Creare tabelle in MySQL

Alessandro Bugatti (alessandro.bugatti@istruzione.it)

6 novembre 2016

1 Tipi di dati in MySQL

Come visto nella precedente dispensa l'istruzione CREATE TABLE ci permette di creare lo schema all'interno del quale andremo poi a memorizzare i nostri dati. Nell'esempio visto i dati erano solamente di due tipi, VARCHAR e DATE, ma nella realtà noi potremmo avere bisogno di rappresentare dati di altro tipo (ad esempio numerici). MySQL permette di utilizzare tipi di dati appartenenti a tre diverse famiglie: valori numerici, valori stringa e data/ora.

1.1 I valori numerici

I valori numerici sono tutti quei valori come 456 oppure 129.34, cioè degli interi oppure dei numeri in virgola mobile. Sia gli interi che i numeri in virgola mobile possono essere di diverso tipo, come riassunto nella tabella 1.

Tabella 1: Dati di tipo numerico

Tipo	**Significato**
TINYINT	Intero molto piccolo (1 byte)
SMALLINT	Intero piccolo (2 byte)
MEDIUMINT	Intero di medie dimensioni (3 byte)
INT	Intero (4 byte)
BIGINT	Intero di grandi dimensioni (8 byte)
FLOAT	Numero in virgola mobile in precisione singola
DOUBLE	Numero in virgola mobile in precisione doppia
DECIMAL	Numero in virgola mobile rappresentato come una stringa

Questa tabella mostra solo i tipi di dati, per una descrizione più dettagliata è utile consultare il manuale di MySQL della versione che si sta utilizzando. Le uniche cose che possiamo notare sono:

- nei tipi interi l'unica differenza è la dimensione in byte e quindi il numero massimo rappresentabile: una buona regola è quella di scegliere sempre il tipo più piccolo che permetta di rappresentare completamente il dominio dell'attributo. Inoltre ognuno di questi tipi può essere dichiarato come UNSIGNED per evitare che vi possano essere assegnati valori negativi e anche di tipo AUTO_INCREMENT per far si che ogni valore contenuto venga generato automaticamente dal DBMS incrementando di uno il valore precedente (un solo attributo nella tabella può essere di tipo AUTO_INCREMENT, solitamente lo è la chiave primaria).

- i tipi in virgola mobile servono a rappresentare numeri con virgola e mentre l'unica differenza fra FLOAT e DOUBLE è il range e la precisione che si ottiene (un DOUBLE ha una capacità rappresentativa doppia di un FLOAT)
- i tipi DECIMAL utilizzano una rappresentazione di tipo stringa che permette di avere una precisione arbitraria (è quindi possibile rappresentare numeri con un qualsiasi numero di cifre) a scapito dell'efficienza nei calcoli.

1.2 I valori stringa

I valori stringa servono a rappresentare qualsiasi tipo di dato che non sia un numero o una data. Non sono altro che una serie di byte che possono servire a rappresentare del testo (che è il caso più comune), ma anche file binari, multimediali, immagini, ecc.

Nella tabella 2 sono indicati tutto i tipi di dato stringa possibili.

Tabella 2: Dati di tipo stringa

Tipo	**Significato**
CHAR	Stringa di caratteri a lunghezza fissa
VARCHAR	Stringa di caratteri a lunghezza variabile
TINYBLOB	BLOB (*binary large object* oggetto binario di grandi dimensioni) di dimensioni molto ridotte
BLOB	BLOB di dimensioni ridotte
MEDIUMBLOB	BLOB di medie dimensioni
LONGBLOB	BLOB di grandi dimensioni
TINYTEXT	Stringa di testo di dimensioni molto ridotte
TEXT	Stringa di testo di dimensioni ridotte
MEDIUMTEXT	Stringa di testo di medie dimensioni
LONGTEXT	Stringa di testo di grandi dimensioni
ENUM	Enumerativo: a ogni valore della colonna può corrispondere un solo elemento di un insieme enumerativo
SET	Un insieme: a ogni valore della colonna possono essere assegnati più elementi di un insieme

Anche in questo caso per informazioni più precise è necessario consultare il manuale MySQL, noi ci limitiamo alle seguenti osservazioni:

- i tipi CHAR e VARCHAR sono quelli più utilizzati per rappresentare delle stringhe e ambedue hanno un valore massimo di byte di 255, che viene rappresentato racchiudendolo tra parentesi tonde. La differenza è che mentre i tipi CHAR hanno una lunghezza fissa indipendentemente dalla lunghezza effettiva dei dati contenuti (se ad esempio abbiamo un CHAR(20) tutti i dati di quella colonna occuperanno 20 byte), i VARCHAR occupano solo lo spazio strettamente necessario a contenere i dati[1].

[1] Alcune volte il motore di MySQL può convertire delle nostre dichiarazioni di tipo da CHAR a VARCHAR in base a una logica di maggior efficienza.

- i tipi BLOB e TEXT sono simili fra i loro a parte il fatto che i dati nei BLOB sono di tipo binario, mentre nei campi di tipo TEXT sono dei caratteri appartenenti ad un certo insieme di caratteri. Questo influenza le operazioni di ordinamento e comparazione che portano a risultati differenti. All'interno poi di ognuno di questi tipi ci sono dei sottotipi (TINY, MEDIUM, LONG) la cui differenza risiede solamente nella quantità di byte memorizzabili. Quindi i campi di tipo TEXT sono da utilizzare per memorizzare testi abbastanza lunghi, superiori ai 255 caratteri, mentre in campi BLOB per file binari, come foto, documenti, file audio[2], ecc.

- i tipi ENUM e SET permettono di inserire dei valori che sono presenti all'interno di un certo insieme e la differenza è che mentre ENUM permette di inserire un solo valore, SET permette di inserire un insieme di valori, eventualmente anche tutti.

1.3 I valori data e ora

Gli ultimi tipi di dati che prendiamo in considerazione sono quelli di tipo data e ora, che sono rappresentati nella tabella 3.

Tabella 3: Dati di tipo data e ora

Tipo	**Significato**
DATE	Valore data in formato 'SSAA-MM-GG'
TIME	Valore tempo in formato 'oo:mm:ss'
DATETIME	Valore data e tempo in formato 'SSAA-MM-GG oo:mm:ss'
TIMESTAMP	Valore timestamp in formato SSAAMMGGoommss
YEAR	Valore anno in formato SSAA

L'unica osservazione per questo tipo di dati il cui significato è piuttosto evidente è per il tipo *timestamp*: esso serve per tenere traccia di quando siano state effettuate le ultime modifiche all'interno di un record.

2 Esempi di creazione di tabelle

Proviamo adesso a utilizzare i tipi di dati visti in precedenza per creare delle tabelle[3], aggiungendo anche la possibilità di definire la chiave primaria.

Supponiamo di voler rappresentare un'entità articolo contenente:

- un ID che si autoincrementa e che farà da chiave primaria
- un nome che può essere lungo al massimo 100 caratteri
- una descrizione che può avere una lunghezza di qualche centinaia di caratteri

[2]Nella pratica i campi BLOB non vengono utilizzati molto spesso, perchè si preferisce memorizzare gli oggetti binari nel file system e inserire in un campo testuale all'interno del database le informazioni per recuperarli, quindi il nome del file e il suo percorso.

[3]Può essere pratico quando si utilizza il client a riga di comando di MySQL per eseguire delle istruzioni di scriverle all'interno di Scite o di qualsiasi editor di testo e poi copiarle all'interno della finestra dove è in esecuzione il client: questo permetterà generalmente di dover digitare meno codice a fronte di eventuali errori o di ripetizioni di istruzioni simili.

- la data in cui è entrato in produzione

In questo caso l'istruzione DDL[4] da utilizzare sarà:

```
CREATE TABLE articolo
(
    ID INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    nome CHAR(100) NOT NULL,
    descrizione TEXT NOT NULL ,
    data_di_produzione DATE NOT NULL
);
```

Come possiamo essere sicuri di avere effettivamente creato la tabella e che questa contenga i campi desiderati? Per visualizzare le tabelle contenute all'interno di un database è sufficiente digitare l'istruzione:

```
SHOW TABLES;
```

mentre per visualizzare la struttura della tabella utilizzeremo l'istruzione:

```
DESCRIBE articolo;
```

Passiamo a un altro esempio: vogliamo rappresentare l'entità libro con i seguenti attributi:

- il codice ISBN che sarà la chiave primaria ed è composto da 16 caratteri
- il titolo
- l'autore
- il prezzo in euro
- la data di pubblicazione
- un timestamp che ci permettà di risalire a quando il record relativo ad un determinato libro è stato inserito nel database

In questo caso la CREATE TABLE potrebbe essere di questo tipo:

```
CREATE TABLE libro
(
    ISBN CHAR(16) NOT NULL PRIMARY KEY,
    titolo VARCHAR(150) NOT NULL,
    autore VARCHAR(50) NOT NULL,
    prezzo FLOAT NOT NULL ,
    data_di_pubblicazione DATE NOT NULL,
    data_di_inserimento TIMESTAMP NOT NULL
);
```

[4] Data Definition Language, è l'insieme delle istruzioni SQl che servono a creare la struttura del database.

Si può notare come sia stato scelto di rappresentare il codice ISBN come un **CHAR** di 16 caratteri, perchè la dimensione è sempre la stessa e quindi sembrava più adatto. In realtà, utilizzando l'istruzione **DESCRIBE**, si potrà notare come esso sia stato convertito in un **VARCHAR** a nostra insaputa: questo è dovuto alle regole di ottimizzazione interne di MySQL e non può essere modificato. Per dare un valore corretto all'attributo data_di_inserimento sarà opportuno nell'istruzione **INSERT** utilizzare la funzione **NOW()**, che ritorna il *timestamp* corrente.

Vediamo un ultimo esempio nel quale vogliamo creare l'entità atleta composta da:

- ID che è autoincrementante e sarà la chiave primaria
- nome
- cognome
- nazionalità
- altezza
- peso
- sesso (MASCHIO o FEMMINA)

L'istruzione che dovremo lanciare sarà:

```
CREATE TABLE atleta
(
    ID INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    nome CHAR(50) NOT NULL,
    cognome CHAR(50) NOT NULL,
    altezza FLOAT NOT NULL ,
    peso FLOAT NOT NULL,
    sesso ENUM( 'Maschio' , 'Femmina' )
);
```

In quest'ultimo esempio si può notare come il campo *sesso* sia di tipo **ENUM** e l'insieme dei valori possibili sia 'Maschio' o 'Femmina'. Questo comporterà che nella fase di inserimento verranno accettati solo questi due valori e rifiutati tutti gli altri.

3 Vincoli di integrità referenziale

Il modello relazionale è fondato sul meccanismo di referenzialità indotto dalla presenza delle chiavi primarie (PRIMARY KEY) e delle chiave esterne (FOREIGN KEY) all'interno di tabelle che sono tra di loro legate da una relazione nel modello ER.

Avendo ad esempio un'entità studente e un'entità voto che sono nella relazione 1:N "uno studente può prendere più voti, un voto è relativo a un solo

studente", studente e voto saranno due tabelle e la tabella voto conterrà come attributo una chiave esterna, chiamata ad esempio *id_studente*, che farà riferimento alla chiave primaria di uno studente.

Una volta compreso il meccanismo con cui si possono creare delle relazioni tra tabelle, viene naturale domandarsi se è possibile evitare il verificarsi di situazioni che ledano l'integrità dei dati, ad esempio la presenza di un voto che contenga un valore di *id_studente* che non ha un corrispondente negli *id* della tabella studente.

A questo proposito la maggior parte dei database relazionali offre la possibiltà di creare vincoli di integrità referenziale[5], proprio con lo scopo di salvaguardare l'integrità dei dati in un database.

Facendo riferimento all'esempio degli studenti e dei voti il vincolo di integrità referenziale può essere creato nel modo seguente:

```
CREATE TABLE voto
(
    ID INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    punteggio float NOT NULL,
    data DATE NOT NULL,
    tipologia ENUM( 'Scritto' , 'Orale', 'Altro' ),
    id_studente INT UNSIGNED,
    FOREIGN KEY (id_studente) REFERENCES studente(ID)
);
```

Da notare che la chiave esterna deve essere dello stesso tipo della chiave primaria (in questo esempio intero senza segno), altrimenti verrà mostrato un messaggio di errore che impedirà di creare la tabella.

Una volta imposto questo vincolo il database impedirà l'inserimento di voti che non facciano riferimento a degli studenti presenti nel database, garantendo così l'integrità referenziale anche a fronte di errori nell'inserimento o a errori di programmazione dell'applicativo che utilizza il database.

Un ultimo passaggio, che è la conseguenza naturale dell'imposizione di questi vincoli, è demandare al database anche la gestione delle problematiche che si potrebbero avere durante l'aggiornamento o la cancellazione di uno studente. In pratica si vuole che il database, in maniera automatica, gestisca le seguenti condizioni:

- cosa succede cambiando (ON UPDATE) l'id di uno studente che ha dei voti nel database?
- cosa succede eliminando (ON DELETE) uno studente che ha dei voti nel database?

Entrambe le situazioni, se non gestite, darebbero dei problemi, quindi esistono delle istruzioni per indicare al database come reagire nel caso di UPDATE o

[5]Si tenga presente che in linea di principio un database relazionale non necessita della presenza di meccanismi per la creazione di vincoli di integrità referenziale: ad esempio MySQL fino alla versione 3 lasciava al programmatore la responsibilità di gestire le problematiche legate all'integrità referenziale.

DELETE di dati legati a dati presenti in altre tabelle attraverso un vincolo referenziale.

Le possibili scelte sono tre e sono le seguenti

- RESTRICT (equivalente a NO ACTION): l'azione viene impedita, cioè non si possono modificare/cancellare dati se hanno altri dati che fanno riferimento a loro in altre tabelle
- CASCADE: l'azione viene eseguita in cascata sui dati collegati, ad esempio eliminando uno studente verrebbero eliminati tutti i suoi voti
- SET NULL: la chiave esterna dei dati che fanno riferimento al dato modificato/eliminato viene impostata a NULL

La scelta tra queste tre azioni spetta al progettista, che in base al contesto sceglierà l'azione più adatta.

Tornando all'esempio precedente, potrebbe essere modificato in questo modo per aggiungere le azioni da svolgere in relazione alle situazioni di modifica e cancellazione

```
CREATE TABLE voto
(
    ID INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    punteggio float NOT NULL,
    data DATE NOT NULL,
    tipologia ENUM( 'Scritto' , 'Orale', 'Altro' ),
    id_studente INT UNSIGNED,
    FOREIGN KEY (id_studente) REFERENCES studente(ID)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);
```

In questo modo si indica che eventuali cambiamenti sulla chiave primaria[6] di studente modificheranno automaticamente la chiave esterna *id_studente* relativa allo studente modificato, mentre invece l'eliminazione di uno studente verrà impedita nel caso egli abbia dei voti (volendolo eliminare sarà quindi necessario prima eliminare i suoi voti e solo successivamente lo studente). Da notare la mancanza di virgole dopo ogni riga delle istruzioni legate alle azioni, perchè comunque fanno ancora parte dell'istruzione FOREIGN KEY.

[6]Cambiamenti della chiave primaria nella realtà non capitano di frequente, soprattutto quando questa è una chiave artificiale.