MySQL: DataManipulationLanguage

Prof. Viglietti Francesco

Www.in4matika.altervista.org

Introduzione

Questo modulo descrive come inserire, cancellare e modificare i dati nella vostra base di dati MySQL attraverso i comandi INSERT, DELETE e UPDATE.

In particolare si parla di:

- uso di INSERT
- uso di DELETE
- uso di UPDATE
- inserimento dei dati con il comando LOAD DATA INFILE
- uso delle estensioni REPLACE e TRUNCATE.

Si stanno quindi discutendo gli aspetti DML (Data Manipulation Language) di SQL.

Dopo aver imparato come inserire i dati in una base di dati, nel seguito vedremo quanti e quali modi ci sono di ritrovare i dati in una base di dati.

BACKUP

Per fare il backup di un DB esistente digitare, dal prompt di comando e dalla directory di mySql, il seguente comando:

mysqldump –u root –p impiegato > prova-backup.sql

Dove impiegato è il nome del DB in mySql e prova-backup.sql è il file di testo che memorizzerà il backup del DB.

Il controllo del contenuto del DB può essere fatto scrivendo dal prompt di comando il seguente comando:

type prova-backup.sql dove prova-backup.sql è il file di backup del DB

RIPRISTINO

Per fare il ripristino di un DB, dall'interno del monitor di mysql, bisogna prima creare il DB digitare:

Create database impiegato;

Quindi uscire dal monitor di mysql e fare il restore del file di testo:

mysql -u root -p[root_password] [database_name] < dumpfilename.sql

dove [database_name] rappresenta il nome del DB che andremo a ripristinare impiegato nel nostro caso, e dumpfilename.sql rappresenta il nome del file di testo che contiene le informazioni prova-backup.sql nel nostro caso.

Uso di *INSERT*

Il comando INSERT si usa per inserire righe in una tabella.

Per fissare i concetti riferiamoci come al solito ad un esempio concreto, sempre legato alla solita base di dati:

use impiegato;

delete from dipartimento;

insert into dipartimento values (42, 'Finanza'), (128, 'Ricerca e Sviluppo'), (NULL, 'Risorse Umane'), (NULL, 'Vendite');

Uso di *INSERT*

```
delete from impiegato;

insert into impiegato values (7513,'Nora Edwards',Programmatore',128),
    (9842, 'Ben Smith', 'DBA', 42),  (6651, 'Ajay Patel', 'Programmatore',
    128), (9006, 'Candy Burnett', 'Amministratore di Sistema', 128);

delete from impiegatoCompetenze;

insert into impiegatoCompetenze values (7513, 'C'), (7513, 'Perl'),
    (7513, 'Java'), (9842, 'DB2'), (6651, 'VB'), (6651, 'Java'), (9006, 'NT'),
    (9006, 'Linux');
```

Uso di *INSERT*

delete from client;

insert into client values (NULL, 'Telco Inc', '1 Collins St Melbourne', 'Fred Smith', '95551234'), (NULL, 'The Bank', '100 Bourke St Melbourne', 'Jan Tristan', '95559876');

delete from assignment;

insert into assignment values (1, 7513, '2003-01-20', 8.5);

Si noti come ancora una volta si usa un comando DELETE prima di inserire i dati in ciascuna tabella; non è strettamente necessario, ma elimina eventuali dati di prova che potete aver introdotto fino a questo momento.

Si osservi anche come i dati inseriti si adattino al database di esempio. Inoltre sono state inserite alcune righe in più.

Uso di ***INSERT***

Nell’esempio tutti i comandi INSERT si assomigliano. Diamo un’occhiata al primo per vedere coma funziona:

use impiegato;
delete from dipartimento;
insert into dipartimento values (42, 'Finanza'), (128, 'Ricerca e Sviluppo'), (NULL, ‘Risorse Umane'), (NULL, ‘Vendite');

Nella prima linea viene specificato il nome della tabella in cui si vogliono inserire i dati; in questo caso, dipartimento. In questo esempio si stanno includendo nella tabella quattro righe. Vi ricorderete che la tabella dipartimento ha due colonne, IDdipartimento e nome (si può fare un controllo digitando il comando describe dipartimento).

Uso di *INSERT*

Nelle prime due righe è indicato quale IDdipartimento si intende usare. Tornando un attimo indietro alla definizione di IDdipartimento, ricorderete che nello scorso capitolo è stato definito come:

IDdipartimento int not null auto_increment primary key

Dal momento che si tratta di una colonna auto_increment, potete specificare voi il valore, o lasciare che sia MySQL a calcolarne uno automaticamente (di solito, in casi come questo, si lascia che sia MySQL ad allocare un numero, ma possono esserci situazioni in cui esiste già un valore che si vuole usare).

Noterete certamente che nelle righe di 'Risorse umane e Vendite' è stato indicato IDdipartjmento come valore NULL. Questa scelta permette ad auto_increment di intervenire e calcolare un valore da inserire.

Uso di *INSERT*

Osservando i vari comandi INSERT, vedrete che sono inseriti i dati string o date type tra apici, ad esempio ‘Ricerca e Sviluppo’. Quando invece il dato è numerico gli apici non vanno usati.

Cosa fare quando il dato possiede degli apici? La risposta è che occorre segnalare l’apice. In parole povere, dovete inserire un backslash (\) davanti all’apice, ad esempio ‘O\’Leary’.

Ovviamente questo conduce alla domanda: “cosa facciamo se vogliamo che il carattere backslash sia semplicemente un carattere backslash, senza alcun significato speciale?” In questo caso dovete segnalare il backslash nello stesso modo, premettendo do un altro backslash \\.

Uso di ***INSERT***

Ritroviamo i dati nella base di dati usando il comando SELECT. Si parlerà di SELECT in modo esaustivo nel seguito. Per il momento, dovete soltanto sapere che digitando:

select * from nome_tabella;

si ritrovano tutti i dati memorizzati al momento nella tabella.

Se digitate:

select * from dipartimento;

comparirà sul video una schermata tipo:

```
+----------------+---------------------+
| IDdipartimento | nome                |
+----------------+---------------------+
|             42 | Finanza             |
|            128 | Ricerca e Sviluppo  |
|            129 | Risorse Umane       |
|            130 | Vendite             |
+----------------+---------------------+

+4 rows in set (0.01 sec)
```

Noterete che l'azione di auto_increment produce un dato di una unità più grande valore maggiore presente in quel momento nella colonna.

Uso di *INSERT*

La sintassi generale del comando INSERT è la seguente:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] nome_tabella [(nome_col,...)]
       VALUES ((espressione | DEFAULT),...),(...),...
       [ ON DUPLICATE KEY UPDATE nome_col=espressione, ... ]
```

oppure

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] nome_tabella [(nome_col,...)]
       SELECT ...
```

oppure

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] nome_tabella
SET nome_col=(espressione | DEFAULT), ...
[ ON DUPLICATE KEY UPDATE nome_col=espressione, ... ]
```

Uso di ***INSERT***

Tutti gli esempi visti seguono la prima forma sintattica dell'espressione. Si osservi come la parola chiave INTO sia opzionale. Si sarebbe potuto ometterla e iniziare l'interrogazione con insert impiegato values, ma sarebbe stata meno intuitiva da leggere.
Usando questa forma si devono elencare i valori di ciascuna colonna per ciascuna riga seguendo lo stesso ordine che hanno le colonne nella tabella. Ad esempio, è stato indicato prima IDdipartimento e poi il nome, perché questa è la struttura della tabella dipartimento. Come già dimostrato, questa forma permette di inserire più righe con un solo comando INSERT.
La seconda forma sintattica termina con un comando di SELECT. Invece di immettere i dati manualmente, questa forma permette di estrarre i dati da una o più tabelle della base di dati e inserirli in questa tabella.

Uso di *INSERT*

La terza permette di specificare in quale colonna occorre inserire i dati. Un esempio di uso del comando INSERT in questo modo è:

insert into dipartimento
set nome = Gestione_Risorse

Questa forma sintattica permette di introdurre soltanto una singola riga alla volta, ma non obbliga a mettere valori per tutte le colonne della tabella. In questo caso si sta inserendo un valore solo per il nome. Tutte le altre colonne prenderanno il valore di default, se ne hanno uno, oppure NULL. In questo esempio, IDdipartimento assumerà valore NULL costringendo auto_increment a intervenire creando un nuovo numero di dipartimento (potete verificano digitando select * from dipartimento).

Uso di ***INSERT***

Esistono alcune proprietà opzionali nel comando INSERT:
è possibile specificare che INSERT sia LOW PRIORITY o DELAYED. Entrambe le opzioni ritardano l'operazione di inserimento fino al momento in cui nessun client usa la tabella in lettura. La differenza tra le due è che LOW PRIORITY causa il blocco del client che sta facendo l'inserimento del dato. Se eseguite un inserimento LOW PRIORITY, potreste dover aspettare prima di poter continuare con le interrogazioni sul vostro client. Con DELAYED, vi viene segnalato un OK e potete continuare a eseguire le vostre interrogazioni, ma l'inserimento non verrà eseguito finché la tabella è ancora in uso;
l'opzione IGNORE è molto utile quando si inseriscono più righe in una volta. Normalmente, se una delle righe va in conflitto con una PRIMARY KEY o un valore UNIQUE di una riga già esistente si genera un errore e l'inserimento viene annullato. Se si indica IGNORE, l'errore viene ignorato e l'operazione di inserimento continua;
è possibile indicare che una colonna deve assumere il suo valore di default digitando DEFAULT come valore da inserire nella colonna stessa;

Uso di ***INSERT***

l'opzione ON DUPLICATE KEY UPDATE vi permette di risolvere i conflitti sulle chiavi primarie o sui valori UNIQUE. L'opzione va completata con un comando di UPDATE che modifica il valore della chiave primaria o UNIQUE esistente che genera il conflitto con la nuova riga in modo da eliminarlo.

Il breve esempio di seguito mostra un modo frequente di usare l'opzione ON DUPLICATE KEY UPDATE:

```
create table allarme (IDimpiegato int primary key not null references
impiegato (IDimpiegato) contatore int default 1) ;

insert into allarme(IDimpiegato) values (6651)
    on duplicate key update contatore=contatore+1;
```

Uso di *INSERT*

Questa opzione è molto utile per quelle situazioni in cui non soltanto si vogliono memorizzare dei valori unici, ma anche eseguire azioni come incrementare un contatore quando compare un valore non unico. Un buon esempio potrebbe essere qualsiasi log o giornale, ma per rimanere alla base di dati che si sta usando, si memorizzino gli impiegati che abbiano un allarme nella tabella allarme. Per registrare un allarme di un impiegato, dovete eseguire un comando di INSERT.

Dal momento che il contatore ha valore standard 1 e voi non specificate un valore preciso nel comando di INSERT, la prima volta che questo comando viene eseguito per ciascun impiegato il contatore varà 1.

Gli inserimenti successivi sullo stesso impiegato faranno scattare l'opzione ON DUPLICATE KEY UPDATE incrementando il contatore.

Uso di ***REPLACE***

Il comando REPLACE funziona esattamente come il comando INSERT, ma, qualora si verificasse un conflitto su una chiave, la nuova riga andrebbe a sostituire la vecchia. La sintassi generale di REPLACE è:

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] nome_tabella [(nome_col,...)]
        VALUES (espressione,...),(...),...
```

oppure

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] nome_tabella [(nome_col,...)]
        SELECT ...
```

oppure

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] nome_tabella
        SET nome_col=espressione, nome_col=espressione,...
```

La somiglianza con INSERT dovrebbe essere ovvia

Uso di *DELETE*

Il comando DELETE permette di eliminare le righe da una tabella. Ci sono alcuni comandi di DELETE nell'esempio precedente, ad esempio:

delete from dipartimento;

Il comando DELETE, scritto in questo modo, elimina tutte le righe della tabella dipartimento.

Si possono specificare quali righe vanno eliminate con l'ausilio di una condizione WHERE, ad esempio:

delete from dipartimento where nome = Gestione_risorse;

Questo comando elimina soltanto le righe che rispettano quanto espresso nella condizione WHERE. In questo caso, verranno cancellate soltanto le righe il cui nome è 'Gestione_risorse'.

E' raro voler cancellare tutte le righe di una tabella. Comunque, visto che questa è la forma più semplice del comando delete, potrebbe capitarvi di digitarla senza alcuna condizione WHERE.

Uso di ***DELETE***

La sintassi generale del comando DELETE è la seguente:

```
DELETE [LOW_PRIORITY] [QUICK] FROM nome_tabella
       [WHERE definizione_condizioni]
       [ORDER BY ...]
       [LIMIT righe]
```

oppure

```
DELETE [LOW_PRIORITY] [QUICK] nome_tabella[.*] [,nome_tabella[.*] ...]
       FROM tabelle_di_riferimento
       [WHERE definizione_condizioni]
```

oppure

```
DELETE [LOW_PRIORITY] [QUICK]
       FROM nome_tabella[.*] [,nome_tabella[.*] ...]
       USING tabelle_di_riferimento
       [WHERE definizione_condizioni]
```

Uso di ***DELETE***

La prima forma è quella vista fino ad ora negli esempi. Le altre due servono per cancellare righe da una o più tabelle che richiamano altre tabelle. Ad esempio:

delete impiegato, impiegatoCompetenze from impiegato, impiegatoCompetenze, dipartimento where impiegato.IDimpiegato = impiegatoCompetenze.IDimpiegato and impiegato.IDdipartimento = dipartimento.IDdipartimento and dipartimento.nome = Finanza;

Questo esempio elimina tutti gli impiegati che lavorano per il Dipartimento Finanza e cancella tutte le righe delle loro competenze. Notate come le righe siano eliminate sia dalla tabella impiegato che da quella impiegatoCompetenze (cioè le tabelle elencate all'inizio nella condizione WHERE), ma non da dipartimento (perché viene elencata solo nella parte FROM della condizione).

Le tabelle che compaiono all'inizio del comando delete si ritroveranno con alcune righe eliminate, mentre le tabelle elencate nella condizione FROM vengono usate solo per cercare dei dati e non subiranno alcuna cancellazione anche se sono presenti nella struttura del comando.

Uso di ***DELETE***

Si osservi che questo è un esempio abbastanza complesso perché coinvolge tre tabelle! Ritorneremo a rivedere questa condizione WHERE dopo il join.
In WHERE sono stati utilizzati un paio di concetti nuovi: l'operatore AND, per collegare (join) insieme le nostre condizioni, e la notazione tabella.colonna. Si tratta un semplice AND booleano.
E stata usata anche la notazione impiegato.IDimpiegato che indica "la colonna IDimpiegato della tabella impiegato". Rivedrete meglio questi concetti quando tratteremo le interrogazioni.
La terza forma del comando DELETE è simile alla seconda, eccetto che, in questo caso, si eliminano le righe delle tabelle elencate nella condizione FROM facendo riferimento alle tabelle della condizione USING:

```
delete from impiegato, impiegatoCompetenze
using impiegato, impiegatoCompetenze, dipartimento
where impiegato.IDimpiegato = impiegatoCompetenze.IDimpiegato
and impiegato.IDdipartimento = dipartimento.IDdipartimento
and dipartimento.nome = Finanza
```

Questo esempio è equivalente al precedente, ma usa una forma sintattica alternativa.

Uso di *DELETE*

Nella sintassi generale del comando DELETE ci sono anche altre proprietà opzionali:

- l'opzione LOW PRIORITY funziona nello stesso modo che in INSERT;
- specificare QUICK velocizza l'esecuzione del comando DELETE dicendo a MySQL di non fare i controlli sugli indici mentre elimina le righe dalla tabella;
- l'opzione ORDER BY specifica l'ordine in cui eliminare le righe. È utile unito all'opzione LIMIT; ad esempio potreste voler cancellare le n righe più vecchie nella tabella;
- l'opzione LIMIT vi permette di indicare il numero massimo di righe da cancellare. È utile se usato ad esempio con l'opzione ORDER BY per impedirvi di cancellare accidentalmente troppe righe.

Uso di ***TRUNCATE***

Il comando TRUNCATE permette di eliminare tutte le righe di una tabella:

TRUNCATE TABLE impiegato;

Questa istruzione elimina tutti gli impiegati dalla tabella impiegato. È più veloce del comando DELETE perché funziona eliminando l'intera tabella in una volta sola e ricreandola vuota. Una cosa da tenere bene a mente è che l'effetto di TRUNCATE non può essere ripristinato in caso di fallimento della transazione.

Uso di *UPDATE*

Si può usare il comando UPDATE per modificare delle righe già presenti nella base di dati. Ad esempio, uno degli impiegati cambi mansione:

update impiegato
set mansione = DBA
where IDimpiegato = 6651

Il comando modifica il valore della colonna mansione per l'impiegato 6651.
La sintassi generale del comando UPDATE è la seguente:

```
UPDATE [LOW_PRIORITY] [IGNORE] nome_tabella
       SET nome_col1=espr1 [, nome_col2=espr2 ...]
       [WHERE definizione_condizioni]
       [ORDER BY ...]
       [LIMIT righe]
```

oppure

```
UPDATE [LOW_PRIORITY] [IGNORE] nome_tabella [,nome_tabella ...]
       SET nome_col1=espr1 [, nome_col2=espr2 ...]
       [WHERE definizione_condizioni]
```

Uso di ***UPDATE***

Per molti aspetti UPDATE è simile a DELETE. E' possibile usare una condizione WHERE per modificare alcune righe particolari o non usarla affatto, cambiando così tutte le righe. Ancora una volta si potrebbe cadere nella trappola di dimenticare di inserire la condizione WHERE ad esempio digitando distrattamente qualcosa del tipo:

update utente
set password = test

La seconda versione della sintassi del comando UPDATE elencata precedentemente consiste in una modifica su più tabelle. Questa funziona come l'eliminazione di righe da più tabelle, che avete visto poco sopra. Notate che verranno modificate soltanto le colonne che avete esplicitamente elencato nell'istruzione SET. Le proprietà LOW_PRIORITY e IGNORE funzionano nello stesso modo che in INSERT; le proprietà ORDER BY e LIMIT esattamente come in DELETE.

Inserimento con LOAD DATA INFILE

Per inserire grandi quantità di dati da un file utilizzare il comando LOAD DATA INFILE. Ad esempio è possibile caricare i seguenti dati nel nostro database di riferimento usando il file department_infile.txt

42 'Finance'
128 'Research and Development'
NULL 'Human Resources'
NULL 'Marketing‘

E digitando il comando:
Load data local infile ‘c:\department_infile.txt’
Into table department;

Il formato richiede che i valori da mettere in una riga siano su una riga del file e che i campi sia separati dal carattere di tabulazione.

Inserimento con LOAD DATA INFILE

E’ possibile adattarsi a formati alternativo compatibile con formati predefiniti. un esempio relativo al formato csv generabile da molti spreadsheet. E’ un formato che prevede i campi suddivise per righe e separati da virgole.

Digitando il comando:

load data local infile ‘c:\new_programmers.csv’
into table employee
fields terminated by ‘,’
lines terminated by ‘\n’
ignore 2 lines
(name, job, departmentID);

si può leggere in input il file new_programmers.csv

Esercizi

1. Scrivere un insieme di comandi INSERT per inserire dati in ciascuna delle tabelle della base di dati ordini
2. Eliminare i dati dalle tabelle