
```

Clausola *DISTINCT* (query senza duplicati)

```
select distinct stato_id
from elemento;

select distinct stato_id, gruppo_id
from elemento;
```

Capitolo 2
Espressioni e operatori condizionali

Condizioni

Tutte le volte che si vuole trovare un particolare elemento o gruppo di elemento in un database, occorre specificare una o più condizioni. Le condizioni sono introdotte dalla clausola WHERE.

```
select *
from elemento
where stato_id=1;

select *
from elemento
where stato_id=1
and serie_id=3;
```

Operatori aritmetici

Sono gli operatori aritmetici: + (somma), - (sottrazione), / (divisione), * (moltiplicazione).

Operatore moltiplicazione (*):

```
select nome, massaatomica, massaatomica*1000
from elemento;
```

Gli operatori relazionali: maggiore (>) , maggiore o uguale (>=), minore (<), minore o uguale (<=), diverso (<>) o (!=):

```
select *
from elemento
where numeroatomico>50;

select *
from elemento
where numeroatomico>=50;

select *
from elemento
where numeroatomico<>50;
```

L'operatore *IS*:

```
select *
from elemento
where numeroatomico is null;
```

Operatori di caratteri

Gli operatori di caratteri possono essere utilizzati per manipolare il modo in cui le stringhe devo essere ricercate.

Operatore *LIKE*:

In genere si usa il % per indicare caratteri qualsiasi; in Access si usa il simbolo *

```
select *
from elemento
where nome like 'M%';
```

Gli operatori logici (Algebra di BOOLE)

Operatore *AND*:

```
select *
from elemento
where nome like 'M%'
and numeroatomico<30;
```

Operatore *OR*:

```
select *
from elemento
where nome like 'M%'
or numeroatomico<30;
```

Operatore *NOT*:

```
select *
from elemento
where nome not like 'M%';
```

Gli operatori di insieme

SQL mette a disposizione degli operatori insiemistici, da applicare nella scrittura delle nostre interrogazioni. Tali operatori operano *sul risultato* di più *select*. **Gli attributi interessati dagli *operatori di insieme* devono esser di tipo compatibile tra loro.**
Gli operatori disponibili sono gli operatori di UNION (unione), INTERSECT (intersezione) e MINUS (differenza), il significato è analogo ai corrispondenti operatori dell'algebra insiemistica che adesso vedremo brevemente:

Operatore *UNION*:

L'operatore UNION restituisce il risultato di più query escludendo le righe duplicate, vediamo un esempio:

```
select numeroatomico, simbolo, nome
from elemento
where numeroatomico < 10
UNION
select numeroatomico, simbolo, nome
from elemento_copia
where numeroatomico > 100;
```

Gli Operatori *INTERSECT* e *MINUS* non sono implementati in Access

Altri operatori: *IN* e *BETWEEN*

Gli operatori IN e BETWEEN forniscono una scorciatoia per quelle operazioni che possono essere svolte anche in altri modi.
Invece di scrivere:

```
select *
from elemento
where stato_id = 1
or stato_id = 2;
```

si può scrivere:

```
select *
from elemento
where stato_id IN (1,2);
```

Invece di scrivere:

```
select *
from elemento
where numeroatomico>=20
and numeroatomico<=30
```

si può scrivere:

```
select *
from elemento
where numeroatomico BETWEEN 20 and 30;
```

(N.B. gli estremi sono compresi!!!)

Capitolo 3
Funzioni

Le *funzioni*, nell'ambito dei linguaggi di terza generazioni (linguaggi procedurali), sono delle particolari procedure le quali passandogli dei valori (parametri) esse ci restituiscono (ritornano) un valore. Anche se SQL non è un linguaggio procedurale, implementa le funzioni nella stessa maniera ma con una differenza sostanziale: nei linguaggi procedurali noi stessi possiamo crearci delle funzioni, con SQL ciò non è possibile e quindi possiamo utilizzare solo quelle funzioni che ci mette a disposizione il DBMS che stiamo usando.
In questo capitolo vedremo alcune funzioni, ma soltanto le prime 5 (COUNT, SUM, AVG, MAX e MIN) sono definite nello standard SQL. Queste prime cinque funzioni sono le più importanti e dobbiamo impararle bene, esse sono sempre presenti nella maggior parte dei DBMS a differenza delle restanti, che a volte non appaiono affatto o sono implementate con una sintassi diversa.

Funzioni aggregate

Le funzioni che analizzeremo in questo paragrafo hanno la particolarità di restituire *un solo valore*. Inoltre, dato che operano su insiemi di righe, vengono anche chiamate **funzioni di gruppo**.

COUNT

Restituisce il numero di righe che soddisfano la condizione specificata nella clausola WHERE.

```
select count(*)
from elemento
where stato_id=2;
```

SUM
Questa funzione somma tutti i valori di una colonna:

```
select sum(massaatomica)
from elemento;
```

La funzione SUM opera soltanto con i numeri, se viene applicata a un campo non numerico, si ottiene un messaggio di errore.

AVG
Calcola la media aritmetica dei valori di una colonna:

```
select avg(massaatomica)
from elemento;
```

La funzione AVG opera soltanto con i numeri.

MAX
Questa funzione serve a trovare il valore massimo di una colonna:

```
select max(massaatomica)
from elemento;
```

La funzione MAX opera anche con i caratteri: la stringa ‘Maria’ è maggiore della stringa ‘Giovanna’.

MIN
Questa funzione opera in modo analogo a MAX, ad eccezione del fatto che restituisce il valore minimo di una colonna:

```
select min(massaatomica)
from elemento;
```

La funzione MIN opera anche con i caratteri: la stringa ‘AAA’ è minore della stringa ‘BB’.
Assiomi delle funzioni aggregate:

- Restituiscono un solo valore
- La clausola SELECT può essere seguita solo e soltanto dalla funzione di aggregazione
- Vanno applicate a tipi di dato a loro compatibili

Funzioni aritmetiche e temporali (non implementate o parzialmente da Access)
Funzioni di caratteri
Queste funzioni ci permettono di manipolare i dati da visualizzare in tutti i modi e formati desiderati. Sono particolarmente utili quando abbiamo la necessità di rendere i dati più leggibili o quando vogliamo estrapolare delle informazioni sulle stringhe e i caratteri rappresentanti le informazioni.

LENGTH
La funzione LENGTH restituisce la lunghezza del suo argomento, come in questo esempio:

```
select nome,length(nome)
from elemento;
```

Capitolo 4
Le clausole SQL

Questo capitolo è dedicato alle clausole utilizzate con l’istruzione SELECT , in particolare saranno trattate le seguenti clausole:

- WHERE
- ORDER BY
- GROUP BY
- HAVING

WHERE
La clausola WHERE serve per implementare delle condizioni *verificabili a livello delle singole righe*.
Questa clausola è abbastanza semplice da usare ed è già stata utilizzata precedentemente in questo corso:

```
select *
from elemento
where numeroatomico=10;
```

ORDER BY
A volte potrebbe essere necessario presentare i risultati di una query in un certo ordine, la clausola ORDER BY assolve a questo scopo:

```
select *
from elemento
order by nome;
```

E’ possibile ordinare i record in senso inverso, con la lettera o il numero più alti in prima posizione? Sì che è possibile, tramite la parola chiave DESC. Vediamo un esempio:

```
select *
from elemento
order by nome DESC;
```

La clausola ORDER BY può essere applicata a più campi:

```
select nome, gruppo_id, numeroatomico
from elemento
order by gruppo_id, numeroatomico;
```

GROUP BY
Questa clausola ci permette di formare dei sottoinsiemi per quelle colonne specificate. Vediamo cosa significa quanto affermato:

```
select gruppo_id,count(*)
from elemento
group by gruppo_id;
```

È possibile applicare la clausola GROUP BY anche a più di un campo per volta. Vediamo come funziona:

```
select gruppo_id,stato_id,count(*)
from elemento
group by gruppo_id,stato_id;
```

HAVING
Abbiamo visto come tramite la clausola GROUP BY le righe possano venire raggruppate in sottoinsiemi. Una particolare interrogazione può avere la necessità di estrapolare solo quei sottoinsiemi di righe che soddisfano certe condizioni, in questo caso però non è possibile usare la clausola WHERE in quanto tale clausola verifica la condizione che la segue, su tutte le righe e non in maniera singola sui valori estrapolati per ogni sottoinsieme di righe. Non è possibile utilizzare WHERE per verificare condizioni sui risultati di funzioni di gruppo. Vediamo un esempio:

```
select gruppo_id,stato_id,count(*)
from elemento
group by gruppo_id,stato_id
having count(*)>=7;
```

Riepilogo:
select *lista attributi o espressioni*
from *lista tabelle*
[where *condizioni semplici***]**
[group by *lista attributi di raggruppamento***]**
[having *condizioni aggregate***]**
[order by *lista attributi di ordinamento***]**

Capitolo 5
Join - Combinazione di tabelle

Questo capitolo tratta un importante tipo di operazione tra le tabelle: la Join. Il vocabolo join significa unione e nel caso di SQL sta ad indicare unione tra tabelle. Esistono vari tipi di join, ma tutti derivano o possono essere ricondotti a vari operatori dell'algebra insiemistica. L'importanza principale della join risiede nella possibilità che ci offre per correlare e visualizzare dati appartenenti a tabelle diverse o alla medesima tabella, logicamente correlati tra di loro. I semplici dati, da noi uniti, possono assumere la forma di complesse informazioni così come noi li vogliamo.

CROSS JOIN
Per comprendere a pieno l'operazione CROSS JOIN (unione incrociata) bisogna aver ben chiaro il concetto di **prodotto cartesiano:**
Ora considerando che le tabelle non sono altro che insiemi i cui elemento sono le righe ecco che possiamo individuare l'operazione di CROSS JOIN in quella di prodotto cartesiano appartenente alle teorie degli insiemi. Dunque il prodotto cartesiano tra due o più tabelle si traduce in una istruzione chiamata CROSS JOIN. Il CROSS JOIN si ottiene in maniera molto semplice elencando dopo la FROM le tabelle che devono essere coinvolte. Vediamo un esempio di CROSS JOIN:

```
select *
from elemento, stato;
```

Il CROSS JOIN non è particolarmente utile e viene usato raramente, ma se in una CROSS JOIN si utilizza la clausola WHERE potremmo ottenere join molto più interessanti.

NATURAL JOIN
Il NATURAL JOIN è un tipo di operazione che ci permette di correlare due o più tabelle sulla base di valori uguali in attributi contenenti lo stesso tipo di dati.

```
select *
from elemento, stato
where elemento.stato_id=stato.id;
```

INNER JOIN
È un tipo di join in cui le righe delle tabelle vengono combinate solo se i campi collegati con join soddisfano una determinata condizione. Vediamo un esempio:

```
select *
from elemento, stato
where elemento.stato_id=stato.id
and stato_id=2;

select nome, descrizione
from elemento, stato
where elemento.stato_id=stato.id;
```

Capitolo 6
Manipolare i dati

Fino a questo punto del corso abbiamo trattato quella parte di SQL che assolve alle funzioni di QL (Query Language) ovvero interrogazioni dei dati senza nessuna possibilità di manipolarli come ad esempio cambiarne il valore. In questo capitolo vedremo invece quella parte di SQL che assolve alle funzioni di DML (Data Manipulation Language). Questa parte di SQL ci consente di *inserire* dati nelle tabelle, di *modificarli* e di *cancellarli*; le corrispondenti istruzioni che assolvono a tale scopo sono: INSERT, UPDATE, DELETE.
Possiamo operare con queste tre istruzioni sui dati in modo selettivo o in modo globale. Nel modo selettivo useremo delle select e/o delle condizioni per far riferimento a particolari valori o a particolari posizioni nella tabella; nel modo globale non faremo uso di select o di condizioni. Vediamo in dettaglio quanto esposto.

INSERT

```
insert into stato
values (4, 'plama');
```

È evidente che i valori che inseriamo in una tabella tramite l'istruzione INSERT devono essere dello stesso tipo del campo che li riceve; simile discorso vale per la dimensione o la lunghezza del dato da inserire che non deve superare la dimensione o la lunghezza del campo che lo riceve. Una stringa lunga 23 caratteri non può essere contenuta in un campo di tipo stringa lungo 20 caratteri.
Abbiamo usato le parole '*dimensione*' e '*lunghezza*': 'lunghezza' viene usato per campi di tipo stringa, 'dimensione' viene usato per campo di tipo numerico.

UPDATE

Questa istruzione serve per modificare i dati contenuti in una tabella

```
update stato
set descrizione='Plasma'
where id=4;
```

È anche possibile modificare più campi alla volta.

DELETE

Oltre a inserire dati in una tabella o modificarli, occorre anche poterli cancellare. Questa istruzione assolve allo scopo. Va fatto notare che DELETE non cancella un singolo campo per volta ma una riga o più righe per volta; inoltre questa istruzione cancella i record e non l'intera tabella. Vediamo ora alcuni esempi per meglio comprendere come funziona DELETE.

```
delete from stato
where id=4;
```

(Per cancellare tutte le righe di una tabella: `delete from stato`)
Come si può vedere dalla sintassi dei due esempi sull'istruzione DELETE, non è possibile cancellare l'intera tabella o singoli campi all'interno delle righe selezionate. Qualcuno a questo punto si potrebbe domandare come far scomparire l'intera tabella o come cancellare un singolo campo? Per cancellare un campo all'interno di una riga possiamo usare l'istruzione UPDATE, mentre invece come far scomparire un'intera tabella si usa un altro comando (Drop Table).
Ricapitolando possiamo affermare che DELETE ci permette di:

- cancellare una sola riga
- cancellare più righe
- cancellare tutte le righe

ROLLBACK, COMMIT

Per le sintassi di questi comandi si è fatto riferimento al tool SQL Plus 8.0 di Oracle. **Tali comandi non sono implementati da Access**.
Quando si inseriscono o si cancellano o si modificano i dati in un database è possibile intervenire al fine di annullare l'operazione o confermarla definitivamente. Ciò è particolarmente utile quando ci accorgiamo di aver eseguito uno dei tre comandi, visti in questo capitolo, sui dati errati, o quando vogliamo confermare definitivamente il comando mandato in esecuzione.
Per far maggiore chiarezza a quanto affermato va detto che i DBMS i quali implementano i comandi ROLLBACK e COMMIT, non rendono effettivi istantaneamente i comandi DELETE, UPDATE e INSERT, ma tengono memoria temporaneamente delle modifiche effettuate in un'altra area. Questo fa si che un utente, che non sia quello che ha eseguito uno dei comandi DELETE, UPDATE e INSERT, non veda le modifiche appartate; mentre l'altro, quello che ha eseguito uno dei tre comandi, veda le tabelle in oggetto come se fossero state modificate definitivamente. Dunque il comando di COMMIT rende definitive le modifiche apportate e il comando ROLLBACK elimina ogni modifica da queste ultime

Riepilogo

In questo capitolo abbiamo visto le tre istruzioni più importanti per poter manipolare i dati. Gli esempi esposti erano particolarmente semplici; c'è però da far notare che con le istruzioni *INSERT*, *UPDATE* e *DELETE* possiamo utilizzare *select* o condizioni complesse e articolate quanto vogliamo.