INSERT: inserimento dei dati

L'inserimento dei dati in una tabella avviene tramite l'istruzione **INSERT**. Ovviamente dovremo avere il permesso di INSERT sulla tabella.

Vediamo quali sono le diverse sintassi che possiamo utilizzare:

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] *nome_tabella* [(*nome_colonna*,...)]
VALUES ({*espressione* | DEFAULT},...),(...),...
[ON DUPLICATE KEY UPDATE *nome_colonna=espressione*, ...]

Con questa sintassi possiamo inserire una o più righe nella tabella. Prima della clausola VALUES è possibile indicare i nomi delle colonne interessate dalla INSERT: a questi nomi corrisponderanno i valori compresi in ogni parentesi dopo VALUES. Per inserire più righe useremo più coppie di parentesi tonde dopo VALUES. Se non indichiamo la lista delle colonne, dovremo fornire un valore per ogni colonna della tabella, nell'ordine in cui sono definite.

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] *nome_tabella*
SET *nome_colonna*={*espressione* | DEFAULT}, ...
[ON DUPLICATE KEY UPDATE *nome_colonna=espressione*, ...]

In questo caso usiamo la clausola SET per assegnare esplicitamente un valore ad ogni colonna indicata. Con questa sintassi è possibile inserire una sola riga.

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] *nome_tabella* [(*nome_colonna*,...)]
SELECT ...
[ON DUPLICATE KEY UPDATE *nome_colonna=espressione*, ...]

Qui utilizziamo direttamente una SELECT per fornire i valori alla nuova tabella. Con questo sistema si possono inserire più righe. Come nel primo caso, è possibile elencare le colonne interessate; in caso contrario la SELECT dovrà fornire valori per tutte le colonne.

Vediamo ora la funzione delle varie clausole utilizzabili:

LOW PRIORITY: l'inserimento non viene effettuato fino a quando esistono client che leggono sulla tabella interessata; questo può portare ad attese anche lunghe.

DELAYED: anche in questo caso l'inserimento viene ritardato fino a quando la tabella non è libera. La differenza rispetto al caso precedente è che al client viene dato immediatamente l'ok, e le righe da inserire vengono mantenute in un buffer gestito dal server fino al momento della effettiva scrittura. Ovviamente le righe non saranno visibili con una SELECT fino a quando non saranno inserite sulla tabella.

HIGH PRIORITY: annulla l'effetto di una eventuale opzione –low-priority-updates che fosse attiva sul server.

IGNORE: permette di gestire eventuali errori che si verificano in fase di inserimento (chiavi duplicate o valori non validi); invece di generare errori bloccanti, i record con chiavi doppie vengono semplicemente scartati, mentre i valori non validi vengono "aggiustati" al valore più prossimo.

ON DUPLICATE KEY UPDATE: nel caso in cui si verifichi una chiave doppia, l'istruzione specificata viene eseguita sulla riga preesistente. Con questa opzione non è possibile usare DELAYED.

I **valori** da inserire nella tabella con le prime due sintassi possono essere indicati da costanti o espressioni, oppure richiamando esplicitamente il DEFAULT. Il default viene usato anche per le colonne non specificate. Tuttavia, in **strict mode**, è obbligatorio specificare valori per tutte le colonne che non hanno un default esplicito; in caso contrario si avrà un errore.

Le colonne di tipo AUTO_INCREMENT vengono valorizzate automaticamente indicando NULL (oppure tralasciandole). Per conoscere il valore generato si può usare, dopo l'inserimento, la funzione LAST_INSERT_ID(), che restituisce l'ultimo valore creato nel corso della connessione attuale.

Oltre alla INSERT, MySQL offre l'istruzione **REPLACE**, che è un'estensione allo standard SQL e che consente di sostituire le righe preesistenti con le righe inserite qualora si verifichi una situazione di chiave doppia. In pratica, usando REPLACE, qualora non sia possibile inserire una riga perchè una PRIMARY KEY o un indice UNIQUE esistono già sulla tabella, MySQL cancella la riga vecchia ed inserisce la nuova. Questo comportamento è opposto a quello di INSERT IGNORE, con il quale è la nuova riga ad essere scartata.

Per effettuare una REPLACE dovremo avere i permessi di INSERT e DELETE; le sintassi

sono pressochè identiche a quelle della INSERT; vediamole:

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] *nome_tabella* [(*nome_colonna*,...)]
VALUES ({*espressione* | DEFAULT},...),(...),...
oppure
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] *nome_tabella*
SET *nome_colonna*={*espressione* | DEFAULT}, ...
oppure
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] *nome_tabella* [(*nome_colonna*,...)]
SELECT ...

Un altro modo di inserire dati in una tabella è quello che ci consente di importare un file di testo: **LOAD DATA INFILE**. Vediamone la sintassi:

LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE '*nome_file*.txt'
[REPLACE | IGNORE]
INTO TABLE *nome_tabella*
[FIELDS
[TERMINATED BY '*stringa*']
[[OPTIONALLY] ENCLOSED BY '*char*']
[ESCAPED BY '*char*']
]
[LINES
[STARTING BY '*stringa*']
[TERMINATED BY '*stringa*']
]
[IGNORE *numero* LINES]
[(*nome_colonna_o_variabile*,...)]
[SET *nome_colonna = espressione*,...)]

Le opzioni **LOW_PRIORITY** e **IGNORE** funzionano come per una INSERT. L'opzione **REPLACE** permette all'istruzione di funzionare come una REPLACE.

L'opzione **LOCAL** specifica che il file da leggere si trova sulla macchina del client. Se non è presente, si assume che il file si trovi sulla macchina del server. In quest'ultimo caso è necessario il privilegio FILE per eseguire l'operazione.

Se si usa **FIELDS** è obbligatorio indicare almeno una delle opzioni che la compongono:

- TERMINATED BY indica quale stringa separa i campi;

- ENCLOSED BY indica i caratteri usati per racchiudere i valori;
- ESCAPED BY specifica il carattere di escape usato per i caratteri speciali (cioè quelli utilizzati nelle altre opzioni di FIELDS e LINES)

LINES può indicare le opzioni per le righe: STARTING BY indica una stringa che sarà omessa in ogni riga (può anche non trovarsi all'inizio della riga, nel qual caso sarà omesso tutto ciò che si trova prima).

TERMINATED BY indica il carattere di fine riga. I default per FIELDS e LINES sono i seguenti:

FIELDS TERMINATED BY 't' ENCLOSED BY '' ESCAPED BY '\' LINES TERMINATED BY 'n' STARTING BY ''

Quindi: campi suddivisi da tabulazioni e nessun carattere a racchiuderli; righe senza prefisso e che terminano con il carattere di newline; backslash come carattere di escape (per tabulazioni e newline).

IGNORE *n* LINES si usa per saltare le prime *n* righe del file di input (ad esempio perchè contengono intestazioni).

Può essere indicata una lista di nomi di colonna o variabili alle quali assegnare i valori letti dal file. Se questa lista non viene fornita, si presume che il file contenga in ogni riga un valore per ogni colonna della tabella; se si forniscono variabili, i valori non finiranno direttamente nella tabella, ma potranno essere utilizzati nella clausola **SET** per effettuare elaborazioni; con tale clausola è anche possibile assegnare valori ad altre colonne indipendentemente dai dati presenti nel file (ad esempio un timestamp).

UPDATE e DELETE: modifica e cancellazione dei dati

Dopo aver visto come inserire e cercare i dati, vediamo ora come modificarli o eliminarli. Per gli aggiornamenti si usa l'istruzione **UPDATE**, di cui vediamo la sintassi:

UPDATE [LOW_PRIORITY] [IGNORE] *nome_tabella*
SET *nome_colonna=espressione* [, *nome_colonna2=espressione2* ...]
[WHERE *condizioni*]
[ORDER BY ...]
[LIMIT *numero_righe*]

Il funzionamento è abbastanza intuitivo:

- dopo UPDATE indichiamo quale tabella è interessata
- con SET specifichiamo quali colonne modificare e quali valori assegnare
- con WHERE stabiliamo le condizioni che determinano quali righe saranno interessate dalle modifiche (se non specifichiamo una WHERE tutte le righe saranno modificate)

Inoltre possiamo usare ORDER BY per decidere in che ordine effettuare gli aggiornamenti sulle righe, e LIMIT per stabilire un numero massimo di righe che saranno modificate. Evidentemente l'uso di ORDER BY difficilmente ha senso se non accoppiato con LIMIT.

L'UPDATE restituisce il numero di righe modificate; attenzione però: se tentate di assegnare ad una riga valori uguali a quelli che ha già, MySQL se ne accorge e non effettua l'aggiornamento. Ai fini della LIMIT la riga viene comunque conteggiata.

È possibile anche usare LOW_PRIORITY, come già visto per le INSERT, per ritardare l'esecuzione dell'aggiornamento ad un momento nel quale la tabella non è impegnata da altri client. Con la clausola IGNORE invece indichiamo al server di ignorare gli errori generati dall'aggiornamento. Eventuali modifiche che causassero chiavi doppie non saranno, in questo caso, effettuate.

In una UPDATE è possibile fare riferimento ad una colonna per utilizzare il suo valore precedente all'aggiornamento; ad esempio:

UPDATE vendite SET venduto=venduto+1 WHERE idVenditore=5;

In questo caso il valore della colonna *venduto* viene incrementato di 1.

L'operazione di UPDATE può essere effettuata anche su più tabelle. In questo caso indicheremo i nomi delle tabelle interessate con la stessa sintassi già vista per le join. Con gli update multi-tabella però non è possibile usare le clausole ORDER BY e LIMIT.

Per effettuare una UPDATE è necessario avere il privilegio UPDATE sulle tabelle da modificare più il privilegio SELECT su eventuali altre tabelle a cui viene fatto accesso in sola lettura.

Passiamo ora alla **DELETE**, con la quale cancelliamo una o più righe da una o più tabelle (per farlo dobbiamo avere il privilegio DELETE).
Questa è la sintassi per la DELETE su una sola tabella:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM *nome_tabella*
[WHERE *condizioni*]
[ORDER BY ...]
[LIMIT *numero_righe*]

Le opzioni LOW_PRIORITY e IGNORE hanno lo stesso significato visto per la UPDATE. La clausola QUICK è utile solo per tabelle MyISAM: velocizza l'operazione ma non effettua l'ottimizzazione degli indici. È utile se i valori degli indici cancellati saranno sostituiti da valori simili.

Anche ORDER BY e LIMIT funzionano come nella UPDATE: permettono di stabilire l'ordine delle cancellazioni e di limitare il numero di righe cancellate. Con la WHERE stabiliamo le condizioni in base alle quali le righe verranno eliminate. Se non la indichiamo, tutte le righe saranno eliminate.

La cancellazione di righe da una tabella può portare alla presenza di spazio inutilizzato nella tabella stessa: se si effettuano molte DELETE su una tabella sarà bene effettuare periodicamente una OPTIMIZE TABLE oppure eseguire l'utility *myisamchk* (vedere lezione 24).

Vediamo ora come effettuare una DELETE su più tabelle: per questa operazione esistono due possibili sintassi

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
nome_tabella[.*] [, *nome_tabella*[.*] ...]
FROM *tabelle*
[WHERE *condizioni*]
oppure

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM *nome_tabella*[.*] [, *nome_tabella*[.*] ...]
USING *tabelle*
[WHERE *condizioni*]

In questo caso può capitare che abbiamo la necessità di cancellare righe da una o più tabelle leggendo i dati anche da altre tabelle, senza cancellare niente da queste ultime. Le tabelle che subiranno le cancellazioni sono elencate dopo DELETE nella prima sintassi, e dopo FROM nella seconda. La join sulle tabelle da cui leggere i dati viene invece espressa con la clausola FROM per il primo caso, e con USING nel secondo.

Vediamo due esempi equivalenti:

DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id; DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;

In entrambi i casi verranno cancellate le righe corrispondenti da t1 e t2, ma solo quando esiste in t3 un ulteriore valore corrispondente. Come avrete notato, anche per la DELETE multi-tabella non è possibile usare le opzioni ORDER BY e LIMIT.

Quando vogliamo eliminare per intero il contenuto di una tabella possiamo utilizzare l'istruzione **TRUNCATE** che è più veloce:

TRUNCATE [TABLE] *nome_tabella*

Nella maggior parte dei casi, con la TRUNCATE la tabella viene eliminata e ricreata. Questo porta, fra l'altro, alla reinizializzazione dei valori AUTO_INCREMENT.