
FUTURA
LA SCUOLA
PER L'ITALIA DI DOMANI
Finanziato
dall'Unione europea
NextGenerationEU
Ministero dell'Istruzione
e del Merito
Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA
COESIONE
ITALIA 21-27
PIEMONTE
Cofinanziato
dall'Unione europea
REGIONE
PIEMONTE

Creazione, Lettura, Aggiornamento e Eliminazione dei Record (CRUD)

Una volta **creata** la **struttura** del nostro database, ci ritroveremo, ovviamente, con una serie di **tabelle vuote**. Prima di **aggiungere record** a una tabella, è necessario conoscere il **tipo di dati** previsto per ogni campo, quali campi non possono avere valore nullo e quali campi hanno l'incremento automatico.

Quando si inseriscono i dati, bisogna utilizzare le virgolette o gli **apici** per i dati di tipo **stringa** (compresa la data), mentre **non si utilizzano** virgolette o apici per i dati di tipo **numerico**. Inoltre, non si inseriscono i valori per i campi definiti con l'attributo auto_increment.

Create: INSERT INTO

`INSERT INTO` è l'istruzione utilizzata per inserire nuovi record in una tabella. Ha due parti:

1. `INSERT INTO` seleziona la tabella e i campi per i quali effettuare l’inserimento.
2. `VALUE` / `VALUES` elenca i valori dei campi da inserire.
   È possibile inserire più record con un solo `INSERT`, separando l’elenco dei valori di ogni record con la virgola (`,`).

Altra sintassi per un singolo record con l'istruzione `SET`:

```
INSERT INTO tableName (field1, field3)
VALUES (value1, value3);
```

```
INSERT INTO tableName (field1, field2, field3,...)
VALUES (r1_value1, r1_value2, r1_value3, ...), (r2_value1, r2_value2, r2_value3, ...);
```

```
INSERT INTO tableName
SET field1 = 'value1', field2 = 'value2', field3 = 'value2';
```

INSERT INTO

È possibile utilizzare il comando `INSERT INTO` senza l'uso di nomi di campo se si inserisce un record rispettando l'ordine dei campi della tabella. In questo caso, devono essere inseriti i valori di tutti i campi, **anche i valori AUTO_INCREMENT o TIMESTAMP** (passando "default" per inserimento automatico). Per i campi che accettano i valori nulli, è possibile passare "null".

```
INSERT INTO tableName
VALUES (value1, value2, value3);
```

Esempio:

```
INSERT INTO studente
VALUES (default, 'fabio', 'rossi', 'fbr@gmail.com', null, default);
```

Read: Mostrare i Record di una Tabella

È possibile visualizzare i record di una tabella utilizzando l'istruzione `SELECT`. Per visualizzare tutti i record da una tabella, si usa il carattere jolly `*`. Dobbiamo anche utilizzare l'istruzione `FROM` per identificare la tabella che vogliamo interrogare. Di solito si visualizzano campi specifici, piuttosto che l'intera tabella. Dopo l'istruzione `SELECT`, elencare i campi che interessano, separati da una virgola.

```sql
SELECT * FROM tableName;
```

```sql
SELECT fieldName, fieldName2, fieldName3 FROM tableName;
```

Combinare INSERT INTO e SELECT

È possibile inserire i dati prendendoli da un’altra tabella utilizzando l'istruzione `INSERT INTO ... SELECT`:

```
INSERT INTO amici (nome, cognome)
SELECT nome, cognome
FROM studenti;
```

Nell'esempio qui sopra, abbiamo immaginato di popolare la tabella `amici` inserendo automaticamente i dati già presenti nella tabella `studenti`.

È importante notare che i campi nelle due tabelle devono contenere lo stesso tipo di dato e che la tabella `amici` deve esistere.

CREATE TABLE ... SELECT

Si possono creare delle tabelle già popolate di dati mediante l'uso congiunto delle istruzioni `CREATE TABLE` e `SELECT`.

`CREATE TABLE` crea la nuova tabella, mentre `SELECT` carica i dati prelevandoli da un'altra tabella. Il suo funzionamento, in pratica, è analogo a quello di `INSERT INTO ... SELECT`.

```
CREATE TABLE parenti (
    id INT AUTO_INCREMENT,
    nome VARCHAR(20),
    cognome VARCHAR(30),
    PRIMARY KEY(id)
) SELECT nome, cognome FROM amici;
```

Nell'esempio qui sopra, abbiamo creato la tabella `parenti` con la stessa struttura della tabella `amici`, popolandola con i dati prelevati dalla tabella `amici`.

Duplicare Tabelle e Contenuti

Se abbiamo necessità di copiare il contenuto di una tabella in un'altra tabella, possiamo utilizzare l'istruzione `CREATE TABLE` combinata con `LIKE` e le istruzioni `SELECT`.

Per duplicare esattamente una tabella (con indici e chiavi) e i suoi contenuti, bisogna usare due istruzioni separate:

```
CREATE TABLE studenti_bk LIKE studenti;
INSERT INTO studenti_bk SELECT * FROM studenti;
```

Si può anche utilizzare un'unica istruzione. In questo caso, gli indici non vengono ricreati, cioè le strutture delle tabelle sono diverse:

```
CREATE TABLE studenti_bk2 AS SELECT * FROM studenti;
```

Nell'esempio sopra, abbiamo duplicato la tabella `studenti` nella tabella `studenti_bk` mantenendo la struttura e i dati.

Update: istruzioni per modificare i dati

L'istruzione `UPDATE` viene utilizzata per aggiornare i record in una tabella. Questa istruzione modifica il valore presente in una colonna di un record già esistente. Viene utilizzata insieme all’istruzione `SET`.

```
UPDATE tableName
SET field1 = value1, field2 = value2
WHERE field3 = value3;
```

Dopo `UPDATE`, indichiamo quale tabella è interessata. Con `SET`, specifichiamo quali colonne modificare e quali valori assegnare. Con `WHERE` (opzionale), stabiliamo le condizioni che determinano quali righe saranno interessate dalle modifiche. Attenzione: se non specifichiamo una condizione, tutte le righe saranno modificate.

- Per operare simultaneamente su più campi, è sufficiente suddividere le coppie chiave/valore con una virgola.
- Quando si inseriscono i dati in una tabella, è importante considerare sempre come sono stati definiti gli attributi per evitare errori di inserimento.
- Se si inserisce un valore troppo lungo o non compreso dalla definizione dell’attributo, MySQL restituirà un errore e non effettuerà alcuna modifica.

Ad esempio, consideriamo il seguente comando di aggiornamento:

```
UPDATE studenti SET genere = 's' WHERE id = 1;
```

Se il campo `genere` della tabella `studenti` è definito come `ENUM('m','f')`, accetta solo i valori 'm' o 'f'. Nel caso sopra riportato, stiamo tentando di inserire un valore non ammesso ('s'), il che provocherà un errore come questo:

```
ERROR 1265 (01000): Data truncated for column 'genere' at row 1
```

Questo errore indica che il valore inserito è stato troncato perché non è conforme alla definizione dell'attributo.

È importante notare che il comportamento dipende dall'impostazione della variabile globale `@@sql_mode`: di default, MySQL lavora in strict mode.

Delete: Eliminazione dei Record in una Tabella

L'istruzione `DELETE` viene utilizzata per eliminare gruppi di record in una tabella. È necessario utilizzare la parola chiave condizionale `WHERE` per isolare quali record si desidera eliminare, altrimenti si eliminano tutti i record.

La sintassi di base per l'istruzione è la seguente:

```
DELETE FROM tableName
WHERE field = value;
```

Esempio:

```
DELETE FROM studenti
WHERE genere = 'm';
```

Nell'esempio sopra, stiamo eliminando i record dalla tabella `studenti` dove il genere è "m".

Eliminare Tutti i Record della Tabella

Per **svuotare una tabella**, si usa l'istruzione `TRUNCATE`. Questa soluzione è la più veloce perché elimina la struttura della tabella per poi ricrearne una uguale vuota, azzerando il valore di eventuali campi AUTO_INCREMENT.

Usando `DELETE`, si eliminano tutti i record presenti nella tabella specificata uno per uno. Questo metodo, seppur funzionante, è poco efficiente poiché **dipende dalla quantità** di righe presenti in tabella. Inoltre, usando `DELETE`, il valore di un eventuale **AUTO_INCREMENT rimane inalterato**; si può comunque modificare il valore di incremento con l'istruzione `ALTER`:

```
TRUNCATE [TABLE] tableName;

DELETE FROM tableName;

ALTER TABLE tableName AUTO_INCREMENT = 1;
```

La differenza principale tra `TRUNCATE` e `DELETE` è che `TRUNCATE` è un'operazione non registrata, mentre `DELETE` viene registrato in log, quindi `TRUNCATE` è generalmente più veloce su grandi tabelle. Tuttavia, `TRUNCATE` non può essere eseguito su tabelle referenziate da vincoli esterni di chiave, mentre `DELETE` può.