Sql è un linguaggio standard che permette di operare con i database. Per database intendo uno qualsiasi e non il solito Access, ma anche Oracle, Microsoft SQL Server, Informix, DB2, Sybase...

Sql sta per **S**tructured **Q**uery **L**anguage, e permette di inserire, modificare e cancellare dei dati da database, ma permette anche di prelevarli, elaborarli e visualizzarli in diversi modi, nonchè di effettuare varie operazioni (creazione e cancellamento di tabelle e database).

Impararlo è molto semplice, esistono manuali, o meglio mattoni, su SQL, ma qui verranno illustrati tutti i comandi più utilizzati... e utili.

I comando SQL vengono dati attraverso delle stringhe.

Per una lettura più veloce e facilmente comprensibile illustreremo i termini principali che incontreremo in questa guida.

Database

I Database sono formati da delle **Tabelle**, ognuna con un proprio nome, che contengono i **Records** formati da righe e colonne. Nelle righe sono presenti una informazione (record) e le colonne sono i singoli dati (parti di una informazione).

Query SQL

Una query Sql ha una sintassi molto semplice, del tipo:

SELECT [records] FROM [tabella]

Dove ciò che è tra parentesi sono dei dati variabili.
Da notare che, a [records] per indicarli tutti basta inserire *

Manipolare i Dati

E' molto semplice, ed è possibile attraverso 4 comandi:

SELECT : Estrae i dati
INSERT : Aggiunge nuovi records (dati)
UPDATE : Aggiorna i dati presenti
DELETE : Cancella i dati presenti

SELECT

Permette di estrarre alcuni (o tutti) i dati di una tabella in base a dei determinati criteri.

Sintassi

SELECT nome_records FROM nome_tabella

Esempio

La tabella "anagrafici"

Nome	Cognome	Età	Città
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

Da ora e in poi, utilizzeremo sempre questa tabella.

SELECT nome FROM anagrafici

Risultato

Lorenzo
Marcello
Michele

Utilizzando invece la query

SELECT * FROM anagrafici

Selezioniamo tutti i records, quindi avremo la riproduzione di tutta tabella con i dati che contiene.

WHERE

Permette di estrarre alcuni dati in base a criteri determinati appunto dalla clausola WHERE

SELECT records FROM tabella WHERE records CONDIZIONE valore

Esempio

La tabella "anagrafici"

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

SELECT nome FROM anagrafici WHERE eta = 17

Risultato

Lorenzo
Michele

Condizioni

= uguale
< minore
> maggiore
<> diverso
<= minore o uguale
>= maggiore o uguale
LIKE contiene

Importante

Nella clausola WHERE se il record deve essere uguale a del testo, allora il testo deve essere incluso tra gli apici ', in questo modo:

SELECT nome FROM anagrafici WHERE citta = 'Milano'

mentre se il record deve essere uguale ad un numero, gli apici non devono essere utilizzati.

SELECT nome FROM anagrafici WHERE eta = 17

Like

Like permette di ricercare nei records parole, numeri o parti di essi (iniziali, centrali e finali). La sintassi è leggermente diversa, ma molto semplice:

SELECT nome FROM anagrafici WHERE nome LIKE 'm'

In questo caso vengono dati come risultati tutti i dati, che sono presenti nella colonna nome e che iniziano per M.

Per cercare invece nomi che finiscono per M si utilizza la seguente query:

SELECT nome FROM anagrafici WHERE nome LIKE '%m'

Infine per cercare nomi che finiscono, iniziano o che contengono la M, si utilizza la seguente query:

SELECT nome FROM anagrafici WHERE nome LIKE '%m%'

INSER INTO

Insert Into viene utilizzato per inserire uno o più records nuovi.

Sintassi

INSERT INTO tabella VALUES (valore1, valore2, ...)

mentre se si vogliono inserire dei dati solo in alcune colonne:

INSERT INTO tabella (colonna1, colonna2) VALUES (valore1, valore2)

Esempio

La tabella "anagrafici"

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano

INSERT INTO anagrafici VALUES ('Michele', 'Basso', 17, 'Udine')

Risultato

Sarà inserita una nuova riga con i dati:

Nome	Cognome	Eta	Citta
Michele	Basso	17	Udine

UPDATE

Update viene utilizzato per aggiornare i records.

Sintassi

UPDATE tabella SET colonna = valore WHERE colonna = valore

Esempio

La tabella "anagrafici"

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

UPDATE anagrafici SET eta = 22 WHERE cognome = 'Tansini'

Risultato

Sarà aggiornato il record inerente a tutti i Tansini (in questo caso solo Marcello) modificando, in questo caso, solo l'età da 21 a 22.

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	22	Milano
Michele	Basso	17	Udine

DELETE

Delete viene utilizzato per cancellare uno o più records.

Sintassi

DELETE FROM tabella WHERE colonna = valore

Esempio

La tabella "anagrafici"

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

DELETE FROM anagrafici WHERE cognome = 'Pascucci'

Risultato

Sarà cancellata la riga che contiene i dati:

Lorenzo Pascucci 17 Oriolo Romano

Nome	**Cognome**	**Eta**	**Citta**
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

Istruzioni 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 singola)
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
```

Ottimizzazione delle Query

Molto spesso l'efficienza di una applicazione, sia essa una pagina ASP/JSP/PHP o un vero programma, dipende dall'efficienza del sottostante database. Ma un database efficiente non serve a nulla se le query che facciamo sono il massimo dell'inefficienza. Come si puo' migliorare l'efficienza di una Query?

Come funziona il "motore del database" ?

Per poter utilizzare in modo efficiente il motore del database, dobbiamo sapere prima come funziona. Cioe' come le nostre query vengono elaborate e trasformate in una sequenza di record.

Questo articolo presenta i passi necessari per la "trasformazione ed elaborazione delle Query", inoltre verranno discussi i vari metodi di ottimizzazione in uso.

Vi sono molti testi sul soggetto, e quasi tutti sono in completo disaccordo sul metodo esatto di come applicare una ottimizzazione, molti sono in disaccordo anche su cosa e' una "ottimizzazione".

Attenzione: le informazioni su cui questo articolo si basa sono largamente basate sul motore di elaborazione di MS Sql Server, altri database potrebbero utilizzare metodi di trasformazione diversi e quindi impiegare processi di ottimizzazione molto diversi. Riferirsi alla documentazione del produttore del database in caso di problemi o dubbi.

Transformazione delle Query

Quando una query SQL e' inviata al "motore" di un database (RDBMS), questo effettua diversi passaggi per trasformare la query in una sequenza di record che rispondano a determinate caratteristiche.

I passaggi possono essere diversi per query che NON ritornano risultati (query di comando), noi ci concentreremo solo sulle query che ritornano dei risultati (query di interrogazione), dato che sono queste che piu' spesso richiedono ottimizzazione.

Parsing

Una volta che la query viene ricevuta dal motore del database, il primo passaggio e' il "parsing", cioe' la separazione della query nelle sue componenti. Questo processo ha due funzioni principali:

1. verificare la correttezza della query
2. identificare tutte le parti che compongono la query

Ogni singolo "pezzo" della query e' identificato e memorizzato in una struttura interna al motore di database, solitamente nella forma di un albero (query tree).

Un albero e' una rappresentazione che puo' essere facilmente manipolata dal sistema interno, aggiungendo, rimuovendo o spostando le sue componenti.

Standardizzazione

Uno dei punti di forza di un database relazionale e' l'abilita' di accettare query da utenti che non hanno una elevata comprensione della sottostante struttura del database, come risultato, una query puo' essere molto complessa, il sistema deve essere in grado di "risolvere" svariate combinazioni di istruzioni ed identificare i risultati corretti.

Il processo di "standardizzazione" e' di trasformare la query in un formato piu' comprensibile al motore stesso. Questo si effettua mediante una serie di manipolazioni sull'albero di query costruito precedentemente. Durante questo processo, vengono rimosse tutte le clausole eventualmente ridondanti e l'intero albero viene riarrangiato. L'albero risultante viene passato all'ottimizzazione.

Ottimizzazione della query

Lo scopo del processo di ottimizzazione della query e' produrre un "piano di esecuzione" il piu' possibile efficiente, basandosi su quanto e' specificato dall'albero della query.

Un "ottimizzatore" teoricamente puo' produrre un piano di esecuzione "ottimale" per ogni query, in realta' questo finira' per produrre un piano solamente accettabile per la maggioranza delle query. Questo perche' il numero di combinazioni possibili in una Join aumenta geometricamente e nello stesso modo aumenta la complessita' della query.

Senza l'utilizzo di tecniche di "pruning" o altri metodi euristici per limitare il numero di combinazioni valutate, il tempo richiesto per ottenere una reale ottimizzazione della query risulta assolutamente inaccettabile. In molti casi l'ottimizzatore sceglie una query meno efficiente (o totalmente inefficiente) perche' la selezione di una query piu' efficiente richiede piu' tempo che l'esecuzione della query inefficiente.

I vari database utilizzano di solito differenti tecniche di ottimizzazione per ottenere una certa efficienza nel piano di esecuzione.

Ottimizzazione euristica
Questo e' un meccanismo basato su regole specifiche per produrre un piano di esecuzione efficiente. Dato che la query ricevuta e' una struttura definita, ogni nodo dell'albero viene mappato direttamente in una espressione algebrica-relazionale. La funzione euristica e' quindi applicata per ridurre l'espressione ai suoi termini di base, ottenendo quindi una rappresentazione piu' efficiente.

Utilizzando un'espressione algebrica si assicura anche che nessuna delle necessarie informazioni richieste per esaminare i dati verra' persa durante il processo.

Ottimizzazione sintattica
Questo tipo di ottimizzazione si appoggia pesantemente sulla comprensione dell'utente sia del database sottostante, sia della distribuzione dei dati tra le varie tabelle. In soldoni, si fa' affidamento sul fatto che l'utente ha gia' fatto delle scelte in base alle proprie conoscenze. L'ottimizzatore cerca di migliorare l'efficienza della query scegliendo gli indici appositi tra quelli disponibili per ogni singola tabella.

Questo tipo di ottimizzazione e' estremamente efficiente quando si accede a dati in un'ambiente sostanzialmente statico e quando l'utente sa quello che sta' facendo. Ovviamente, se il database ed i suoi contenuti cambiano in maniera molto varia o se l'utente non sa esattamente cosa richiedere (la query non e' gia' ottimizzata di suo), i risultati possono essere pessimi.

Ottimizzazione "Cost-Based"
Per eseguire questo tipo di ottimizzazione, l'ottimizzatore richiede informazioni specifiche relative alle informazioni del database stesso. Queste informazioni sono strettamente dipendenti dal sistema e possono includere cose come la dimensione dei files, la struttura degli stessi, la disponibilita' di indici, la percentuale di record da recuperare da ogni tabella etc. etc.

Dato che lo scopo di ogni ottimizzazione e' quello di ridurre al minimo il numero di record estratti ed il tempo di estrazione, l'ottimizzazione cost-based utilizza le informazioni sulla struttura del database e la distribuzione dei dati per assegnare un "costo" stimato, in termini di tempo e numero di record da estrarre da ogni tabella, numero di accessi etc. per ogni operazione.

Valutando la somma totale di questi "costi" e' possibile selezionare la sequenza piu' efficiente di estrazione dei dati.

Ovviamente, i "costi" assegnati saranno piu' o meno validi a seconda delle informazioni che il sistema ha/mantiene sulla composizione delle tabelle, dei files etc. etc. Mantenere aggiornate queste informazioni occupa tempo e risorse, quindi ogni sistema memorizza un blocco di informazioni e poi lo aggiorna (ricostruendolo) di tanto in tanto. Se sul database

vengono effettuate molte operazioni che coinvolgono la distruzione totale e la ricostruzione da zero di svariate tabelle, l'efficienza di questo metodo di ottimizzazione e' seriamente compromessa.

Ottimizzazione semantica
Questo tipo di ottimizzazione non e' ancora entrata nel novero delle ottimizzazioni "standard", ma e' oggetto di ricerche. Questo metodo si basa sulla conoscenza della struttura del sottostante database per ignorare o eliminare parti della query che non ritornerebbero risultato o non ritornerebbero risultati utili.

Selezione degli indici

Per ottimizzare una query, la maggior parte degli ottimizzatori verifica se nel database sono presenti degli indici utili per migliorare l'efficienza di accesso ai dati. Un indice e' considerato "utile" solo se inizia con le stesse colonne (campi) che sono contenute nella query. Questa deve essere una corrispondenza esatta.

Selezione delle Join

Quando gli indici sono stati scelti e tutte le clausole sono state associate ad un "costo di processo", l'ottimizzatore esegue la selezione delle Join. Questo e' un tentativo di scegliere il migliore ordine per combinare le varie clausole.

L'ottimizzatore confronta vari ordinamenti delle clausole, quindi seleziona quello con il minor tempo di processo stimato.

La maggior parte dei database (compreso SQL Server), utilizza una ottimizzazione cost-based. Perche' ? Perche' e' molto piu' semplice da implementare piuttosto che una Euristica.

Come ottimizzare la query?

Ok, adesso sappiamo tutto di come funziona il nostro motore "sotto al cofano", come usiamo queste informazioni a nostro vantaggio?

Ovviamente, fornendo al "motore" delle query che vengano interpretate correttamente e che diano meno dubbi possibile su quello che il motore deve fare, inoltre si puo' agire anche sulla struttura del database, riducendo al minimo i problemi che esso puo' incontrare.

Indici

Obbligare il motore a scegliere gli indici giusti (o a scegliere gli indici in assoluto invece di passare tutta la tabella come fa' a volte). Questo si fa' sia **forzando** l'uso di un certo indice mediante la clausola

```
SELECT .. FROM table WITH (INDEX n) ...
```
nella SQL, sia costruendo "ad hoc" le query, come spiegato di seguito.

In alcuni casi (tabelle di piccole dimensioni) e' opportuno forzare il motore ad effettuare un TableScan.

Clausole WHERE

Un'espressione del tipo

```
WHERE campo >= valore * 12 + 3000   o
WHERE SUBSTRING( campo, 1, 1) = 'C'
```
non risultera' nella selezione di un indice, anche se un indice e' disponibile per quel campo.

Una clausola del tipo

```
WHERE NOT campo = '...' o
WHERE CAMPO != '...'
```

Non verra' usato per la selezione di un indice, quindi evitare se possibile l'utilizzo di questo costrutto.

Un'espressione del tipo

```
WHERE CAMPO = 'valore'
```
e' l'optimum per la selezione se un indice unico e' disponibile. Un range

```
BETWEEN o
WHERE CAMPO >='..' AND CAMPO < '..'
```
viene subito dopo come termini di efficienza. Un'espressione aperta come

```
WHERE campo < 'valore'    o
WHERE campo > 'valore'
```
e' il peggio che si possa avere come uso degli indici.

Clausole da evitare

Evitare dove possibile clausole **OR** o **IN**, in quanto questo tipo di "disgiunzioni" portano quasi sicuramente alla creazione di una tabella temporanea per risolvere la clausola, quindi un elevato input/output del database stesso. Inoltre, se per risolvere una clausola **OR** viene richiesto un TableScan (nessun uso di indice), lo stesso verra' usato per tutta la query.

Tabelle temporanee

Una tabella temporanea e' creata per risolvere i problemi dati da

- GROUP BY
- ORDER BY se nessun indice e' disponibile o se l'ordine e' in conflitto con il SORT ORDER del server stesso
- SELECT INTO
- DISTINCT
- Subquery

Se nessun indice e' disponibile per una Join, verra' costruita una tabella temporanea per la piu' piccola delle due tabelle della Join, ed un indice clustered verra' costruito sulla tabella temporanea.

Progettare il database e l'applicazione

Una parte di ottimizzazione si fa' anche progettando BENE il database, prima ancora di fare una sola Select e' possibile lavorare per avere un database ottimizzato.

Evitare (se possibile) aggiornamenti multipli sulla stessa "pagina", questo avviene quasi sempre sulle tabelle non-indicizzate, in quanto tutte le aggiunte sono fatte sull'ultima pagina "aperta".

Evitare transazioni che richiedano l'intervento dell'utente, questo richiede il mantenimento di Lock per tutta la durata della transazione, aumentando la possibilita' che si abbiano degli accessi concorrenti.

Usare Stored Procedures e View per accedere ai dati dove possibile, le SP e le Views sono compilate all'interno del server, quindi il "piano di esecuzione" e' gia' memorizzato e non deve essere interpretato ogni volta.

Creare gli indici "giusti" sulle varie tabelle. Ogni indice deve essere mantenuto, quindi evitare di creare indici su tutti i campi ma creare solo quelli essenziali. Ricordarsi che se la sequenza dei campi non corrisponde ESATTAMENTE a quanto richiesto, l'indice non viene usato.

Evitare (dove possibile) le OUTER JOIN (LEFT JOIN,RIGHT JOIN), soprattutto su tabelle in cui i campi di Join possono contenere NULL.

Controllare se le cose funzionano

Utilizzando **SET SHOWPLAN ON**, si puo' controllare se la Query che abbiamo ottimizzato e' stata ottimizzata come noi avevamo previsto. In caso contrario, e' il caso di ripensare le nostre scelte e scegliere (magari) un diverso modo di ottimizzazione.

Sql: Backup delle tabelle di un database con l'istruzione INTO

L'istruzione **INTO** di Sql permette di effettuare il backup di una tabella all'interno del database stesso in cui è contenuta la tabella da salvare, oppure (consigliabile) all'interno di un database utilizzato apposta per i backup.

Per effettuare qualche prova creeremo due file di database Access sotto la partizione C, ovvero

```
C:\database.mdb     # DB principale
C:\backup.mdb       # DB per le copie di backup
```

Si crei all'interno di database.mdb la tabella **utenti** composta dai campi **id** (Contatore), **nome** (Testo) e **cognome** (Testo) e la si popoli con dei dati di prova.

Il seguente codice crea la tabella **backup** all'interno di database.mdb stesso

```
SELECT * INTO backup FROM utenti;
```

Il seguente codice, invece, crea la tabella **utenti** all'interno del database di appoggio backup.mdb

```
SELECT * INTO utenti IN 'C:\backup.mdb' FROM utenti;
```

E' possibile creare delle condizioni di ricerca per filtrare i dati da salvare, utilizzando le istruzioni e le funzioni già studiate nelle documentazioni della sezione SQL del sito.

BETWEEN….AND

Between ... And serve a prelevare dei dati compresi tra due valori.

Sintassi

SELECT records FROM tabella WHERE records BETWEEN valore1 AND valore2

Esempio

La tabella "anagrafici"

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

SELECT nome FROM anagrafici WHERE nome BETWEEN Lorenzo AND Michele

Risultato

Marcello

DISTINCT

Distinct serve a non ripetere nei risultati lo stesso valore.

Sintassi

SELECT DISTINCT records FROM tabella

Esempio

La tabella "anagrafici"

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

SELECT DISTINCT eta FROM anagrafici

Risultato

17
21

L'età di Michele (17 anni come quella di Lorenzo) non viene data come risultato in quanto già esiste un risultato dello stesso valore. La stessa cosa vale per le parole...

è differente dal semplice:

SELECT eta FROM anagrafici

in quanto questa istruzione sql avrebbe dato come risultato:

17
21
17