Creazione e gestione di database e tabelle

Iniziamo con questa lezione a **costruire un database**. Il lavoro verrà essenzialmente svolto impartendo ordini al DBMS attraverso il linguaggio SQL.

Per eseguire gli esempi presentati qui e nelle lezioni successive, si può utilizzare il client testuale *mysql*. Se ne è già parlato in precedenza, pertanto ricordiamo solo che è sufficiente, da riga di comando, invocarlo fornendo nome utente e password:

```
mysql -u root -p
```

Per i nostri scopi useremo l'account *root*, mentre la password sarà richiesta interattivamente.

I comandi SQL verranno impartiti direttamente nel prompt del client *mysql* dopo l'autenticazione o, in caso di comandi troppo lunghi come la creazione di tabelle, si potrà preparare un file ed inviarlo direttamente al client con l'operatore '<':

```
mysql -u root -p < test.sql;
```

Creare database

Il primo passo da affrontare è la creazione di un database. Dal prompt di *mysql* possiamo innanzitutto vedere quanti database abbiamo già a disposizione, tramite il comando seguente:

```
show databases;
```

Se non ne abbiamo mai creati, probabilmente verranno mostrati solo alcuni database "di servizio", che occorrono allo stesso MySQL per svolgere il suo lavoro.

Ecco come **creare un database con SQL**:

```
CREATE DATABASE nuovodb;
```

dove `nuovodb` è il nome che è stato scelto per il database da creare.

Conviene puntualizzare che anche questo comando deve terminare con un punto e virgola (`;`). Inoltre, sebbene `CREATE DATABASE` è stato scritto interamente in maiuscolo per rispettare le consuetudini e migliorare la leggibilità, MySQL non è case-sensitive per quanto riguarda il nome dei comandi. In altre parole, scrivere `CReaTe DaTAbaSE` `nuovodb` avrebbe avuto lo stesso effetto.

La prova che il database è stato creato si può quindi ottenere tramite `SHOW DATABASES`.

Il nuovo database non diventa però automaticamente quello attivo (cioè quello su cui i comandi SQL saranno indirizzati). Per potervi iniziare a lavorare è dunque necessario selezionare il nostro database, tramite il comando `USE`:

```
USE nuovodb;
```

In alternativa a *mysql*, si può utilizzare il manager universale da riga di comando, *mysqladmin*.

Anche in questo caso dovremo fornire credenziali di accesso da riga di comando e successivamente utilizzare il comando `CREATE` seguito dal nome del nuovo database:

```
mysqladmin -u root -p CREATE nuovodatabase
```

Per verificare l'esito del comando impartito, si può di nuovo usare `SHOW DATABASES`, oppure un altro programma a disposizione del DBMS, *mysqlshow*. Ci si dovrà far riconoscere con le opzioni `-u` e `-p` come di consueto, e tanto sarà sufficiente a vedere l'elenco dei database gestiti al momento.

```
mysqlshow  -u root -p
```

Creare tabelle

Il database fornisce il contesto alla creazione del nostro progetto ma i veri nodi che costituiranno la sua rete di informazioni sono le tabelle. Di seguito impareremo a crearle, tramite un'opazione che può sembrare un'operazione semplice in molti casi, soprattutto con l'aiuto di strumenti visuali, ma che invece è un lavoro delicato e molto importante.

L'esempio proposto può essere svolto all'interno del client *mysql* dopo aver creato un database ed aver dichiarato di volerlo usare (direttiva *USE*).

```
CREATE TABLE `Persone` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NULL,
  `cognome` VARCHAR(45) NULL,
  `dataDiNascita` DATE NULL,
  `sesso` ENUM('M','F') NULL,
  PRIMARY KEY (`id`))
```

```
ENGINE = InnoDB;
```

Il codice precedente crea una tabella denominata `Persone` con cinque campi di vario tipo. Durante la scrittura di un comando così lungo, nel prompt di *mysql* si può andare a capo; sarà poi il punto e virgola (`;`) a segnalare la fine dell'istruzione.

La sintassi mostra che `CREATE TABLE`, seguito dal nome della nuova tabella, è il comando che esegue l'operazione.

Tra le parentesi tonde vengono specificati i campi che la compongono; per ognuno di essi si indica:

- **nome** del campo: nell'esempio i nomi sono, rispettivamente, `id`, `nome`, `cognome`, `dataDiNascita`, `sesso`;
- **tipo di dato** del campo: il formato dell'informazione che vi verrà inserita. I tipi di dato saranno oggetto di approfondimento in una lezione futura della guida; al momento si consideri che quelli mostrati sono alcuni dei tipi più comuni. `INT` serve a specificare un numero intero, `VARCHAR` si utilizza per le stringhe, `DATE` per le date ed `ENUM` definisce un tipo di dato personalizzato che contempla l'assegnazione di un elemento di un limitato insieme di valori (in questo caso, si è deciso di specificare `M` ed `F` per indicare, rispettivamente, maschio e femmina);
- parametri che seguono il tipo di dato e possono specificare **vincoli** attribuiti al campo: `NULL` indica che il campo può essere lasciato vuoto; `NOT NULL`, viceversa, obbliga ad assegnare un valore; `AUTO_INCREMENT` indica che il numero del campo può essere generato in autonomia dal DBMS in ordine progressivo.

Infine si vede che è stato specificato il campo `id` come **chiave primaria**, mediante il comando `PRIMARY KEY`. Una chiave primaria, argomento approfondito nel seguito, indica un valore composto da uno o più campi che individua univocamente il record in cui è collocato.

È possibile anche creare una tabella temporanea utilizzando il comando `CREATE TEMPORARY TABLE` specificando i campi come di consueto. Questo tipo di tabelle esiste per il tempo di una sessione, il che permette a più utenti collegati, appartenenti quindi a sessioni diverse, di poter utilizzare le stesse tabelle. La loro utilità si esplica per lo più nell'immagazzinamento di dati temporanei a supporto di elaborazioni lunghe.

Altro aspetto molto utile da considerare è che se chiediamo di creare una **tabella che già esiste**, viene restituito un errore. Si può quindi ordinare a MySQL di creare la tabella solo nel caso in cui non ne esista già una omonima:

```
CREATE TABLE IF NOT EXISTS Persone
(
    ...
    ...
)
```

Modificare le tabelle

Abbiamo imparato a creare le tabelle del database. Può però capitare di dovere **cambiare la struttura della tabella**: le modifiche possono riguardare ogni aspetto, ma quelli di cui ci occuperemo in questa lezione riguardano per lo più il nome della tabella stessa o i suoi campi (numero, tipo, denominazione).

Il principale costrutto che utilizzeremo sarà **`ALTER TABLE`**. In generale viene seguito dal nome della tabella oggetto di modifica, e da un elemento che specifica il tipo di azione che verrà eseguita. Quelli che ci interesseranno maggiormente in questa lezione sono:

- `RENAME` per rinominare la tabella;
- `ADD` e `DROP`, rispettivamente, per aggiungere e rimuovere un campo;
- `CHANGE` per modificare il nome, il tipo di dato o altri parametri di un campo.

Considerato che tratteremo la modifica di tabelle, dovremo avere un database a disposizione. Immaginiamo quindi di crearne uno con il seguente script:

```
CREATE DATABASE Biblioteca;
USE Biblioteca;
CREATE TABLE `Libri` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Titolo` VARCHAR(45) NULL,
  `Autore` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

Prima di continuare, specifichiamo che il risultato degli esempi che saranno mostrati nel seguito di questa lezione può essere verificato con il comando seguente:

```
SHOW TABLES;
```

Ciò è vero se la modifica riguarda il nome di una tabella. Se invece essa ha effetto su un elemento interno alla tabello, potremo utilizzare:

```
DESCRIBE Libri;
```

Cambiare nome ad una tabella

La prima casistica analizzata è il cambio del nome di una tabella, circostanza che può verificarsi anche per un semplice errore di digitazione durante la progettazione del database.

Immaginiamo di voler modificare il nome della tabella, cambiandolo da `Libri` in `Opere`. A tal fine dovremo far seguire il comando `ALTER TABLE` dall'azione `RENAME`:

```
ALTER TABLE Libri RENAME Opere;
```

La correttezza della modifica apportata potrà essere verificata, in questo caso, con `SHOW TABLES`: se tutto è andato bene, sarà infatti visualizzato il nuovo nome della tabella.

Modifica di nome e tipo di dato dei campi

Utilizzando il comando `CHANGE` in aggiunta ad `ALTER TABLE` è possibile richiedere la modifica di una colonna già esistente.

Ad esempio:

```
ALTER TABLE Libri CHANGE Titolo Titolo varchar(100);
```

La modifica richiesta comporterà la variazione della dimensione del campo. Le stringhe contenute potranno arrivare ora a 100 caratteri anziché 45.

Analizziamo meglio il comando appena impartito:

- `ALTER TABLE Libri`: specifica quale tabella sarà modificata;
- `CHANGE Titolo`: specifica che sarà modificato un elemento esistente (il campo `Titolo`);
- `Titolo varchar(100)`: sono i nuovi attributi assegnati.

Alla stessa maniera sarà possibile modificare il nome di un campo:

```
ALTER TABLE Libri CHANGE Titolo Opera varchar(100);
```

Come si vede, il comando è analogo al precedente. La differenza è rappresentata dalla nuova descrizione che viene assegnata al campo interessato.

Come presumibile, le modifiche ad un campo – qui mostrate in due esempi separati – possono essere apportate con la medesima direttiva.

Aggiunta o rimozione di campi

Il comando `ALTER TABLE` può essere utilizzato per aggiungere o rimuovere i campi di una tabella. Considerando la versione non ancora modificata della tabella `Libri`, il comando:

```
ALTER TABLE Libri ADD NumeroPagine INT;
```

aggiungerà un nuovo campo alla tabella. È stato sufficiente indicare `ADD` come azione, e fare seguire tale direttiva dalla descrizione del nuovo campo, che in questo caso è un intero denominato `NumeroPagine`.

Analogamente, il comando `DROP` in congiunzione con `ALTER TABLE` permetterà di eliminare un campo:

```
ALTER TABLE Libri DROP NumeroPagine;
```

`DROP` dovrà semplicemente essere seguito dal nome del campo da cancellare.

Eliminare tabelle e database

Dopo avere visto come creare e modificare tabelle e database, è arrivato il momento di imparare a cancellarli.

Si tratta di operazioni irreversibili e quindi particolarmente delicate, considerato anche che, quando eseguite da riga di comando, vengono effettuate senza alcuna richiesta di conferma da parte del client testuale.

Come fatto nelle lezioni precedenti, utilizzeremo un database con una sola tabella, che possiamo creare come segue:

```
CREATE DATABASE Biblioteca;
USE Biblioteca;
CREATE TABLE `Libri` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Titolo` VARCHAR(45) NULL,
  `Autore` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

L'**eliminazione di una tabella** può essere effettuata con il seguente comando:

Per verificare il risultato del precedente comando si potrà utilizzare:

Se invece vogliamo **rimuovere l'intero database**, possiamo ancora utilizzare `DROP`, ma nel seguente modo:

```
DROP DATABASE Biblioteca;
```

Anche in questo caso non verrà richiesta alcuna conferma dell'eliminazione.

Sappiamo che, come mostrato nello script di preparazione all'esempio, per poter lavorare con un database è necessario selezionarlo utilizzando il comando `USE`.

L'eliminazione di un database potrebbe riguardare proprio quello in uso, ed anche in questo caso ciò non comporterà alcuna "obiezione" da parte del DBMS. Semplicemente, dopo la cancellazione nessun database risulterà in uso.

Possiamo, infine, **verificare l'avvenuta cancellazione del database**, controllando che esso non compaia nella lista dei database mostrata con il seguente comando:

```
SHOW DATABASES;
```