Laboratorio di Sistemi Informativi

SQL e MySQL: comandi di definizione dei dati

Il linguaggio SQL può essere pensato come composto da due sotto-linguaggi: un linguaggio per la definizione dei dati (*DDL: data definition language*) ed uno per la manipolazione dei dati (*DML: data manipulation language*). In questa lezione studieremo i comandi di definizione dei dati, ovvero quelli che servono a manipolare la strutturadegli schemi e delle tabelle. Sebbene questi comandi siano già stati oggetto di studio nel corso teorico di Sistemi Informativi, noi li analizzeremo con particolare riferimento alla loro implementazione in MySQL.

Comandi per la manipolazione di schemi

MySQL può gestire contemporaneamente più di uno schema. In questo modo lo stesso server può essere utilizzato per più di una applicazione. Ogni applicazione opera su uno schema distinto e, di solito, la gestione di ogni schema è affidata a uno o più utenti differenti.

Nella configurazione di default, gli utenti hanno accesso soltanto agli schemi `test` e `information_schema`, tranne un utente particolare, l'utente `root`, che ha accesso totale al server MySQL e può anche creare nuovi schemi. Quando si fa partire il monitor di MySQL, è possibile specificare il nome utente con il quale collegarsi. Se vogliamo collegarci col nome utente `root` basta usare l'opzione `-u`, in questo modo:

```
mysql -u root
```

In aula informatizzata occorre usare il comando `mysql -u root -h lamp -p`. L'opzione `-p` serve perché l'account è protetto da una password che, ovviamente, bisogna conoscere se si vuole che l'operazione di accesso abbia successo.

È ovvio che in un sistema "vero", il collegamento con nome utente "root" dovrebbe essere protetto da password, altrimenti chiunque può avere accesso indiscriminato a tutti gli schemi.

Quando si entra come utente root, il comando `SHOW SCHEMAS` mostra anche lo schema `mysql`. Questo è uno schema speciale usato internamente da MySQL e di cui noi non ci occuperemo. È possibile manipolare con i comandi:

- `CREATE SCHEMA <nome_schema>`: crea un nuovo schema di nome `<nome_schema>`. A questo schema può accedere soltanto l'utente root. Vedremo in una lezione successiva come concedere ad altri utenti i diritti di accesso allo schema.
- `DROP SCHEMA <nome_schema>`: cancella lo schema `<nome_schema>` e tutte le tabelle presenti al suo interno.
- `SHOW SCHEMAS`: mostra l'elenco degli schemi accessibili dall'utente attuale.

Nelle versioni di MySQL precedenti alla 5, i comandi di cui sopra non esistono. Al loro posto, si possono usare `CREATE DATABASE`, `DROP DATABASE` e `SHOW DATABASES` che, però, non fanno parte dell'SQL standard.

Attenzione. Notare che gli utenti MySQL e gli utenti Linux non coincidono del tutto. Normalmente, se l'utente "pippo" da il comando `mysql`, il monitor si collega al server con nome utente "pippo".

Tuttavia, col parametro `-u`, è possibile cambiare a piacimento il nome utente utilizato per il collegamento. Subito dopo l'installazione del software tutti i nomi utenti sono equivalenti, tranne root.

Comandi per la manipolazione di tabelle

Una volta creato uno schema, è possibile inserire al suo interno tabelle, indici, viste, etc... Esamineremo ora i comandi utili a manipolare le tabelle, lasciando ad un una eventuale lezione futura la manipolazione di altri oggetti.

- `CREATE TABLE <nometabella> ( <definizione tabella> )`: crea una tabella. Il comando è ampiamente trattato nel vostro libro di basi di dati, al quale vi rimando. Tenete conto, però, che MySQL ignora completamente la clausola `CHECK` e, in alcuni casi, anche la clausola `REFERENCES` (all'interno della definizione dei singoli attributi).
- `DROP TABLE <nometabella>`: elimina un tabella.
  Attenzione al fatto che cancellare una tabella con `DROP` non è lo stesso che eliminare il suo contenuto con `DELETE`. Il comando `DELETE FROM <nometabella>` elimina tutto il contenuto della tabella indicata, che però continua ad esistere. È possibile infatti aggiungere nuovi elementi con il comando `INSERT`. Quando una tabella viene cancellata con `DROP`, invece, cessa di esistere, e se necessario bisognerà ricrearla col comando `CREATE`. È la stessa differenza che c'è tra cancellare un file e svuotarlo.
- `ALTER TABLE <nometabella> DROP PRIMARY KEY`: elimina la chiave primaria. Dopo questo comando la tabella non ha più nessuna chiave primaria.
- `ALTER TABLE <nometabella> ADD PRIMARY KEY (<nome campi>)`: aggiunge una chiave primaria alla tabella. I campi specificati tra parentesi formeranno la nuova chiave primaria.
- `ALTER TABLE <nometabella> DROP [COLUMN] <nomecolonna>`: elimina una colonna.
- `ALTER TABLE <nometabella> CHANGE <vecchio_nomecolonna> <nuovo_nomecolonna> <nuono tipo>`: cambia nome e tipo di una colonna.
- `ALTER TABLE <nometabella> ADD [COLUMN] <nome_colonna> <tipo>`: aggiunge un nuovo campo alla tabella.
- `ALTER TABLE <nometabella> RENAME <nuovo nome tabella>`: cambia nome a una tabella.

Piuttosto che imparare a memoria tutte le possibili varianti, si consiglia di consultare la documentazione (o usare il comando `HELP`) quando necessario.

Tipi di Dato

Vediamo ora più in dettaglio acluni dei tipi di dato disponibili in MySQL. Per una trattazione completa si veda il manuale di riferimento di MySQL.

Tipi numerici interi

Sono usati per rappresentare numeri interi positivi e/o negativi. Ne esistono diverse varianti, a seconda del numero di bit destinati alla codifica del numero, e quindi dell'intervallo massimo di valori ammissibili.

- `TINYINT`: 8 bit, -128...127 (non standard)
- `SMALLINT`: 16 bit, -32.768 ... 32.767
- `MEDIUMINT`: 24 bit, -8.338.608 ... 8.338.607 (non standard)
- `INTEGER / INT`: 32 bit, -2.147.483.648 ... 2.147.483.647
- `BIGINT`: 64 bit, -9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807 (non standard)

Con "(non standard)" sono indicati dei tipi che sono supportati da MySQL ma non dallo standard SQL. Se ne sconsiglia pertanto l'uso indiscriminato. Si noti inoltre che il numero di bit utilizzati dai tipi `INTEGER` e `SMALLINT` non è definito dallo standard SQL: DBMS diversi possono usare convenzioni diverse.

Tutti i tipi interi posso essere dichiarati `UNSIGNED` (si tratta anche questa di una estensione non standard di MySQL). In questo caso il valore massimo dei dati aumenta: `TINYINT` passa fa -128.. 127 a 0..255, `SMALLINT` da -32.768..32.767 a 0...65535.. e così via. Il modificatore `UNSIGNED` va messo dopo il tipo e non prima, come di solito si fa in Java e in C. Ad esempio:

```
mysql> CREATE TABLE t (val INT UNSIGNED);
```

In generale, per motivi di efficienza, conviene utilizzare il tipo di dato più piccolo che è in grado di contenere tutti i possibili valori che può assumere un determinato campo. Se siamo interessati alla portabilità del database su altri DMBS, è meglio utilizzare soltanto i tipi di dato previsti dallo standard SQL.

Tipi numerici non interi

Si distinguono due sottotipi: numeri approssimati in virgola mobile e numeri esatti in virgola fissa. Per quanto riguarda i primi, si distinguono in base al numero di bit usati per la rappresentazione, che influisce sia sull'intervallo massimo di valori rappresentabili che sul numero di cifre significative.

- `FLOAT`: 32 bit, singola precisione.
  Valori ammessi: da `-3.402823466E+38` a `-1.175494351E-38`, `0`, e da `1.175494351E-38` a `3.402823466E+38`
- `DOUBLE PRECISION`: 64 bit, doppia precisione.
  Valori ammessi: da`-1.7976931348623157E+308` a `-2.2250738585072014E-308`, `0`, e da `2.2250738585072014E-308` a `1.7976931348623157E+308`

I tipi in virgola fissa sono invece i seguenti:

- `NUMERIC(<m>,<d>)`: numero decimale con m cifre significative, di cui d cifre sono dopo la virgola. Il valore massimo di m è 65, il valore massimo di d è 30.
- `NUMERIC(<m>)`: sinonimo di `NUMERIC(<m>,0)`.
- `NUMERIC`: sinonimo di `NUMERIC(10)`.
- `DECIMAL`: sinonimo di `NUMERIC`, in tutte e tre le forme viste sopra.

Tutti i tipi numerici non interi possono essere dichiarati `UNSIGNED`, ma in tal caso l'intervallo di valori positivi rappresentabili non aumenta come avveniva nel caso dei tipi interi.

A meno di esigenze particolari, è meglio non usare i tipi in virgola mobile, ma solo quelli in virgola fissa. Quelli in virgola mobile possono causare tutta una serie di problemi dovuti alla rappresentazione approssimata dei numeri.

```
mysql> CREATE TABLE tround (v DOUBLE PRECISION);
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO tround VALUES (0.3);
```

```
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tround WHERE v=0.3;
+------+
| v    |
+------+
|  0.3 |
+------+
1 row in set (0.00 sec)

mysql> UPDATE tround SET v=v+1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE tround SET v=v-1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM tround;
+------+
| v    |
+------+
| ?????? |
+------+
1 row in set (0.00 sec)

mysql> SELECT * FROM tround WHERE v=0.3;
Empty set (0.00 sec)
```

Quale valore è presente nella tabella?

Nonostante la tabella contenga 0.3 sia prima che dopo i comandi `UPDATE`, una `SELECT` con condizione `v=0.3` ha successo nel primo caso ma non nel secondo. Questo perché, in realtà, 0.3 è un numero periodico in base 2 (che è la base usata dai computer), l'aritmetica è approssimata e sommare e sottrarre uno può far perdere ulteriormente precisione. I due valori prima e dopo gli `UPDATE` sembrano uguali solo perché il computer li arrotonda prima di visualizzarli, ma la differenza è evidente con il seguente comando:

```
mysql> select v-0.3 from tround;
+----------------------+
| v-0.3                |
+----------------------+
| 5.55111512312578e-17 |
+----------------------+
1 row in set (0.00 sec)
```

Se si ripete l'esperimento usando il tipo `DECIMAL(10,2)` invece di `DOUBLE PRECISION`, tutto funziona regolarmente. Questo perchè `DECIMAL` usa una rappresentazione interna in base 10.

Tipi stringa

- `CHAR(<n>) / CHARACTER(<n>)`: stringa di n caratteri a lunghezza fissa. Se la stringa è più corta, viene riempita di spazi alla fine, n <= 255=2^8-1.
- `VARCHAR(<n>) / CHARACTER VARYING(<n>)`: stringa di al massimo n caratteri, n <= 255=2^8-1

I tipi a lunghezzi fissa come `CHAR(<n>)` creano tabelle più efficienti di quelli a lunghezza variabile, a fronte di uno spreco di memoria di massa. Se serve memorizzare delle stringhe più lunghe, si può usare il tipo non standard `TEXT`:

- `TINYTEXT(<n>)`: come `VARCHAR(<n>)`
- `TEXT(<n>)`: come `VARCHAR(<n>)` ma n fino a 65535 = 2^{16}-1

- `MEDIUMTEXT(<n>)`: come `VARCHAR(<n>)` ma n fino a 2^{24}
- `LONGTEXT(<n>)`: come `VARCHAR(<n>)` ma n fino a 2^{32}

Tipi insiemi ed enumerati

Si tratta di tipi non standard, che possono essere usati al posto di tipi stringa o numerici per rendere più leggibile la struttura del database:

- `ENUM('val1','val2',.... 'valn')`: un valore qualunque tra quelli indicati (massimo 65535). In realtà è ammesso anche il valore '' (stringa nulla).
- `SET('val1','val2'... 'valn')`: uno o più valori tra quelli indicati (massimo 64).

Ad esempio:

```
mysql> CREATE TABLE tenum ( vero ENUM('si','no'), colore SET('rosso','verde','giallo') );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tenum VALUES ('si','rosso,verde');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO tenum VALUES ('forse','rosso,indaco');
ERROR 1265 (01000): Data truncated for column 'vero' at row 1

mysql> SELECT * FROM tenum;
+------+-------------+
| vero | colore      |
+------+-------------+
| si   | rosso,verde |
+------+-------------+
2 rows in set (0.00 sec)
```

MySQL rappresenta i valori `ENUM` e `SET` con dei numeri, per cui, quando è possibile, è preferibile riccorre a questi tipi piuttosto che a semplici stringhe. Per i valori `ENUM` usa un numero da 1 e 65535 per ogni valore elencato (e usa 0 per la stringa vuota). Per i valori `SET` usa un numero a 64 bit, dove ogni bit rappresenta un possibile elemento dell'insieme (il bit è a 1 set l'elemento fa parte dell'insieme, a 0 altrimenti).

Tipi per la data l'ora

I tipi per data e ora di MySQL sono abbastanza diversi da quelli dello standard SQL. Ne elenchiamo soltanto due:

- `DATE`: data, nel formato aaaa-mm-gg, dal '0000-00-00' al '9999-12-31'.
- `TIME`: ora, nel formato hh:mm:ss.
- `DATETIME`: ora e data, nel formato aaaa-mm-gg hh:mm:ss, dal '0000-00-00 00:00:00' al '9999-12-31 23:59:59' (non standard).

I valori di tipo `DATE`, `TIME` e `DATETIME` vanno scritti racchiusi tra apici, come fossero delle stringhe. Notate inoltre che, probabilmente per codificare condizioni di errore, sono accettati valori nulli per l'anno, il mese e il giorno. In particolare, il valore 0000-00-00 viene utilizzato da MySQL per indicare una data non valida per il tipo `DATE`. Analogamento per '0000-00-00 00:00:00' nel caso di tipo `DATETIME`. Esempio:

```
mysql> CREATE TABLE tdate ( quando DATE );
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> INSERT INTO tdate VALUE ('2002-10-10');
Query OK, 1 row affected (0.00 sec)
```

mysql> > INSERT INTO tdate VALUE ('0000-00-00');
```
Query OK, 1 row affected (0.040 sec)
```

```
mysql> SELECT * FROM tdate;
+------------+
| quando     |
+------------+
| 2002-10-10 |
| 0000-00-00 |
+------------+
```

Campi AUTO_INCREMENT

Nella tabella `aerei` creata nell'Esercizio 1, abbiamo usato un campo `id` di tipo `char(20)` come chiave primaria. Questo ci ha creato un po' di fastidio, in quanto è stato necessario "inventarsi" delle stringhe da utilizzare come chiave primaria ogni volta che inserivamo un nuovo aereo. In realtà, è quasi sempre preferibile usare chiavi primarie di tipo intero (tranne al più in quei casi in cui le entità hanno degli attributi naturali che possono essere usati come chiavi primarie, ad esempio il codice fiscale), data la loro maggior efficienza. Questo non risolve però il problema di doversi inventare dei valori per appropriati per queste chiavi.

Creiamo ad esempio la seguente tabella:

```
CREATE TABLE tauto (id INT PRIMARY KEY, nome CHAR(20), cognome CHAR(20), nascita DATE);
```

Per inserire dei dati in questa tabella dobbiamo specificare anche il valore di `id`, con comandi del tipo:

```
INSERT INTO tauto VALUES (1,'mario','rossi','2002-12-3');
```

Il problema è che, visto che `id` non ha alcun significato, non è facile decidere un valore appropriato, visto che bisogna stare attenti a sceglierne uno che non è già presente nella tabella. Possiamo semplificarci il lavoro specificando la chiave `id` come AUTO_INCREMENT.

```
ALTER TABLE tauto CHANGE id id INT AUTO_INCREMENT;
```

In questo modo non abbiamo più bisogno di inserire valori per il campo `id`: li genera MySQL automaticamente, quando si tenta di inserire un valore null per la chiave. Ad esempio:

```
INSERT INTO tauto VALUES (null,'mario','bianchi','2000-1-4');
```

oppure

```
INSERT INTO tauto (nome,cognome,nascita) VALUES ('mario','bianchi','2000-1-4');
```

generano il seguente risultato:

```
+--------+----------+---------+------------+
| id     | nome     | cognome | nascita    |
+--------+----------+---------+------------+
|      1 | mario    | rossi   | 2002-12-03 |
|      2 | mario    | bianchi | 2000-01-04 |
+--------+----------+---------+------------+
2 rows in set (0.00 sec)
```

Nota. l'attributo `AUTO_INCREMENT` è una caratteristica esclusiva di MySQL. Altri DBMS usano metodi diversi per risolvere lo stesso problema. Inoltre, solo un campo di tipo intero può essere `AUTO_INCREMENT`.

Esercizi e Soluzioni

Esercizio 1

Si vuole creare una tabella che contiene informazioni sugli aerei di una ipotetica compagnia aerea. I campi di questa tabella devono essere:

1. `id`, di tipo char(20) che funge da chiave primaria
2. `produttore`
3. `modello`
4. `dataimm` che contiene la data di immatricolazione (primo volo) dell'aereo
5. `numposti`, che contiene il numero di posti disponibili sull'aereo

I campi produttore e modello non possono essere NULL.

Creare la tabella, di nome "aerei" con MySQL. A tale scopo, si consiglia di scrivere il comando necessario dentro un editor di testi, e poi effettuare il "copia e incolla" quando si è terminato.

Successivamente, riempire la tabella in modo che il comando `select * from aerei` dia il seguente risultato:

```
+----------+------------+---------+------------+----------+
| id       | produttore | modello | dataimm    | numposti |
+----------+------------+---------+------------+----------+
| superjet | boeing     | 747     | 2000-10-10 |      350 |
| minijet  | MDD        | Super80 | NULL       |     NULL |
+----------+------------+---------+------------+----------+
```

Se a un certo punto si immettono dati sbagliati, si può cancellare tutto il contenuto della tabella col comando

```
delete from aerei;
```

Provare a inserire anche una nuova riga in modo da avere il risultato seguente (N.B. non si può ottenere a meno di non inserire la stringa 'null' anziché il valore null)

```
+----------+------------+---------+------------+----------+
| id       | produttore | modello | dataimm    | numposti |
+----------+------------+---------+------------+----------+
| superjet | boeing     | 747     | 2000-10-10 |      350 |
| minijet  | MDD        | Super80 | NULL       |     NULL |
| NULL     | boeing     | null    | 2001-2-4   |      400 |
+----------+------------+---------+------------+----------+
```

oppure il seguente

```
+----------+------------+---------+------------+----------+
| id       | produttore | modello | dataimm    | numposti |
+----------+------------+---------+------------+----------+
| superjet | boeing     | 747     | 2000-10-10 |      350 |
| minijet  | MDD        | Super80 | NULL       |     NULL |
| superjet | boeing     | 767     | 2001-2-4   |      400 |
+----------+------------+---------+------------+----------+
```

È possibile?

Esercizio 2

Modificare la struttura della tabella aerei con il comando `ALTER TABLE` in modo che il comando `describe aerei` restituisca il seguente risultato (è stato aggiunto il campo `commento` e tolto il vincolo di chiave primaria al campo `id`).

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| id         | char(20)     | NO   |     | NULL    |       |
| produttore | char(20)     | NO   |     | NULL    |       |
| modello    | char(20)     | NO   |     | NULL    |       |
| dataimm    | date         | YES  |     | NULL    |       |
| numposti   | int(11)      | YES  |     | NULL    |       |
| commento   | text         | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

Adesso aggiungere una nuova riga alla tabella, allo scopo di ottenere i seguenti valori:

```
+----------+------------+---------+------------+----------+----------+
| id       | produttore | modello | dataimm    | numposti | commento |
+----------+------------+---------+------------+----------+----------+
| superjet | boeing     | 747     | 2000-10-10 |      350 |          |
| minijet  | MDD        | Super80 | NULL       |     NULL |          |
| superjet | boeing     | 767     | NULL       |     NULL | prova    |
+----------+------------+---------+------------+----------+----------+
```

Provare a questo punto a riattivare il vincolo di chiave primaria con `ALTER TABLE`. Cosa succede?

Esercizio 3

Fare un po' di esperimenti con i tipi. In particolare:

- cosa succede se tento di inserire in un campo intero un numero più grande di quello che è possibile rappresentare? (ad esempio un valore maggiore di 127 in un campo `tinyint`)
- cosa succede se trasformo (con `ALTER TABLE`) un campo int in un campo int unsigned e la tabella conteneva dei valori negativi?
- provate a memorizzare il numero 0.3 in un campo `FLOAT`.. e poi cambiare il campo in `DOUBLE PRECISION`.. cosa succede?

Esercizio 4

Modificare la tabella aerei in modo che rispetti la seguente struttura

```
+------------+---------------+------+-----+---------+----------------+
| Field      | Type          | Null | Key | Default | Extra          |
+------------+---------------+------+-----+---------+----------------+
| id         | int(11)       | NO   | PRI | NULL    | auto_increment |
| produttore | char(20)      | NO   |     | NULL    |                |
| modello    | char(20)      | NO   |     | NULL    |                |
| dataimm    | date          | YES  |     | NULL    |                |
| numposti   | int(11)       | YES  |     | NULL    |                |
| commento   | text          | YES  |     | NULL    |                |
+------------+---------------+------+-----+---------+----------------+
```

Cosa è successo ai vecchi valori di tipo stringa contenuti nel campo `id`?

Esercizio 5

Abbiamo visto che MySQL dispone del modificatore `UNSIGNED` per i tipi numerici. Esiste un altro modificatore: `ZEROFILL`. Provare a creare una tabella con un campo di tipo intero zerofill, inserire dei dati, e controllare il risultato.

Lezione Precedente Elenco Lezioni Lezione Successiva