Appunti pratici MySql

QL, DDL e DML

Sommario

Creazione di un database:

Crea il database vuoto.

```
CREATE DATABASE <nome>;
```

Crea il database solo se non esiste

```
CREATE DATABASE IF NOT EXISTS <nome>;
```

Elimina il database se esiste, poi lo crea nuovo.

```
DROP DATABASE IF EXISTS <nome>
CREATE DATABASE <nome>;
```

Creazione di una tabella:

```
[DROP TABLE IF EXISTS <nome>]
CREATE TABLE <nome>(
   <campo stringa> VARCHAR(25),
   <campo numerico> INT(11),
   <campo data> DATE,
   <campo con ora> DATETIME
);
```

→L'ultimo elemento non necessita della virgola finale←

Aggiunta di chiavi o campi particolari:

```
[DROP TABLE IF EXISTS <nome>]
CREATE TABLE <nome>(
  id INT(11) AUTO_INCREMENT,
  nome VARCHAR(25) NOT NULL ,
  cognome VARCHAR(25) DEFAULT 'Rossi',
  età INT(11) DEFAULT NULL,
  data_nascita DATE UNIQUE,
  id_sec INT(11),
  PRIMARY KEY(id),
  FOREIGN KEY (id_sec ) REFERENCES <nomeTabella2>(<campoTabella2>)
);
```

→AUTO INCREMENT: ogni volta che aggiungo un dato, questo campo aumenta di 1. →NOT NULL: non permette valori nulli. →DEFAULT: se non viene inserito alcun valore viene assunto quello specificato da questo campo. →UNIQUE: tutti i campi in questa colonna devono essere diversi uno dall'altro. →PRIMARY KEY: permette di definire una o più chiavi primarie →FOREIGN KEY: permette di assegnare ad un campo della tabella corrente un riferimento ad un campo di una tabella primaria.

QUESTO AVVIENE SOLO SE LA TABELLA PRIMARIA È STATA GIÀ CREATA←

Inserimento di dati:

Per inserire i dati principalmente si usa INSERT TO, accompagnato da VALUES

```
USE <nomeDatabase>;
INSERT INTO <nomeTabella>(<colonna1>,<colonna2>)
VALUES(<valoreColonna1>,<valoreColonna2>);
```

Posso anche impostare un valore a tutte le istanze, tramite SET. Posso anche applicare dei filtri, usando WHERE.

```
USE <nomeDatabase>;
INSERT INTO <nomeTabella>
SET <colonna> = <valore>;
```

ATTENZIONE: In questo modo inserirò un nuovo dato e non modificherò un dato esistente, per farlo dovrò usare UPDATE.

È inoltre possibile effettuare un inserimento di dati multiplo, nel seguente modo:

```
USE <nomeDatabase>;
INSERT INTO <nomeTabella>(<colonna1>,<colonna2>)  VALUES
(<valore1Colonna1>,<valore1Colonna2>),
(<valore2Colonna1>,<valore2Colonna2>),
(<valore3Colonna1>,<valore3Colonna2>),
(<valore4Colonna1>,<valore4Colonna2>);
```

Modifica di una tabella:

Eliminare una tabella:

```
DROP TABLE <nomeTabella>
```

Rinominare una tabella:

```
ALTER TABLE <nomeTabella>
RENAME TO <nuovoNomeTabella>;
```

Aggiungere una colonna a una tabella:

```
ALTER TABLE <nomeTabella>
ADD COLUMN <nuovaColonna> <tipoNuovaColnna>;
```

Eliminare una colonna:

```
ALTER TABLE <nomeTabella>
DROP COLUMN <nomeColonna>;
```

Spostare una colonna:

```
ALTER TABLE <nomeTabella>

MODIFY COLUMN <nomeColonna> <tipoColonna>  AFTER <altroNomeColonna>;
```

Modificare il nome di un campo della tabella:

CHANGE serve per cambiare il nome ad un campo, e potenzialmente anche il tipo.

```
ALTER TABLE <nomeTabella>
CHANGE <nomeCampo> <nuovoNomeCampo> <tipoCampo>;
```

Modificare il tipo di un campo della tabella:

MODIFY serve quindi per cambiare il tipo di un campo.

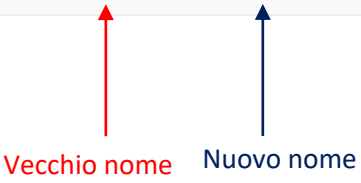
```
ALTER TABLE <nomeTabella>
MODIFY <nomeCampo> <nuovoTipo>;
```

Esempio con MODIFY:

```
ALTER TABLE amici MODIFY cellulare VARCHAR(25);
```

Esempio con CHANGE:

```
ALTER TABLE amici CHANGE cellulare telefono_cellulare INT(11);
```



L'integrità referenziale

→Sono le azioni che posso impostare quando viene modificato un elemento con figli nella tabella, le condizioni per cui questo si verifichi sono due:

ON UPDATE;
ON DELETE;

Posso applicare quattro comportamenti:

- NO ACTION (DEFAULT); se si prova ad eliminare/modificare un elemento, la query fallisce.
- SET NULL: se elimino/modifico un elemento, i campi nelle tabelle figlie vengono settati a null.
- SET DEFAULT: se elimino/modifico un elemento, i campi nelle tabelle figlie vengono settate al valore di default.
- CASCADE: Se Elimino/modifico un elemento, i campi nelle tabelle figlie vengono eliminati o aggiornati.

ESEMPIO:
Se modifico il campo campo_prova nella tabella esempio, i dati nel campo id_sec verranno messi a null, se invece elimino il campo tutto verrà eliminato.

```
CREATE TABLE <tabellaFiglia>(
   id_sec INT(11),
   PRIMARY KEY(id),
   FOREIGN KEY (id_sec) REFERENCES esempio(campo_prova)
      ON DELETE  cascade
      ON UPDATE  set null
);
```

Azioni e filtraggio sui dati:

Selezione di tutti i dati contenuti in una tabella:

```
SELECT * FROM <nomeTabella>
```

Esempio risultato:

```
+----+--------+---------+---------+------+--------------+----------+
| id | nome   | cognome | statura | peso | data_nascita | nazione  |
+----+--------+---------+---------+------+--------------+----------+
|  1 | Luca   | May     | 1.50    |   66 | 1991-01-01   | Svizzera |
|  2 | Maria  | Callas  | 1.45    |   95 | 1990-05-10   | Svizzera |
|  3 | Laura  | Pistol  | 1.90    |  110 | 1995-10-10   | Italia   |
+----+--------+---------+---------+------+--------------+----------+
```

Posso scegliere di ottenere solo alcuni elementi:

```
SELECT nome,cognome

FROM <nomeTabella>;
```

Esempio risultato:

```
+--------+---------+
| nome   | cognome |
+--------+---------+
| Luca   | May     |
| Maria  | Callas  |
| Laura  | Pistol  |
+--------+---------+
```

Oppure di rimuovere i duplicati:

```
SELECT DISTINCT <nomeCampo> FROM <nomeTabella>;
```

Posso anche applicare filtri più restrittivi, con più condizioni:

```
SELECT * FROM <nomeTabella>

WHERE nome= 'Luca'

AND cognome= 'May' AND statura > 1.40;
```

Esempio risultato:

```
+----+------+---------+---------+------+--------------+----------+
| id | nome | cognome | statura | peso | data_nascita | nazione  |
+----+------+---------+---------+------+--------------+----------+
| 1  | Luca |   May   |  1.50   |  66  |  1991-01-01  | Svizzera |
+----+------+---------+---------+------+--------------+----------+
```

Oppure anche applicare dei filtri più generici, per includere più istanze:

```
SELECT * FROM <nomeTabella>
WHERE statura >=1.50 AND statura <1.80
AND data_nascita BETWEEN 1991-01-01 AND 1995-10-10
AND nazione= 'Svizzera' OR nazione= 'Italia'
AND cognome LIKE 'M%' AND peso <> 110;
```

Esempio risultato:

```
+----+------+---------+---------+------+--------------+----------+
| id | nome | cognome | statura | peso | data_nascita | nazione  |
+----+------+---------+---------+------+--------------+----------+
|  1 | Luca |   May   |  1.50   |  66  |  1991-01-01  | Svizzera |
+----+------+---------+---------+------+--------------+----------+
```

→>=: maggiore uguale.
→BETWEEN x AND y: indica un range in cui devono essere compresi i valori(estremi inclusi) →OR: permette di selezionare due o più' valori specifici →LIKE: indica i campi che possiedono un cognome che inizi con 'M', e continui con vari caratteri. Il simbolo '%' indica infatti la presenza di un numero indefinito di caratteri, mentre '_' indica un singolo carattere. Quindi se uso:

cognome LIKE 'M__'; Restituirà
May.

→<>: permette di escludere un valore, (NOT=)

Posso anche decidere di mostrare solamente la struttura di una tabella, in questo modo:

```
DESCRIBE <nomeTabella>;
```

Modifica di dati già esistenti.

Per modificare un dato già presente nella tabella, devo fare:

```
UPDATE <nomeTabella>
SET <nomeCampo>=<val>
[WHERE <condizione>];
```

→UPDATE: permette di modificare dei valori già presenti nella tabella:

Per eliminare dei dati, devo usare il costrutto DELETE, nel seguente modo.

```
DELETE FROM <nomeTabella>
[WHERE <condizione>];
```

→ATTENZIONE: Se la condizione WHERE viene omessa vengono eliminati tutti i record nella tabella.

ESEMPIO:

```
DELETE FROM amici
 WHERE cognome = 'Rossi';
```

Posso anche ordinare dei dati per poi eliminarli, ad esempio:

```
DELETE FROM <nomeTabella>
ORDER BY <nomeCampo> ASC
[WHERE <condizione>];
```

→ORDER BY: permette appunto di ordinare dei dati, in questo caso in ordine ascendente (ASC).

Esiste anche un costrutto per svuotare una tabella completamente, chiamato TRUNCATE.

```
TRUNCATE TABLE <nomeTabella>;
```

Modifica di date.

Le date funzionano in maniera differente rispetto ai dati convenzionali, ecco alcuni operatori:

DATEDIFF():

```
DATEDIFF(<data1>,<data2>);
```

→DATEDIFF(): Permette di sottrarre la seconda data alla prima.

ESEMPIO:

```
DATEDIFF(NOW(),data_nascita)/365.25;
```

→NOW(): permette di avere la data attuale. →/365.25: DATEDIFF() restituisce un intervallo in giorni, facendo /365,25 trovo gli anni.

DATEADD():

```
DATEADD(<data1>,<data2>);
```

→DATEADD(): è il contrario di DATEDIFF, permette di aggiungere la seconda data alla prima.

ESEMPIO:

```
SET data=DATE_ADD(data, INTERVAL 2 YEAR);
```

→INTERVAL x YEAR: è un po' il contrario di DATEDIFF(), aggiunge un numero x di anni.

Specifiche di accesso:

Posso anche accedere ad una sola parte della data, per esempio:

```
<PARTE_DATA>(<camopData>);
```

ESEMPIO:

```
…

WHERE YEAR(data_nascita)= '2003';

…
```

→YEAR: In questo modo accedo solamente all'anno, tralasciando il resto.
→NB, posso usare anche MONTH e DAY.

Operatori di confronto speciali:

Posso anche utilizzare elementi per selezionare solo determinati dati, come il massimo, il minimo, ecc.
Questi comandi sono:
MAX,MIN,COUNT,SUM,AVG.

Tabella esempio DATI:

Temperatura
12
6
12
22
24

```
SELECT MAX(temperatura) FROM dati;
```

→Restituisce 24.

```
SELECT MIN(temperatura) FROM dati;
```

→Restituisce 6.

```
SELECT COUNT(temperatura) FROM dati;
```

→Restituisce 5.
Ci sono 5 istanze.

```
SELECT SUM(temperatura) FROM dati;
```

→Restituisce 76.
➔12+6+12+22+24=76

```
SELECT AVG(temperatura) FROM dati;
```

→Restituisce 15.2
➔(12+6+12+22+24)/5=15.2

Aggiungendo la keyword DISTINCT, i dati doppi verranno annullati, ecco un esempio:

```
SELECT COUNT( DISTINCT temperatura) FROM dati;
```

→Restituisce 4. ➔Ci sono 5 istanze, ma ‘12’ è presente 2 volte.

Concatenare più elementi.

Oltre a filtrare, può essere comodo attaccare più elementi, anche raggruppandoli sotto una colonna unica, per un maggiore ordine.

```
SELECT CONCAT(
<campo1>, ' ',<camp2>)[AS <nomeNuovaColonna>] FROM <nomeTabella>;
```

ESEMPIO:

```
SELECT CONCAT(
nome, ' ',cognome)[AS Dati_Anagarfici]
FROM <nomeTabella>;
```

```
+-----------------+
| Dati_Anagrafici |
+-----------------+
| Luca May        |
| Maria Callas    |
| Laura Pistol    |
+-----------------+
```

Esiste poi il costrutto IF che, funzionando un po'come in Excel, permette di porre delle condizioni e delle precise reazioni.

```
SELECT <nomeCampo>,
IF(<condizione>, <seVero>,<seFalso>)[AS <nomeNuovoCampo]
FROM <nomeTabella>;
```

ESEMPIO:

```
SELECT nome,
IF(statura>1.70,'Persona Alta','Persona Bassa')AS altezza
FROM atleta;
```

```
+--------+---------------+
| nome   | altezza       |
+--------+---------------+
| Luca   | Persona Bassa |
| Maria  | Persona Bassa |
| Laura  | Persona Alta  |
+--------+---------------+
```

L'importanza di AS

AS non è fondamentale, ma senza avremmo come risultato tabelle poco chiare, con il nome uguale alla query, ecco le due tabelle qui sopra senza AS:

```
+---------------------------+       +--------+-----------------------------------------------+
| concat( nome,' ',cognome)  |       | nome   | IF(statura>1.70,'Persona Alta','Persona Bassa') |
+---------------------------+       +--------+-----------------------------------------------+
| Luca May                  |       | Luca   | Persona Bassa                                 |
| Maria Callas              |       | Maria  | Persona Bassa                                 |
| Laura Pistol              |       | Laura  | Persona Alta                                  |
+---------------------------+       +--------+-----------------------------------------------+
```

La creazione di indici

Gli indici sono delle strutture che permettono di accedere molto più velocemente ad uno o più dati. Il loro utilizzo è consigliato principalmente su database molto grandi.

Infatti gli indici velocizzano sì le query, ma appesantiscono il database e rendono più lenta la fase di scrittura.

Ecco la sintassi per creare un indice su un campo di una tabella in fase di creazione:

```
CREATE TABLE <nomeTabella>(
 ...
 INDEX(<nomeIndice>),
 ...
);
```

Ecco invece la sintassi per creare un indice su un campo di una tabella già esistente:

```
CREATE INDEX <nomeIndice>
ON <nomeTabella> (<campoTabella>);
```

E per eliminarlo:

```
DROP INDEX <nomeIndice>
ON <nomeTabella>;
```

Si possono anche creare indici multipli:

```
CREATE INDEX <nomeIndice>
ON <nomeTabella> (<campo1>,<campo2>);
```

→Quando si crea un indice il database può impiegare molto tempo.
→Per convenzione, per i nomi degli indci, si tende ad usare <nomeCampoConIndice>_idx
(Es: nome_idx)

Raggruppamento, ordinamento e limite sui dati

SPESA

Articolo	Costo	citta
Tonno	3	Locarno
Pane	2	Lugano
Pane	2	Lugano
Tonno	4	Bellinzona
Latte	5	Locarno

Per raggruppare un insieme di dati devo usare il costrutto GROUP BY, ecco un esempio:
→Quanto ho speso in ogni città

```
SELECT citta, SUM(costo) [AS 'Costo totale']
FROM spesa GROUP BY citta;
```

Esempio risultato:

```
+------------+--------------+
| citta      | Costo totale |
+------------+--------------+
| Locarno    |            8 |
| Lugano     |            4 |
| Bellinzona |            4 |
+------------+--------------+
```

Raggruppamento

Spesso usare un WHERE per distinguere dei dati non funziona in questi casi, per questo utilizziamo un HAVING, abbinato ad una condizione.
→Quali sono gli articoli per i quali ho speso in totale almeno 5?←

```
SELECT articolo, SUM(costo) [AS 'Costo totale']
FROM spesa
GROUP BY articolo
HAVING SUM(costo)>=5;
```

Esempio risultato:

```
+----------+--------------+
| articolo | Costo totale |
+----------+--------------+
| Tonno    |            7 |
| Latte    |            5 |
+----------+--------------+
```

→HAVING è molto dispendioso in termini di prestazioni, perciò è fondamentale filtrare i dati con un WHERE prima.

Ordinamento e limite

Se si presenta l'occasione di dover ordinare dei dati, per esempio dal maggiore al minore devo usare ORDER BY, purtroppo per selezionare un unico dato(ad esempio il dato "più in alto", per un limite di MySql sono costretto a limitare i dati. È SCONSIGLIATO, IN QUANTO SE HO DUE DATI IDENTICI UNO VIENE PERSO.

```
SELECT*FROM spesa
ORDER BY articolo,costo DESC
LIMIT 3,3;
```

→DESC: è usato per mostrare i dati in ordine decrescente;
→ASC è il suo contrario, mostra i dati in ordine crescente.

Esempio risultato:

```
+----+----------+-------+------------+
| id | articolo | costo | citta      |
+----+----------+-------+------------+
|  4 | Tonno    |     4 | Bellinzona |
|  1 | Tonno    |     3 | Locarno    |
+----+----------+-------+------------+
```

Le Union

Un altro elemento di MySql che può tornare utile, sono le UNION.
Attraverso questa istruzione è possibile unire i risultati di più query all'interno di un'unica istruzione.

Per funzionare la UNION necessita di alcuni accorgimenti:

- Il numero delle colonne selezionate deve essere uguale in ciascuna delle query che si desidera unire;
- Il datataype delle colonne deve essere uguale (o convertibile), non è infatti possibile valori estratti da colonne aventi tipi di dato differenti (ad esempio INT e VARCHAR).

```
SELECT <nomeCampo>, <nomeCampo>
FROM <nomeTabella>
  UNION
SELECT <nomeCampo>, <nomeCampo>
FROM <nomeTabella>;
```

ESEMPIO:

```
SELECT id, nome
FROM atleta
  UNION
SELECT id, nome
FROM specialità;
```

Esempio risultato:

```
+----+-------------------+
| id | nome              |
+----+-------------------+
|  1 | Luca              |
|  2 | Maria             | →Query 1
|  3 | Laura             |
|  4 | Paolo             |
|  5 | Mara              |
|  6 | Luigia            |
|  1 | 100 m             |
|  2 | 400 m ad ostacoli | →Query 2
+----+-------------------+
```