Introduzione a MySQL

Francesco Gobbi

I.I.S.S. Galileo Galilei, Ostiglia

5 febbraio 2025

Cos'è MySQL

- MySQL è un sistema di gestione di database relazionali (RDBMS, Relational Database Management System).
- È open source e basato su linguaggio SQL (Structured Query Language).
- Viene utilizzato per applicazioni web, gestione di dati e altro.

Perché usare MySQL?

- È **open-source** e gratuito.
- Ha ottime prestazioni per applicazioni di piccole e grandi dimensioni.
- Supporta diversi tipi di motori di archiviazione (es. *InnoDB* e *MyISAM*).
- È compatibile con molte piattaforme (Windows, macOS, Linux).

Tipi di dati in MySQL

I tipi di dati in MySQL si suddividono in tre categorie principali:

- **Dati numerici**
- **Dati stringa (testo)**
- **Dati temporali (date e orari)**

Tipi di variabili/campi in MySQL

I principali tipi di variabili/campi in MySQL sono suddivisi in:

- **Dati numerici:**
  - **INT**: Numeri interi.
  - **DECIMAL (o NUMERIC)**: Numeri con precisione decimale.
  - **FLOAT e DOUBLE**: Numeri a virgola mobile.
- **Dati stringa:**
  - **CHAR(n)**: Stringa a lunghezza fissa (n caratteri).
  - **VARCHAR(n)**: Stringa a lunghezza variabile (fino a n caratteri).
  - **TEXT**: Stringa di grandi dimensioni.
  - **ENUM**: Lista di valori predefiniti.
- **Dati temporali:**
  - **DATE**: Rappresenta una data (`YYYY-MM-DD`).
  - **DATETIME**: Rappresenta data e ora (`YYYY-MM-DD HH:MM:SS`).
  - **TIME**: Rappresenta un'ora (`HH:MM:SS`).
  - **TIMESTAMP**: Data e ora con zona temporale.

Vincoli in MySQL

Cosa sono i vincoli?

- I **vincoli** (o *constraints*) sono regole applicate ai campi di una tabella per garantire l'integrità e la coerenza dei dati.
- Servono a:
  - Evitare errori e duplicati nei dati.
  - Mantenere relazioni corrette tra le tabelle.
  - Applicare regole aziendali sui dati.

Principali vincoli in MySQL:

- **PRIMARY KEY (PK):** Identifica univocamente ogni riga di una tabella.
- **AUTO_INCREMENT:** Incrementa automaticamente il valore di un campo numerico.
- **FOREIGN KEY (FK):** Collega una tabella a un'altra garantendo l'integrità referenziale.
- **UNIQUE:** Garantisce che i valori in una colonna siano univoci.
- **NOT NULL:** Impedisce che un campo possa avere valori nulli.
- **CHECK:** Applica una condizione sui valori accettati in una colonna.

PRIMARY KEY (PK)

Definizione: Identifica univocamente ogni riga di una tabella. Una tabella può avere una sola `PRIMARY KEY`, composta da una o più colonne. **Esempio:**

```
1 CREATE TABLE studenti (
2     id INT PRIMARY KEY,
3     nome VARCHAR(50)
4 );
```

AUTO_INCREMENT

Definizione: Utilizzato con campi numerici per incrementare automaticamente il valore. È spesso usato insieme a PRIMARY KEY. **Esempio:**

```
CREATE TABLE studenti (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(50)
);
```

FOREIGN KEY (FK)

Definizione: Collega una tabella a un'altra, garantendo l'integrità referenziale. Il valore della `FOREIGN KEY` deve corrispondere a un valore presente nella tabella di riferimento. **Esempio:**

```
CREATE TABLE iscrizioni (
    studente_id INT,
    FOREIGN KEY (studente_id) REFERENCES studenti(id)
);
```

UNIQUE

Definizione: Garantisce che tutti i valori in una colonna siano univoci. Una tabella può avere più vincoli `UNIQUE`. **Esempio:**

```
1 CREATE TABLE docenti (
2     email VARCHAR(100) UNIQUE
3 );
```

NOT NULL

Definizione: Impedisce che una colonna possa contenere valori nulli. È utile per campi obbligatori. **Esempio:**

```
1 CREATE TABLE classi (
2     nome VARCHAR(50) NOT NULL
3 );
```

CHECK

Definizione: Impone una condizione sui valori accettati in una colonna. Questa condizione deve essere soddisfatta per ogni valore inserito. **Esempio:**

```
1 CREATE TABLE esami (
2     voto INT CHECK (voto BETWEEN 18 AND 30)
3 );
```

Struttura generale in MySQL

Regola generale per la sintassi in MySQL:

- `CREATE TABLE nome_tabella (colonna_1 TIPO, colonna_2 TIPO, ...);`
- `INSERT INTO nome_tabella (colonna_1, colonna_2, ...) VALUES (valore_1, valore_2, ...);`
- ```
  SELECT colonna_1, colonna_2
  FROM nome_tabella
  WHERE condizione;
  ```
  ```

Esempio concreto

Scenario: Gestione degli studenti di una scuola.

Creazione della tabella:

```sql
CREATE TABLE studenti (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    classe INT NOT NULL
);
```

Inserimento di dati:

```sql
INSERT INTO studenti (nome, cognome, classe)
VALUES ('Mario', 'Rossi', 5);
```

Query per selezionare gli studenti di quinta classe:

```sql
SELECT nome, cognome
FROM studenti
WHERE classe = 5;
```

Scenario: Sistema di gestione scolastica

Scenario:

- Gestire un database scolastico con tabelle per **studenti**, **classi** e **docenti**.
- Gli studenti sono assegnati a una classe, e ogni classe ha un docente responsabile.

Creazione della tabella `classi`:

```
CREATE TABLE classi (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(20) NOT NULL,
    docente_id INT,
    FOREIGN KEY (docente_id) REFERENCES docenti(id)
);
```

Creazione della tabella `docenti`:

```
CREATE TABLE docenti (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    materia VARCHAR(50)
);
```

Creazione della tabella studenti

Creazione della tabella `studenti`**:**

```sql
CREATE TABLE studenti (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    classe_id INT NOT NULL,
    FOREIGN KEY (classe_id) REFERENCES classi(id)
);
```

Inserimento di dati nelle tabelle:

```sql
INSERT INTO docenti (nome, cognome, materia)
VALUES ('Giovanni', 'Verdi', 'Matematica');

INSERT INTO classi (nome, docente_id)
VALUES ('5A', 1);

INSERT INTO studenti (nome, cognome, classe_id)
VALUES ('Mario', 'Rossi', 1),
       ('Luca', 'Bianchi', 1);
```

Query per la selezione dei dati (JOIN con WHERE)

Query 1: Selezionare tutti gli studenti con la loro classe:

```sql
SELECT studenti.nome AS NomeStudente,
       studenti.cognome AS CognomeStudente,
       classi.nome AS Classe
FROM studenti AS s, classi AS c
WHERE s.classe_id = c.id;
```

Spiegazione delle parole chiave:

- **SELECT:** Indica le colonne da restituire nella query.
- **AS:** Rinomina una colonna nel risultato della query (es. `studenti.nome` diventa `NomeStudente`).
- **FROM:** Specifica le tabelle da cui estrarre i dati.
- **WHERE:** Definisce la condizione per collegare le tabelle o filtrare i dati (`studenti.classe`$_id = classi.id$).

Query per la selezione dei dati (JOIN con WHERE)

Query 2: Selezionare le classi con i rispettivi docenti:

```sql
SELECT classi.nome AS Classe,
       docenti.nome AS NomeDocente,
       docenti.cognome AS CognomeDocente
FROM classi AS c, docenti AS d
WHERE c.docente_id = d.id;
```

Nota: In entrambe le query, `WHERE` è usato per specificare la relazione tra le tabelle.

Query avanzate e aggregazioni (JOIN con WHERE)

Query 1: Contare il numero di studenti per classe:

```sql
SELECT classi.nome AS Classe,
       COUNT(studenti.id) AS NumeroStudenti
FROM studenti, classi
WHERE studenti.classe_id = classi.id
GROUP BY classi.nome;
```

Spiegazione delle parole chiave:

- **COUNT:** Funzione di aggregazione che conta il numero di righe.
- **GROUP BY:** Raggruppa i risultati per una colonna specificata (es. `classi.nome`).

Query avanzate e aggregazioni (JOIN con WHERE)

Query 2: Cercare studenti di una specifica classe:

```sql
SELECT studenti.nome, studenti.cognome
FROM studenti, classi
WHERE studenti.classe_id = classi.id
  AND classi.nome = '5A';
```

Spiegazione delle parole chiave aggiuntive:

- **AND:** Specifica più condizioni da soddisfare contemporaneamente.
- **'5A':** Una stringa tra apici singoli rappresenta un valore testuale.

Modifica e cancellazione dei dati

Aggiornare il docente di una classe:

```sql
UPDATE classi
SET docente_id = (
    SELECT id
    FROM docenti
    WHERE nome = 'Maria' AND cognome = 'Rossi'
)
WHERE nome = '5A';
```

Cancellare uno studente:

```sql
DELETE FROM studenti
WHERE nome = 'Mario' AND cognome = 'Rossi';
```

Cancellare una classe (con verifica referenziale):

```sql
DELETE FROM classi
WHERE id = 1;
```

(N.B. Serve assicurarsi che non ci siano studenti assegnati alla classe.)