EBook Gratuito

APPRENDIMENTO MySQL

Free unaffiliated eBook created from
Stack Overflow contributors.

#mysql

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mysql

It is an unofficial and free MySQL ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official MySQL.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Iniziare con MySQL

Osservazioni

MySQL è un RDBMS (Relational Database Management System) open source sviluppato e supportato da Oracle Corporation.

MySQL è supportato su un gran numero di piattaforme, incluse le varianti di Linux, OS X e Windows. Dispone inoltre di API per un gran numero di lingue, tra cui C, C ++, Java, Lua, .Net, Perl, PHP, Python e Ruby.

MariaDB è un fork di MySQL con un set di funzionalità leggermente diverso . È completamente compatibile con MySQL per la maggior parte delle applicazioni.

Versioni

Versione	Data di rilascio
1.0	1995/05/23
3.19	1996/12/01
3.20	1997-01-01
3.21	1998/10/01
3.22	1999/10/01
3.23	2001/01/22
4.0	2003/03/01
4.1	2004-10-01
5.0	2005-10-01
5.1	2008-11-27
5.5	2010-11-01
5.6	2013/02/01

Versione	Data di rilascio
5.7	2015/10/01

Examples

Iniziare

Creazione di un database in MySQL

```
CREATE DATABASE mydb;
```

Valore di ritorno:

> Domanda OK, 1 riga interessata (0,05 secondi)

Utilizzando il database creato `mydb`

```
USE mydb;
```

Valore di ritorno:

> Database modificato

Creare una tabella in MySQL

```
CREATE TABLE mytable
(
  id              int unsigned NOT NULL auto_increment,
  username        varchar(100) NOT NULL,
  email           varchar(100) NOT NULL,
  PRIMARY KEY     (id)
);
```

`CREATE TABLE mytable` creerà una nuova tabella chiamata `mytable` .

`id int unsigned NOT NULL auto_increment` crea la colonna `id` , questo tipo di campo assegnerà un ID numerico univoco a ciascun record nella tabella (nel senso che in questo caso nessuna riga può avere lo stesso `id`), MySQL assegnerà automaticamente un nuovo, valore univoco per il campo `id` del record (a partire da 1).

Valore di ritorno:

> Query OK, 0 righe interessate (0.10 sec)

Inserimento di una riga in una tabella MySQL

```
INSERT INTO mytable ( username, email )
```

```
VALUES ( "myuser", "myuser@example.com" );
```

Esempio valore di ritorno:

> Query OK, 1 riga interessata (0,06 sec)

Le `strings` `varchar` aka possono anche essere inserite usando le virgolette singole:

```
INSERT INTO mytable ( username, email )
VALUES ( 'username', 'username@example.com' );
```

Aggiornamento di una riga in una tabella MySQL

```
UPDATE mytable SET username="myuser" WHERE id=8
```

Esempio valore di ritorno:

> Query OK, 1 riga interessata (0,06 sec)

Il valore `int` può essere inserito in una query senza virgolette. Stringhe e date devono essere racchiusi in un'unica citazione `'` o doppi apici `"` .

Cancellare una riga in una tabella MySQL

```
DELETE FROM mytable WHERE id=8
```

Esempio valore di ritorno:

> Query OK, 1 riga interessata (0,06 sec)

Questo cancellerà la riga che ha `id` è 8.

Selezione delle righe in base alle condizioni in MySQL

```
SELECT * FROM mytable WHERE username = "myuser";
```

Valore di ritorno:

```
+----+----------+---------------------+
| id | username | email               |
+----+----------+---------------------+
|  1 | myuser   | myuser@example.com  |
+----+----------+---------------------+
```

> 1 riga in set (0,00 secondi)

Mostra l'elenco dei database esistenti

```
SHOW databases;
```

Valore di ritorno:

```
+--------------------+
| Databases          |
+--------------------+
| information_schema|
| mydb               |
+--------------------+
```

2 file in set (0,00 secondi)

Puoi pensare a "information_schema" come "master database" che fornisce accesso ai metadati del database.

Mostra tabelle in un database esistente

```
SHOW tables;
```

Valore di ritorno:

```
+----------------+
| Tables_in_mydb |
+----------------+
| mytable        |
+----------------+
```

1 riga in set (0,00 secondi)

Mostra tutti i campi di un tavolo

```
DESCRIBE databaseName.tableName;
```

o, se già si utilizza un database:

```
DESCRIBE tableName;
```

Valore di ritorno:

```
+-----------+----------------+--------+---------+--------------------+-------+
| Field     | Type           | Null   | Key     | Default            | Extra |
+-----------+----------------+--------+---------+--------------------+-------+
| fieldname | fieldvaluetype | NO/YES | keytype | defaultfieldvalue |       |
+-----------+----------------+--------+---------+--------------------+-------+
```

`Extra` può contenere `auto_increment` per esempio.

`Key` riferisce al tipo di chiave che può influire sul campo. Primaria (PRI), Unica (UNI) ...

    n riga in set (0,00 secondi)

Dove n è il numero di campi nella tabella.

Creazione dell'utente

Innanzitutto, è necessario creare un utente e quindi concedere le autorizzazioni dell'utente su determinati database / tabelle. Durante la creazione dell'utente, è inoltre necessario specificare da dove questo utente può connettersi.

```
CREATE USER 'user'@'localhost' IDENTIFIED BY 'some_password';
```

Creerà un utente che può connettersi solo sul computer locale in cui è ospitato il database.

```
CREATE USER 'user'@'%' IDENTIFIED BY 'some_password';
```

Creerà un utente in grado di connettersi da qualsiasi luogo (eccetto il computer locale).

Esempio valore di ritorno:

    Query OK, 0 righe interessate (0,00 secondi)

Aggiunta di privilegi

Concedere privilegi comuni, di base all'utente per tutte le tabelle del database specificato:

```
GRANT SELECT, INSERT, UPDATE ON databaseName.* TO 'userName'@'localhost';
```

Concedi tutti i privilegi all'utente per tutte le tabelle su tutti i database (attenzione con questo):

```
GRANT ALL ON *.* TO 'userName'@'localhost' WITH GRANT OPTION;
```

Come illustrato sopra, `*.*` Indirizza tutti i database e le tabelle, `databaseName.*` Indirizza tutte le tabelle del database specifico. È anche possibile specificare database e tabelle come `databaseName.tableName` .

`WITH GRANT OPTION` deve essere omesso se l'utente non deve essere in grado di concedere altri privilegi agli utenti.

I privilegi possono essere **entrambi**

```
ALL
```

o una combinazione di quanto segue, ciascuno separato da una virgola (elenco non esaustivo).

```
SELECT
```

```
INSERT
UPDATE
DELETE
CREATE
DROP
```

Nota

In generale, si dovrebbe cercare di evitare l'uso di nomi di colonne o tabelle contenenti spazi o l'utilizzo di parole riservate in SQL. Ad esempio, è meglio evitare nomi come `table` o `first name` .

Se devi usare questi nomi, mettili tra i delimitatori `` tick back tick `` . Per esempio:

```
CREATE TABLE `table`
(
   `first name` VARCHAR(30)
);
```

Una query contenente i delimitatori di back-tick su questa tabella potrebbe essere:

```
 SELECT `first name` FROM `table` WHERE `first name` LIKE 'a%';
```

Esempi di schemi di informazioni

processlist

Questo mostrerà tutte le query attive e in sospeso in quell'ordine, quindi per quanto tempo.

```
SELECT * FROM information_schema.PROCESSLIST ORDER BY INFO DESC, TIME DESC;
```

Questo è un po 'più dettagliato sui timeframe in quanto è in secondi per impostazione predefinita

```
SELECT ID, USER, HOST, DB, COMMAND,
TIME as time_seconds,
ROUND(TIME / 60, 2) as time_minutes,
ROUND(TIME / 60 / 60, 2) as time_hours,
STATE, INFO
FROM information_schema.PROCESSLIST ORDER BY INFO DESC, TIME DESC;
```

Ricerca di stored procedure

Cerca facilmente attraverso tutte le `Stored Procedures` per parole e caratteri jolly.

```
SELECT * FROM information_schema.ROUTINES WHERE ROUTINE_DEFINITION LIKE '%word%';
```

Leggi Iniziare con MySQL online: https://riptutorial.com/it/mysql/topic/302/iniziare-con-mysql

Capitolo 2: AGGIORNARE

Sintassi

- UPDATE [LOW_PRIORITY] [IGNORE] tableName SET column1 = expression1, column2 = expression2, ... [WHERE conditions]; // Semplice aggiornamento a tabella singola
- UPDATE [LOW_PRIORITY] [IGNORE] tableName SET column1 = expression1, column2 = expression2, ... [WHERE conditions] [ORDER BY expression [ASC | DESC]] [LIMIT row_count]; // Aggiornamento con ordine per e limite
- UPDATE [LOW_PRIORITY] [IGNORE] table1, table2, ... SET column1 = expression1, column2 = expression2, ... [WHERE conditions]; // Aggiornamento di più tabelle

Examples

Aggiornamento di base

Aggiornamento di una riga

```
UPDATE customers SET email='luke_smith@email.com' WHERE id=1
```

Questa query aggiorna il contenuto `email` nella tabella `customers` alla stringa `luke_smith@email.com` dove il valore di `id` è uguale a 1. I vecchi e nuovi contenuti della tabella del database sono illustrati di seguito rispettivamente a sinistra ea destra:

customers

id	firstname	lastname	email
1	Luke	Smith	luke@example.com
2	Anna	Carey	anna@example.com
3	Todd	Winters	todd@example.com

customers

id	firstname	lastname	email
1	Luke	Smith	luke_smith@email.com
2	Anna	Carey	anna@example.com
3	Todd	Winters	todd@example.com

Aggiornamento di tutte le righe

```
UPDATE customers SET lastname='smith'
```

Questa query aggiorna il contenuto del `lastname` per ogni voce nella tabella dei `customers` . I vecchi e nuovi contenuti della tabella del database sono illustrati di seguito rispettivamente a sinistra ea destra:

customers

id	firstname	lastname	email
1	Luke	Smith	luke@example.com
2	Anna	Carey	anna@example.com
3	Todd	Winters	todd@example.com

customers

id	firstname	lastname	email
1	Luke	Smith	luke@example.com
2	Anna	Smith	anna@example.com
3	Todd	Smith	todd@example.com

Avviso: è necessario utilizzare le clausole condizionali (WHERE) nella query UPDATE. Se non si utilizza alcuna clausola condizionale, verranno aggiornati tutti i record dell'attributo di quella tabella. Nell'esempio sopra, il nuovo valore (Smith) del cognome nella tabella clienti è impostato su tutte le righe.

Aggiorna con Join Pattern

Prendi in considerazione una tabella di produzione chiamata `questions_mysql` e una tabella `iwtQuestions` (worktable importato) che rappresenta l'ultimo batch di dati CSV importati da `LOAD DATA INFILE` . Il piano di lavoro viene troncato prima dell'importazione, i dati vengono importati e tale processo non viene mostrato qui.

Aggiorna i nostri dati di produzione utilizzando un join per i dati del nostro tavolo di lavoro importato.

```
UPDATE questions_mysql q -- our real table for production
join iwtQuestions i -- imported worktable
ON i.qId = q.qId
SET q.closeVotes = i.closeVotes,
q.votes = i.votes,
q.answers = i.answers,
q.views = i.views;
```

Gli alias `q` e `i` sono usati per abbreviare i riferimenti della tabella. Questo facilita lo sviluppo e la leggibilità.

`qId` , la chiave primaria, rappresenta l'id della domanda StackOverflow. Quattro colonne vengono aggiornate per le righe corrispondenti dal join.

AGGIORNA con ORDINA BY e LIMIT

Se la clausola `ORDER BY` è specificata nell'istruzione SQL di aggiornamento, le righe vengono aggiornate nell'ordine specificato.

Se la clausola `LIMIT` viene specificata nell'istruzione SQL, ciò pone un limite al numero di righe che possono essere aggiornate. Non c'è limite, se la clausola `LIMIT` non è specificata.

`ORDER BY` e `LIMIT` non possono essere utilizzati per l'aggiornamento di più tabelle.

La sintassi per l' `UPDATE` MySQL con `ORDER BY` e `LIMIT` è,

```
UPDATE [ LOW_PRIORITY ] [ IGNORE ]
tableName
SET column1 = expression1,
```

```
    column2 = expression2,
    ...
[WHERE conditions]
[ORDER BY expression [ ASC | DESC ]]
[LIMIT row_count];

---> Example
UPDATE employees SET isConfirmed=1 ORDER BY joiningDate LIMIT 10
```

Nell'esempio precedente, 10 righe verranno aggiornate in base all'ordine dei dipendenti che si `joiningDate` .

UPDATE su più tabelle

Nella tabella multipla `UPDATE` , aggiorna le righe in ogni tabella specificata che soddisfano le condizioni. Ogni riga corrispondente viene aggiornata una volta, anche se corrisponde alle condizioni più volte.

Nella tabella multipla `UPDATE` , `ORDER BY` e `LIMIT` non possono essere utilizzati.

La sintassi per multi tavolo `UPDATE` è,

```
UPDATE [LOW_PRIORITY] [IGNORE]
table1, table2, ...
    SET column1 = expression1,
        column2 = expression2,
        ...
    [WHERE conditions]
```

Ad esempio, considera due tabelle, `products` e `salesOrders` . Nel caso, riduciamo la quantità di un particolare prodotto dall'ordine di vendita che è già stato inserito. Quindi abbiamo anche bisogno di aumentare tale quantità nella nostra tabella dei `products` di magazzino. Questo può essere fatto in un'unica istruzione di aggiornamento SQL come di seguito.

```
UPDATE products, salesOrders
  SET salesOrders.Quantity = salesOrders.Quantity - 5,
      products.availableStock = products.availableStock + 5
WHERE products.productId = salesOrders.productId
  AND salesOrders.orderId = 100 AND salesOrders.productId = 20;
```

Nell'esempio sopra, la quantità "5" verrà ridotta dalla tabella `salesOrders` e la stessa verrà aumentata nella tabella `products` base alle condizioni `WHERE` .

Bulk UPDATE

Quando si aggiornano più righe con valori diversi, è molto più rapido utilizzare un aggiornamento collettivo.

```
UPDATE people
SET name =
  (CASE id WHEN 1 THEN 'Karl'
           WHEN 2 THEN 'Tom'
```

```
            WHEN 3 THEN 'Mary'
    END)
WHERE id IN (1,2,3);
```

Con l'aggiornamento alla rinfusa, è possibile inviare una sola query al server anziché una query per ogni riga da aggiornare. I casi dovrebbero contenere tutti i possibili parametri cercati nella clausola `WHERE` .

Leggi AGGIORNARE online: https://riptutorial.com/it/mysql/topic/2738/aggiornare

Capitolo 3: ALTER TABLE

Sintassi

- ALTER [IGNORE] TABLE tbl_name [**alter_specification** [, alter_specification] ...] [partition_options]

Osservazioni

```
    alter_specification: table_options
      | ADD [COLUMN] col_name column_definition [FIRST | AFTER col_name ]
      | ADD [COLUMN] (col_name column_definition,...)
      | ADD {INDEX|KEY} [index_name] [index_type] (index_col_name,...) [index_option] ...
      | ADD [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...) [index_option]
...
      | ADD [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [index_name] [index_type]
(index_col_name,...) [index_option] ...
      | ADD FULLTEXT [INDEX|KEY] [index_name] (index_col_name,...) [index_option] ...
      | ADD SPATIAL [INDEX|KEY] [index_name] (index_col_name,...) [index_option] ...
      | ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
reference_definition
      | ALGORITHM [=] {DEFAULT|INPLACE|COPY}
      | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
      | CHANGE [COLUMN] old_col_name new_col_name column_definition [FIRST|AFTER col_name]
      | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
      | MODIFY [COLUMN] col_name column_definition [FIRST | AFTER col_name]
      | DROP [COLUMN] col_name
      | DROP PRIMARY KEY
      | DROP {INDEX|KEY} index_name
      | DROP FOREIGN KEY fk_symbol
      | DISABLE KEYS
      | ENABLE KEYS
      | RENAME [TO|AS] new_tbl_name
      | RENAME {INDEX|KEY} old_index_name TO new_index_name
      | ORDER BY col_name [, col_name] ...
      | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
      | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
      | DISCARD TABLESPACE
      | IMPORT TABLESPACE
      | FORCE
      | {WITHOUT|WITH} VALIDATION
      | ADD PARTITION (partition_definition)
      | DROP PARTITION partition_names
      | DISCARD PARTITION {partition_names | ALL} TABLESPACE
      | IMPORT PARTITION {partition_names | ALL} TABLESPACE
      | TRUNCATE PARTITION {partition_names | ALL}
      | COALESCE PARTITION number
      | REORGANIZE PARTITION partition_names INTO (partition_definitions)
      | EXCHANGE PARTITION partition_name WITH TABLE tbl_name [{WITH|WITHOUT} VALIDATION]
      | ANALYZE PARTITION {partition_names | ALL}
      | CHECK PARTITION {partition_names | ALL}
      | OPTIMIZE PARTITION {partition_names | ALL}
      | REBUILD PARTITION {partition_names | ALL}
      | REPAIR PARTITION {partition_names | ALL}
      | REMOVE PARTITIONING
```

```
      | UPGRADE PARTITIONING
   index_col_name: col_name [(length)] [ASC | DESC]
   index_type: USING {BTREE | HASH}
   index_option: KEY_BLOCK_SIZE [=] value
      | index_type
      | WITH PARSER parser_name
      | COMMENT 'string'
```

`table_options: table_option [[,] table_option] ... (see options)` CREATE TABLE `options)`

`partition_options: (see options)` CREATE TABLE `options)`

Rif: MySQL 5.7 Manuale di riferimento / ... / ALTER TABLE Sintassi / 14.1.8 ALTER TABLE Sintassi

Examples

Cambiare il motore di archiviazione; ricostruire la tabella; cambia file_per_table

Ad esempio, se `t1` è attualmente una tabella InnoDB, questa istruzione cambia il suo motore di archiviazione in InnoDB:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

Se la tabella è già InnoDB, questo ricostruirà la tabella e i suoi indici e avrà un effetto simile a `OPTIMIZE TABLE` . Potresti ottenere un miglioramento dello spazio sul disco.

Se il valore di `innodb_file_per_table` è attualmente diverso dal valore in vigore quando `t1` stato creato, questo verrà convertito in (o da) file_per_table.

ALTER COLONNA DELLA TABELLA

```
CREATE DATABASE stackoverflow;

USE stackoverflow;

Create table stack(
    id_user int NOT NULL,
    username varchar(30) NOT NULL,
    password varchar(30) NOT NULL
);

ALTER TABLE stack ADD COLUMN submit date NOT NULL; -- add new column
ALTER TABLE stack DROP COLUMN submit; -- drop column
ALTER TABLE stack MODIFY submit DATETIME NOT NULL; -- modify type column
ALTER TABLE stack CHANGE submit submit_date DATETIME NOT NULL; -- change type and name of
column
ALTER TABLE stack ADD COLUMN mod_id INT NOT NULL AFTER id_user; -- add new column after
existing column
```

La tabella ALTER aggiunge INDICE

Per migliorare le prestazioni si potrebbe voler aggiungere indici alle colonne

```
ALTER TABLE TABLE_NAME ADD INDEX `index_name` (`column_name`)
```

alterazione per aggiungere indici compositi (a colonne multiple)

```
ALTER TABLE TABLE_NAME ADD INDEX `index_name` (`col1`,`col2`)
```

Modifica il valore di incremento automatico

La modifica di un valore di incremento automatico è utile quando non si desidera uno spazio vuoto in una colonna AUTO_INCREMENT dopo una cancellazione massiccia.

Ad esempio, hai ottenuto un sacco di righe indesiderate (pubblicità) pubblicate nella tua tabella, le hai cancellate e vuoi correggere il divario nei valori di incremento automatico. Supponiamo che il valore MAX della colonna AUTO_INCREMENT sia 100 ora. È possibile utilizzare quanto segue per correggere il valore di incremento automatico.

```
ALTER TABLE your_table_name AUTO_INCREMENT = 101;
```

Modifica del tipo di una colonna chiave primaria

```
ALTER TABLE fish_data.fish DROP PRIMARY KEY;
ALTER TABLE fish_data.fish MODIFY COLUMN fish_id DECIMAL(20,0) NOT NULL PRIMARY KEY;
```

Un tentativo di modificare il tipo di questa colonna senza prima rilasciare la chiave primaria comporterebbe un errore.

Cambia la definizione della colonna

Per cambiare la definizione di una colonna db, la query seguente può essere utilizzata, ad esempio, se abbiamo questo schema db

```
users (
    firstname varchar(20),
    lastname varchar(20),
    age char(2)
)
```

Per cambiare il tipo di colonna `age` da `char` a `int` , utilizziamo la query seguente:

```
ALTER TABLE users CHANGE age age tinyint UNSIGNED NOT NULL;
```

Il formato generale è:

```
ALTER TABLE table_name CHANGE column_name new_column_definition
```

Rinominare un database MySQL

Non esiste un singolo comando per rinominare un database MySQL, ma una soluzione semplice può essere utilizzata per ottenere ciò mediante il backup e il ripristino:

```
mysqladmin -uroot -p<password> create <new name>
mysqldump -uroot -p<password> --routines <old name> | mysql -uroot -pmypassword <new name>
mysqladmin -uroot -p<password> drop <old name>
```

passi:

1. Copia le righe sopra in un editor di testo.
2. Sostituisci tutti i riferimenti a `<old name>` , `<new name>` e `<password>` (+ facoltativamente `root` per utilizzare un utente diverso) con i valori pertinenti.
3. Esegui uno alla volta sulla riga di comando (supponendo che la cartella "bin" di MySQL si trovi nel percorso e inserisca "y" quando richiesto).

Passaggi alternativi:

Rinominare (spostare) ogni tabella da un db all'altro. Fatelo per ogni tabella:

```
RENAME TABLE `<old db>`.`<name>` TO  `<new db>`.`<name>`;
```

Puoi creare quelle affermazioni facendo qualcosa di simile

```
SELECT CONCAT('RENAME TABLE old_db.', table_name, ' TO ',
                          'new_db.', table_name)
    FROM information_schema.TABLES
    WHERE table_schema = 'old_db';
```

Avvertimento. Non tentare di creare alcun tipo di tabella o database semplicemente spostando i file sul file system. Questo ha funzionato bene ai vecchi tempi di MyISAM, ma nei nuovi giorni di InnoDB e tablespace, non funzionerà. Soprattutto quando il "Data Dictionary" viene spostato dal filesystem alle tabelle InnoDB del sistema, probabilmente nella prossima major release. Spostando (al contrario di solo `DROPping`) una `PARTITION` di una tabella InnoDB richiede l'uso di "tablespace trasportabili". Nel prossimo futuro, non ci sarà nemmeno un file da raggiungere.

Scambiare i nomi di due database MySQL

I seguenti comandi possono essere utilizzati per scambiare i nomi di due database MySQL (`<db1>` e `<db2>`):

```
mysqladmin -uroot -p<password> create swaptemp
mysqldump -uroot -p<password> --routines <db1> | mysql -uroot -p<password> swaptemp
mysqladmin -uroot -p<password> drop <db1>
mysqladmin -uroot -p<password> create <db1>
mysqldump -uroot -p<password> --routines <db2> | mysql -uroot -p<password> <db1>
```

```
mysqladmin -uroot -p<password> drop <db2>
mysqladmin -uroot -p<password> create <db2>
mysqldump -uroot -p<password> --routines swaptemp | mysql -uroot -p<password> <db2>
mysqladmin -uroot -p<password> drop swaptemp
```

passi:

1. Copia le righe sopra in un editor di testo.
2. Sostituisci tutti i riferimenti a `<db1>` , `<db2>` e `<password>` (+ facoltativamente `root` per utilizzare un utente diverso) con i valori pertinenti.
3. Esegui uno alla volta sulla riga di comando (supponendo che la cartella "bin" di MySQL si trovi nel percorso e inserisca "y" quando richiesto).

Rinominare una tabella MySQL

È possibile rinominare una tabella in un singolo comando:

```
RENAME TABLE `<old name>` TO `<new name>`;
```

La seguente sintassi fa esattamente la stessa cosa:

```
ALTER TABLE `<old name>` RENAME TO `<new name>`;
```

Se si rinomina una tabella temporanea, è necessario utilizzare la versione `ALTER TABLE` della sintassi.

passi:

1. Sostituisci `<old name>` e `<new name>` nella riga sopra con i valori pertinenti. *Nota: se la tabella viene spostata in un altro database, il* `dbname` *db.* `tablename` *sintassi* `tablename` *può essere utilizzata per* `<old name>` *e / o* `<new name>` *.*
2. Eseguilo sul database pertinente nella riga di comando MySQL o su un client come MySQL Workbench. *Nota: l'utente deve disporre dei privilegi ALTER e DROP sulla vecchia tabella e CREARE e INSERIRE su quello nuovo.*

Rinominare una colonna in una tabella MySQL

La ridenominazione di una colonna può essere eseguita in una singola istruzione ma, oltre al nuovo nome, è necessario specificare anche la "definizione della colonna" (ovvero il suo tipo di dati e altre proprietà facoltative come il nullability, l'incremento automatico ecc.).

```
ALTER TABLE `<table name>` CHANGE `<old name>` `<new name>` <column definition>;
```

passi:

1. Apri la riga di comando MySQL o un client come MySQL Workbench.
2. Esegui la seguente dichiarazione: `SHOW CREATE TABLE <table name>;` (sostituendo `<table name>` con il valore pertinente).

3. Annotare l'intera definizione di colonna per la colonna da rinominare *(ovvero tutto ciò che appare dopo il nome della colonna ma prima della virgola che lo separa dal nome della colonna successiva)* .
4. Sostituisci `<old name>` , `<new name>` e `<column definition>` nella riga sopra con i valori pertinenti e quindi eseguilo.

Leggi ALTER TABLE online: https://riptutorial.com/it/mysql/topic/2627/alter-table

Capitolo 4: Amministratore MySQL

Examples

Cambia password di root

```
mysqladmin -u root -p'old-password' password 'new-password'
```

Drop database

Utile per lo scripting per eliminare tutte le tabelle ed eliminare il database:

```
mysqladmin -u[username] -p[password] drop [database]
```

Usare con estrema cautela.

Per il database `DROP` come script SQL (è necessario il privilegio DROP su quel database):

```
DROP DATABASE database_name
```

o

```
DROP SCHEMA database_name
```

RENAME Atomico e Ricarica della Tavola

```
RENAME TABLE t TO t_old, t_copy TO t;
```

Nessun'altra sessione può accedere alle tabelle coinvolte mentre RENAME TABLE viene eseguito, quindi l'operazione di rinomina non è soggetta a problemi di concorrenza.

Atomic Rename è specialmente per ricaricare completamente un tavolo senza aspettare `DELETE` e caricare per finire:

```
CREATE TABLE new LIKE real;
load `new` by whatever means - LOAD DATA, INSERT, whatever
RENAME TABLE real TO old, new TO real;
DROP TABLE old;
```

Leggi Amministratore MySQL online: https://riptutorial.com/it/mysql/topic/2991/amministratore-mysql

Capitolo 5: apici inversi

Examples

Utilizzo di backtick

Ci sono molti esempi in cui vengono usati i backtick all'interno di una query, ma per molti non è ancora chiaro quando o dove usare backtick `` .

I backtick sono usati principalmente per prevenire un errore chiamato " *parola riservata MySQL* ". Quando si crea una tabella in PHPmyAdmin, a volte ci si trova di fronte a un avviso o allerta che si sta utilizzando una " *parola riservata MySQL* ".

Ad esempio quando crei una tabella con una colonna denominata " `group` " ottieni un avviso. Questo perché puoi fare la seguente query:

```
SELECT student_name, AVG(test_score) FROM student GROUP BY group
```

Per assicurarti di non avere un errore nella tua query, devi usare i backtick così la tua query diventa:

```
SELECT student_name, AVG(test_score) FROM student GROUP BY `group`
```

tavolo

Non solo i nomi delle colonne possono essere circondati da apici inversi, ma anche nomi di tabelle. Ad esempio quando devi `JOIN` più tabelle.

```
SELECT `users`.`username`, `groups`.`group` FROM `users`
```

Più facile da leggere

Come puoi vedere usando i backtick attorno ai nomi di tabelle e colonne, anche la query risulta più facile da leggere.

Ad esempio quando sei abituato a scrivere querys tutto in minuscolo:

```
select student_name, AVG(test_score) from student group by group
select `student_name`, AVG(`test_score`) from `student` group by `group`
```

Si prega di consultare la pagina del manuale MySQL intitolata Parole chiave e parole riservate . Quelli con una (R) sono parole riservate. Gli altri sono semplicemente parole chiave. Il Riservato richiede particolare attenzione.

Leggi apici inversi online: https://riptutorial.com/it/mysql/topic/5208/apici-inversi

Capitolo 6: Aritmetica

Osservazioni

MySQL, sulla maggior parte delle macchine, utilizza l' aritmetica in virgola mobile IEEE 754 a 64 bit per i suoi calcoli.

Nei contesti interi utilizza l'aritmetica intera.

- `RAND()` non è un generatore di numeri casuali perfetto. Viene principalmente utilizzato per generare rapidamente numeri pseudocasuali

Examples

Operatori aritmetici

MySQL fornisce i seguenti operatori aritmetici

Operatore	Nome	Esempio
`+`	aggiunta	`SELECT 3+5;` -> 8 `SELECT 3.5+2.5;` -> 6.0 `SELECT 3.5+2;` -> 5.5
`-`	Sottrazione	`SELECT 3-5;` -> -2
`*`	Moltiplicazione	`SELECT 3 * 5;` -> 15
`/`	Divisione	`SELECT 20 / 4;` -> 5 `SELECT 355 / 113;` -> 3.1416 `SELECT 10.0 / 0;` -> NULL
`DIV`	Divisione intera	`SELECT 5 DIV 2;` -> 2
`%` o `MOD`	Modulo	`SELECT 7 % 3;` -> 1 `SELECT 15 MOD 4` -> 3 `SELECT 15 MOD -4` -> 3 `SELECT -15 MOD 4` -> -3 `SELECT -15 MOD -4` -> -3 `SELECT 3 MOD 2.5` -> 0.5

BIGINT

Se i numeri nella tua aritmetica sono tutti interi, MySQL usa il tipo di dati intero `BIGINT` (firmato a 64 bit) per fare il suo lavoro. Per esempio:

`select (1024 * 1024 * 1024 * 1024 *1024 * 1024) + 1` -> 1.152.921.504.606.846.977

e

`select (1024 * 1024 * 1024 * 1024 *1024 * 1024 * 1024` -> `BIGINT` errore fuori intervallo

DOPPIO

Se alcuni numeri nella tua aritmetica sono frazionari, MySQL utilizza l' aritmetica in virgola mobile IEEE 754 a 64 bit . È necessario prestare attenzione quando si utilizza l'aritmetica in virgola mobile, poiché molti numeri in virgola mobile sono, intrinsecamente, approssimazioni piuttosto che valori esatti .

Costanti matematiche

Pi

Quanto segue riporta il valore di `PI` formattato a 6 posizioni decimali. Il valore attuale è buono a `DOUBLE` ;

```
SELECT PI();    -> 3.141593
```

Trigonometria (SIN, COS)

Gli angoli sono in radianti, non in gradi. Tutti i calcoli sono fatti in virgola mobile a 64 bit IEEE 754 . Tutti i calcoli in virgola mobile sono soggetti a piccoli errori, noti come errori di macchina ε (epsilon) , quindi evitare di provare a confrontarli per l'uguaglianza. Non c'è modo di evitare questi errori quando si utilizza il punto mobile; sono integrati nella tecnologia.

Se si utilizzano i valori `DECIMAL` nei calcoli trigonometrici, vengono convertiti implicitamente in virgola mobile e quindi nuovamente in decimali.

Seno

Restituisce il seno di un numero X espresso in radianti

```
SELECT SIN(PI()); -> 1.2246063538224e-16
```

Coseno

Restituisce il coseno di X quando X è dato in radianti

```
SELECT COS(PI()); -> -1
```

Tangente

Restituisce la tangente di un numero X espressa in radianti. Si noti che il risultato è molto vicino allo zero, ma non esattamente zero. Questo è un esempio di macchina ε.

```
SELECT TAN(PI());   -> -1.2246063538224e-16
```

Arco Coseno (coseno inverso)

Restituisce l'arcocoseno di X se X è nell'intervallo da `-1 to 1`

```
SELECT ACOS(1);    -> 0
SELECT ACOS(1.01); -> NULL
```

Arc Sine (sinusoidale inverso)

Restituisce l'arcoseno di X se X è nell'intervallo da `-1 to 1`

```
SELECT ASIN(0.2); -> 0.20135792079033
```

Arco tangente (tangente inversa)

`ATAN(x)` restituisce l'arco tangente di un singolo numero.

```
SELECT ATAN(2); -> 1.1071487177941
```

`ATAN2(X, Y)` restituisce l'arco tangente delle due variabili X e Y. È simile al calcolo dell'arco tangente di Y / X. Ma è numericamente più robusto: t funziona correttamente quando X è vicino allo zero e i segni di entrambi gli argomenti vengono utilizzati per determinare il quadrante del risultato.

Le migliori pratiche suggeriscono di scrivere le formule per utilizzare `ATAN2()` anziché `ATAN()` laddove possibile.

```
 ATAN2(1,1);    -> 0.7853981633974483 (45 degrees)
 ATAN2(1,-1);   -> 2.356194490192345  (135 degrees)
 ATAN2(0, -1);  -> PI  (180 degrees)  don't try ATAN(-1 / 0)... it won't work
```

Cotangente

Restituisce la cotangente di X

```
SELECT COT(12); -> -1.5726734063977
```

Conversione

```
SELECT RADIANS(90) -> 1.5707963267948966
SELECT SIN(RADIANS(90)) -> 1
SELECT DEGREES(1), DEGREES(PI()) -> 57.29577951308232, 180
```

Arrotondamento (ROUND, FLOOR, CEIL)

Arrotondare un numero decimale a un valore intero

Per valori numerici esatti (ad es. `DECIMAL`): se la prima cifra decimale di un numero è 5 o superiore, questa funzione arrotonda un numero al numero intero successivo *lontano da zero* . Se la cifra decimale è 4 o inferiore, questa funzione arriverà al successivo valore intero *più prossimo allo zero* .

```
SELECT ROUND(4.51) -> 5
SELECT ROUND(4.49) -> 4
SELECT ROUND(-4.51) -> -5
```

Per valori numerici approssimativi (ad es. `DOUBLE`): il risultato della funzione `ROUND()` dipende dalla libreria C; su molti sistemi, ciò significa che `ROUND()` utilizza la regola *pari round to the più vicina* :

```
SELECT ROUND(45e-1) -> 4  -- The nearest even value is 4
SELECT ROUND(55e-1) -> 6  -- The nearest even value is 6
```

Arrotonda un numero

Per arrotondare un numero utilizzare la funzione `CEIL()` o `CEILING()`

```
SELECT CEIL(1.23)    -> 2
SELECT CEILING(4.83) -> 5
```

Arrotondare un numero

Per arrotondare un numero, utilizzare la funzione `FLOOR()`

```
SELECT FLOOR(1.99) -> 1
```

PAVIMENTO e CEIL vanno verso / fuori da -infinità:

```
SELECT FLOOR(-1.01), CEIL(-1.01) -> -2 and -1
SELECT FLOOR(-1.99), CEIL(-1.99) -> -2 and -1
```

Arrotondare un numero decimale in un numero specificato

di posizioni decimali.

```
SELECT ROUND(1234.987, 2) -> 1234.99
SELECT ROUND(1234.987, -2) -> 1200
```

Si applica anche la discussione verso l'alto e verso il basso e "5".

Alza un numero a una potenza (POW)

Per aumentare un numero `x` ad una potenza `y` , utilizzare le funzioni `POW()` o `POWER()`

```
SELECT POW(2,2); => 4
SELECT POW(4,2); => 16
```

Radice quadrata (SQRT)

Utilizzare la funzione `SQRT()` . Se il numero è negativo, verrà restituito `NULL`

```
SELECT SQRT(16); -> 4
SELECT SQRT(-3); -> NULL
```

Numeri casuali (RAND)

Genera un numero casuale

Per generare un numero in virgola mobile pseudocasuale tra `0` e `1` , utilizzare la funzione `RAND()`

Supponiamo di avere la seguente domanda

```
SELECT i, RAND() FROM t;
```

Questo restituirà qualcosa di simile

io	RAND ()
1	,6191438870682
2	,93845168309142
3	,83482678498591

Numero casuale in un intervallo

Per generare un numero casuale nell'intervallo a <= n <= b, puoi utilizzare la seguente formula

```
FLOOR(a + RAND() * (b - a + 1))
```

Ad esempio, questo genererà un numero casuale tra 7 e 12

```
SELECT FLOOR(7 + (RAND() * 6));
```

Un modo semplice per restituire casualmente le righe in una tabella:

```
SELECT * FROM tbl ORDER BY RAND();
```

Questi sono numeri **pseudocasuali** .

Il generatore di numeri pseudocasuali in MySQL non è crittograficamente sicuro. Cioè, se usi MySQL per generare numeri casuali da usare come segreti, un determinato avversario che sa che hai usato MySQL sarà in grado di indovinare i tuoi segreti più facilmente di quanto tu possa credere.

Valore assoluto e segno (ABS, SIGN)

Restituisce il valore assoluto di un numero

```
SELECT ABS(2);   -> 2
SELECT ABS(-46); -> 46
```

Il `sign` di un numero lo paragona a 0.

Cartello	Risultato	Esempio
-1	n <0	`SELECT SIGN(42);` -> 1
0	n = 0	`SELECT SIGN(0);` -> 0
1	n> 0	`SELECT SIGN(-3);` -> -1

```
SELECT SIGN(-423421); -> -1
```

Leggi Aritmetica online: https://riptutorial.com/it/mysql/topic/4516/aritmetica

Capitolo 7: Backup con mysqldump

Sintassi

- mysqldump -u [nome utente] -p [password] [altre opzioni] nome_db> dumpFileName.sql /// Per eseguire il backup di un singolo database
- mysqldump -u [nome utente] -p [password] [altre opzioni] nome_db [nome_tbl tbl_nome2 nome_tbl2 ...]> dumpFileName.sql /// Per eseguire il backup di una o più tabelle
- mysqldump -u [nome utente] -p [password] [altre opzioni] - database db_name1 db_name2 nome_db3> dumpFileName.sql /// Per eseguire il backup di uno o più database completi
- mysqldump -u [nome utente] -p [password] [altre opzioni] --all-databases> dumpFileName.sql /// Per eseguire il backup dell'intero server MySQL

Parametri

Opzione	Effetto
-	# Opzioni di accesso al server
`-h` (`--host`)	Host (indirizzo IP o nome host) a cui connettersi. L'impostazione predefinita è `localhost` (`127.0.0.1`) Esempio: `-h localhost`
`-u` (`--user`)	Utente MySQL
`-p` (`--password`)	Password MySQL. **Importante** : quando si utilizza `-p` , non deve esserci uno spazio tra l'opzione e la password. Esempio: `-pMyPassword`
-	# Opzioni di scarico
`--add-drop-database`	Aggiungi un'istruzione `DROP DATABASE` prima di ogni istruzione `CREATE DATABASE` . Utile se si desidera sostituire i database nel server.
`--add-drop-table`	Aggiungi un'istruzione `DROP TABLE` prima di ogni istruzione `CREATE TABLE` . Utile se si desidera sostituire le tabelle nel server.
`--no-create-db`	Sopprimere le istruzioni `CREATE DATABASE` nel dump. Ciò è utile quando sei sicuro che il (i) database (i) che stai scaricando esiste già (s) nel server in cui caricerai il dump.
`-t` (`--no-create-info`)	Elimina tutte le istruzioni `CREATE TABLE` nel dump. Ciò è utile quando si desidera scaricare solo i dati dalle tabelle e utilizzerà il file di dump per popolare tabelle identiche in un altro database / server.
`-d` (`--no-data`)	Non scrivere informazioni sulla tabella. Questo scaricherà solo le istruzioni `CREATE TABLE` . Utile per creare database "modello"

Opzione	Effetto
`-R` (`--routines`)	Includere stored procedure / funzioni nel dump.
`-K` (`--disable-keys`)	Disabilitare le chiavi per ogni tabella prima di inserire i dati e abilitare le chiavi dopo aver inserito i dati. Ciò accelera gli inserimenti solo nelle tabelle MyISAM con indici non univoci.

Osservazioni

L'output di un'operazione `mysqldump` è un file leggermente commentato contenente istruzioni SQL sequenziali compatibili con la versione delle utilità MySQL utilizzate per generarlo (con attenzione alla compatibilità con le versioni precedenti, ma nessuna garanzia per quelle future). Pertanto, il ripristino di un database `mysqldump` ed include l'esecuzione di tali istruzioni. Generalmente, questo file

- `DROP` s la prima tabella o vista specificata
- `CREATE` s quella tabella o vista
- Per le tabelle scaricate con dati (cioè senza l'opzione `--no-data`)
    - `LOCK` il tavolo
    - `INSERT` tutte le righe della tabella originale in un'unica istruzione
- `UNLOCK TABLES`
- Ripete il precedente per tutte le altre tabelle e viste
- `DROP` s la prima routine inclusa
- `CREATE` questa routine
- Ripete lo stesso per tutte le altre routine

La presenza del `DROP` prima di `CREATE` per ogni tabella indica che se lo schema è presente, indipendentemente dal fatto che sia vuoto o meno, l'utilizzo di un file `mysqldump` per il suo ripristino compila o sovrascrive i dati in esso contenuti.

Examples

Creazione di un backup di un database o una tabella

Crea un'istantanea di un intero database:

```
mysqldump [options] db_name > filename.sql
```

Creare uno snapshot di più database:

```
mysqldump [options] --databases db_name1 db_name2 ... > filename.sql
mysqldump [options] --all-databases > filename.sql
```

Crea uno snapshot di una o più tabelle:

```
mysqldump [options] db_name table_name... > filename.sql
```

Crea uno snapshot *escludendo* una o più tabelle:

```
mysqldump [options] db_name --ignore-table=tbl1 --ignore-table=tbl2 ... > filename.sql
```

L'estensione del file `.sql` è completamente una questione di stile. Qualsiasi estensione funzionerebbe.

Specifica nome utente e password

```
> mysqldump -u username -p [other options]
Enter password:
```

Se è necessario specificare la password sulla riga di comando (ad esempio in uno script), è possibile aggiungerla dopo l'opzione `-p` *senza* spazio:

```
> mysqldump -u username -ppassword [other options]
```

Se la password contiene spazi o caratteri speciali, ricorda di usare l'escape a seconda della shell / sistema.

Opzionalmente la forma estesa è:

```
> mysqldump --user=username --password=password [other options]
```

(L'esplicita specificazione della password sulla riga di comando non è consigliata a causa di problemi di sicurezza.)

Ripristino di un backup di un database o di una tabella

```
mysql [options] db_name < filename.sql
```

Nota che:

- `db_name` deve essere un database esistente;
- il tuo utente autenticato ha privilegi sufficienti per eseguire tutti i comandi all'interno del tuo `filename.sql` ;
- L'estensione del file `.sql` è completamente una questione di stile. Qualsiasi estensione funzionerebbe.
- Non è possibile specificare un nome tabella da caricare in anche se è possibile specificarne uno da cui eseguire il dump. Questo deve essere fatto all'interno di `filename.sql` .

In alternativa, quando lo **strumento riga di comando MySQL** , è possibile ripristinare (o eseguire qualsiasi altro script) utilizzando il comando di origine:

```
source filename.sql
```

o

```
\. filename.sql
```

mysqldump da un server remoto con compressione

Per utilizzare la compressione sul filo per un trasferimento più veloce, passare l'opzione `--compress` a `mysqldump` . Esempio:

```
mysqldump -h db.example.com -u username -p --compress dbname > dbname.sql
```

Importante: se non si desidera bloccare il db di *origine* , è necessario includere anche `--lock-tables=false` . Ma potresti non ottenere un'immagine db coerente internamente in questo modo.

Per salvare anche il file compresso, puoi inviare pipe a `gzip` .

```
mysqldump -h db.example.com -u username -p --compress dbname | gzip --stdout > dbname.sql.gz
```

ripristinare un file mysqldump gzip senza decomprimere

```
gunzip -c dbname.sql.gz | mysql dbname -u username -p
```

Nota: `-c` significa scrivere output su stdout.

Backup diretto su Amazon S3 con compressione

Se si desidera eseguire un backup completo di una grande installazione MySql e non si dispone di una memoria locale sufficiente, è possibile scaricarla e comprimerla direttamente su un bucket Amazon S3. È anche una buona pratica farlo senza avere la password del DB come parte del comando:

```
mysqldump -u root -p --host=localhost --opt --skip-lock-tables --single-transaction \
        --verbose --hex-blob --routines --triggers --all-databases |
    gzip -9 | s3cmd put - s3://s3-bucket/db-server-name.sql.gz
```

Viene richiesta la password, dopodiché viene avviato il backup.

Trasferimento dei dati da un server MySQL a un altro

Se hai bisogno di copiare un database da un server a un altro, hai due opzioni:

Opzione 1:

1. Memorizza il file di dump nel server di origine
2. Copia il file dump sul tuo server di destinazione
3. Carica il file di dump nel tuo server di destinazione

Sul server di origine:

```
mysqldump [options] > dump.sql
```

Sul server di destinazione, copiare il file di dump ed eseguire:

```
mysql [options] < dump.sql
```

Opzione 2:

Se il server di destinazione può connettersi al server host, è possibile utilizzare una pipeline per copiare il database da un server all'altro:

Sul server di destinazione

```
mysqldump [options to connect to the source server] | mysql [options]
```

Allo stesso modo, lo script potrebbe essere eseguito sul server di origine, spingendo verso la destinazione. In entrambi i casi, è probabile che sia significativamente più veloce dell'opzione 1.

Database di backup con stored procedure e funzioni

Per impostazione predefinita, stored procedure e funzioni o non generate da `mysqldump` , è necessario aggiungere il parametro `--routines` (o `-R`):

```
mysqldump -u username -p -R db_name > dump.sql
```

Quando si utilizza `--routines` la creazione e la modifica dei timestamp non vengono mantenute, si dovrebbe invece scaricare e ricaricare il contenuto di `mysql.proc` .

Leggi Backup con mysqldump online: https://riptutorial.com/it/mysql/topic/604/backup-con-mysqldump

Capitolo 8: Cambia la password

Examples

Cambia la password di root di MySQL in Linux

Per cambiare la password dell'utente root di MySQL:

Passaggio 1: Arresta il server MySQL.

- in Ubuntu o Debian:
  `sudo /etc/init.d/mysql stop`
- in CentOS, Fedora o Red Hat Enterprise Linux:
  `sudo /etc/init.d/mysqld stop`

Passaggio 2: avviare il server MySQL senza il sistema dei privilegi.

```
sudo mysqld_safe --skip-grant-tables &
```

o, se `mysqld_safe` non è disponibile,

```
sudo mysqld --skip-grant-tables &
```

Passaggio 3: connettersi al server MySQL.

```
mysql -u root
```

Passaggio 4: impostare una nuova password per l'utente root.

5.7

```
FLUSH PRIVILEGES;
ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
FLUSH PRIVILEGES;
exit;
```

5.7

```
FLUSH PRIVILEGES;
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new_password');
FLUSH PRIVILEGES;
exit;
```

Nota: la sintassi `ALTER USER` stata introdotta in MySQL 5.7.6.

Passaggio 5: riavvia il server MySQL.

- in Ubuntu o Debian:
  `sudo /etc/init.d/mysql stop`

```
sudo /etc/init.d/mysql start
```

- in CentOS, Fedora o Red Hat Enterprise Linux:

```
sudo /etc/init.d/mysqld stop
sudo /etc/init.d/mysqld start
```

Cambia la password di root di MySQL in Windows

Quando vogliamo cambiare la password di root in windows, dobbiamo seguire i seguenti passi:

Passaggio 1: avviare il prompt dei comandi utilizzando uno dei seguenti metodi:

Perss `Crtl+R` o Goto `Start Menu > Run` quindi digita `cmd` e premi invio

Passaggio 2: Cambia la directory in cui è installato `MYSQL` , nel mio caso lo è

```
C:\> cd C:\mysql\bin
```

Passo 3: Ora dobbiamo avviare il prompt dei comandi di `mysql`

```
C:\mysql\bin> mysql -u root mysql
```

Passaggio 4: interrogazione incendio per modificare `root` password di `root`

```
mysql> SET PASSWORD FOR root@localhost=PASSWORD('my_new_password');
```

Processi

1. Interrompere il processo server / demone MySQL (mysqld).
2. Avvia il server MySQL elabora l'opzione --skip-grant-tables in modo che non `mysqld_safe --skip-grant-tables &` password: `mysqld_safe --skip-grant-tables &`
3. Connetti al server MySQL come utente root: `mysql -u root`
4. Cambia la password:

- (5.7.6 e successive): `ALTER USER 'root'@'localhost' IDENTIFIED BY 'new-password';`
- (5.7.5 e versioni precedenti o MariaDB): `SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new-password'); flush privileges; quit;`

5. Riavvia il server MySQL.

Nota: questo funzionerà solo se si è fisicamente sullo stesso server.

Documento online: http://dev.mysql.com/doc/refman/5.7/en/resetting-permissions.html

Leggi Cambia la password online: https://riptutorial.com/it/mysql/topic/2761/cambia-la-password

Capitolo 9: CARICARE DATI INFIL

Sintassi

1. CARICARE DATI [LOW_PRIORITY | CONCORRENTE] [LOCAL] INFILE 'nome_file'
2. INTO TABLE nome_bloc
3. [Set caratteri CARATTERE]
4. [{FIELDS | COLUMNS} [TERMINATO DA 'string'] [[OPTIONALLY] ENCLOSED BY 'char']]
5. [LINES [STARTING BY 'string'] [TERMINATO DA 'stringa']]
6. [Numero IGNORE {LINEE | RIGHE}]
7. [(Col_name_or_user_var, ...)]
8. [SET col_name = expr, ...]

Examples

utilizzando LOAD DATA INFILE per caricare grandi quantità di dati nel database

Considera il seguente esempio assumendo che tu abbia un ";" - CSV delimitato da caricare nel tuo database.

```
1;max;male;manager;12-7-1985
2;jack;male;executive;21-8-1990
.
.
.
1000000;marta;female;accountant;15-6-1992
```

Crea la tabella per l'inserimento.

```
CREATE TABLE `employee` ( `id` INT NOT NULL ,
                          `name` VARCHAR NOT NULL,
                          `sex` VARCHAR NOT NULL ,
                          `designation` VARCHAR NOT NULL ,
                          `dob` VARCHAR NOT NULL   );
```

Utilizzare la seguente query per inserire i valori in quella tabella.

```
LOAD DATA INFILE 'path of the file/file_name.txt'
INTO TABLE employee
FIELDS TERMINATED BY ';' //specify the delimiter separating the values
LINES TERMINATED BY '\r\n'
(id,name,sex,designation,dob)
```

Considera il caso in cui il formato della data non è standard.

```
1;max;male;manager;17-Jan-1985
```

```
2;jack;male;executive;01-Feb-1992
.
.
.
1000000;marta;female;accountant;25-Apr-1993
```

In questo caso è possibile modificare il formato della colonna di `dob` prima di inserirlo in questo modo.

```
LOAD DATA INFILE 'path of the file/file_name.txt'
INTO TABLE employee
FIELDS TERMINATED BY ';' //specify the delimiter separating the values
LINES TERMINATED BY '\r\n'
(id,name,sex,designation,@dob)
SET date = STR_TO_DATE(@date, '%d-%b-%Y');
```

Questo esempio di LOAD DATA INFILE non specifica tutte le funzionalità disponibili.

Potete vedere altre referenze su LOAD DATA INFILE qui .

Importa un file CSV in una tabella MySQL

Il comando seguente importa i file CSV in una tabella MySQL con le stesse colonne rispettando le regole di quotatura e di escape di CSV.

```
load data infile '/tmp/file.csv'
into table my_table
fields terminated by ','
optionally enclosed by '"'
escaped by '"'
lines terminated by '\n'
ignore 1 lines; -- skip the header row
```

Carica i dati con i duplicati

Se si utilizza il comando `LOAD DATA INFILE` per compilare una tabella con dati esistenti, si scoprirà spesso che l'importazione non riesce a causa di duplicati. Esistono diversi modi per superare questo problema.

CARICARE DATI LOCALI

Se questa opzione è stata abilitata nel server, può essere utilizzata per caricare un file esistente sul computer client anziché sul server. Un effetto collaterale è che le righe duplicate per valori unici vengono ignorate.

```
LOAD DATA LOCAL INFILE 'path of the file/file_name.txt'
INTO TABLE employee
```

CARICARE DATI INFILE 'fname' SOSTITUIRE

Quando viene utilizzata la parola chiave replace, le chiavi univoche o primarie duplicate comporteranno la sostituzione della riga esistente con nuove

```
LOAD DATA INFILE 'path of the file/file_name.txt'
REPLACE INTO TABLE employee
```

CARICARE DATI INFILE 'fname' IGNORA

L'opposto di `REPLACE` , le righe esistenti verranno mantenute e quelle nuove ignorate. Questo comportamento è simile a `LOCAL` descritto in precedenza. Tuttavia il file non deve esistere sul computer client.

```
LOAD DATA INFILE 'path of the file/file_name.txt'
IGNORE INTO TABLE employee
```

Carica tramite tabella intermedia

A volte ignorare o sostituire tutti i duplicati potrebbe non essere l'opzione ideale. Potrebbe essere necessario prendere decisioni in base al contenuto di altre colonne. In tal caso, l'opzione migliore è caricare in una tabella intermedia e trasferire da lì.

```
INSERT INTO employee SELECT * FROM intermediary WHERE ...
```

importazione / esportazione

importare

```
SELECT a,b,c  INTO OUTFILE 'result.txt' FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n' FROM table;
```

Esportare

```
LOAD DATA INFILE 'result.txt' INTO TABLE table;
```

Leggi CARICARE DATI INFIL online: https://riptutorial.com/it/mysql/topic/2356/caricare-dati-infil

Capitolo 10: Client MySQL

Sintassi

- mysql [OPZIONI] [database_name]

Parametri

Parametro	Descrizione
`-D --database=name`	nome del database
`--delimiter=str`	imposta il delimitatore dell'istruzione. Quello predefinito è ';'
`-e --execute='command'`	eseguire il comando
`-h --host=name`	nome host a cui connettersi
`-p --password=name`	password *Nota: non c'è spazio tra `-p` e la password*
`-p` (senza password)	verrà richiesta la password
`-P --port=#`	numero di porta
`-s --silent`	modalità silenziosa, produce meno output. Usa `\t` come separatore di colonne
`-ss`	come `-s` , ma ometti i nomi delle colonne
`-S --socket=path`	specificare il socket (Unix) o named pipe (Windows) da utilizzare durante la connessione a un'istanza locale
`--skip-column-names`	ometti i nomi delle colonne
`-u --user=name`	nome utente
`-U --safe-updates --i-am-a-dummy`	accedi con la variabile `sql_safe_updates=ON` . Ciò consentirà solo `DELETE` e `UPDATE` che utilizzano esplicitamente le chiavi
`-V --version`	stampare la versione ed uscire

Examples

Accesso di base

Per accedere a MySQL dalla riga di comando:

```
mysql --user=username --password=pwd --host=hostname test_db
```

Questo può essere abbreviato in:

```
mysql -u username -p password -h hostname test_db
```

Omettendo il valore della `password` , MySQL chiederà la password richiesta come primo input. Se si specifica la `password` il client visualizzerà un avviso "non sicuro":

```
mysql -u=username -p -h=hostname test_db
```

Per le connessioni locali `--socket` può essere utilizzato per puntare al file socket:

```
mysql --user=username --password=pwd --host=localhost --socket=/path/to/mysqld.sock test_db
```

Se si omette il parametro `socket` , il client tenterà di collegarsi a un server sul computer locale. Il server deve essere in esecuzione per connettersi ad esso.

Esegui comandi

Questo insieme di esempi mostra come eseguire comandi memorizzati in stringhe o file di script, senza la necessità del prompt interattivo. Ciò è particolarmente utile quando uno script di shell deve interagire con un database.

Esegui il comando da una stringa

```
$ mysql -uroot -proot test -e'select * from people'

+----+-------+--------+
| id | name  | gender |
+----+-------+--------+
|  1 | Kathy | f      |
|  2 | John  | m      |
+----+-------+--------+
```

Per formattare l'output come una griglia separata da tabulazioni, utilizzare il parametro `--silent` :

```
$ mysql -uroot -proot test -s -e'select * from people'

id      name    gender
1       Kathy   f
2       John    m
```

Per omettere le intestazioni:

```
$ mysql -uroot -proot test -ss -e'select * from people'

1       Kathy   f
2       John    m
```

Esegui dal file di script:

```
$ mysql -uroot -proot test < my_script.sql
```

```
$ mysql -uroot -proot test -e'source my_script.sql'
```

Scrivi l'output su un file

```
$ mysql -uroot -proot test < my_script.sql > out.txt

$ mysql -uroot -proot test -s -e'select * from people' > out.txt
```

Leggi Client MySQL online: https://riptutorial.com/it/mysql/topic/5619/client-mysql

Capitolo 11: Clustering

Examples

disambiguation

Disambiguazione "MySQL Cluster" ...

- NDB Cluster: un motore specializzato, principalmente in memoria. Non molto usato
- Galera Cluster aka Percona XtraDB Cluster aka PXC aka MariaDB con Galera. - Un'ottima soluzione ad alta disponibilità per MySQL; va oltre la replica.

Vedi le singole pagine su quelle varianti di "Cluster".

Per "indice cluster" vedere le pagine su `PRIMARY KEY` .

Leggi Clustering online: https://riptutorial.com/it/mysql/topic/5130/clustering

Capitolo 12: Codici di errore

Examples

Codice errore 1064: errore di sintassi

```
select LastName, FirstName,
from Person
```

Restituisce il messaggio:

> Codice di errore: 1064. Si è verificato un errore nella sintassi SQL; controlla il manuale che corrisponde alla tua versione del server MySQL per la sintassi corretta da usare vicino a 'da Persona' alla riga 2.

Ottenere un messaggio "1064 error" da MySQL significa che la query non può essere analizzata senza errori di sintassi. In altre parole, non può dare un senso alla query.

La citazione nel messaggio di errore inizia con il primo carattere della query che MySQL non riesce a capire come analizzare. In questo esempio MySQL non ha senso, nel contesto, di `from Person` . In questo caso, c'è una virgola extra immediatamente prima `from Person` . La virgola indica a MySQL di aspettarsi un'altra descrizione di colonna nella clausola `SELECT`

Un errore di sintassi dice sempre `... near '...'` . La cosa all'inizio delle virgolette è molto vicina a dove si trova l'errore. Per individuare un errore, guarda il primo token tra virgolette e all'ultimo token prima delle virgolette.

A volte avrai `... near ''` ; cioè, niente tra virgolette. Ciò significa che il primo carattere che MySQL non riesce a capire è proprio alla fine o all'inizio della dichiarazione. Ciò suggerisce che la query contiene virgolette non bilanciate (`'` o `"`) o parentesi non bilanciate o che non hai terminato correttamente la dichiarazione.

Nel caso di una Stored Routine, potresti aver dimenticato di usare correttamente `DELIMITER` .

Quindi, quando ottieni l'errore 1064, guarda il testo della query e trova il punto menzionato nel messaggio di errore. Controlla visivamente il testo della query proprio intorno a quel punto.

Se chiedi a qualcuno di aiutarti a risolvere l'errore 1064, è meglio fornire sia il testo dell'intera query sia il testo del messaggio di errore.

Codice errore 1175: Aggiornamento sicuro

Questo errore viene visualizzato durante il tentativo di aggiornare o eliminare i record senza includere la clausola `WHERE` che utilizza la colonna `KEY` .

Per eseguire comunque l'eliminazione o l'aggiornamento, digitare:

```
SET SQL_SAFE_UPDATES = 0;
```

Per attivare nuovamente la modalità provvisoria, digita:

```
SET SQL_SAFE_UPDATES = 1;
```

Codice errore 1215: impossibile aggiungere un vincolo di chiave esterna

Questo errore si verifica quando le tabelle non sono strutturate in modo adeguato per gestire la verifica rapida dei requisiti `FK` (Foreign Key) richiesti dallo sviluppatore.

```
CREATE TABLE `gtType` (
  `type` char(2) NOT NULL,
  `description` varchar(1000) NOT NULL,
  PRIMARY KEY (`type`)
) ENGINE=InnoDB;

CREATE TABLE `getTogethers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `type` char(2) NOT NULL,
  `eventDT` datetime NOT NULL,
  `location` varchar(1000) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_gt2type` (`type`), -- see Note1 below
  CONSTRAINT `gettogethers_ibfk_1` FOREIGN KEY (`type`) REFERENCES `gtType` (`type`)
) ENGINE=InnoDB;
```

Nota 1: un KEY come questo verrà creato automaticamente se necessario a causa della definizione FK nella riga che lo segue. Lo sviluppatore può saltarlo e la KEY (nota anche come indice) verrà aggiunta se necessario. Un esempio di esso che viene saltato dallo sviluppatore è mostrato sotto in `someOther` .

Fin qui tutto bene, fino alla chiamata di sotto.

```
CREATE TABLE `someOther` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `someDT` datetime NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `someOther_dt` FOREIGN KEY (`someDT`) REFERENCES `getTogethers` (`eventDT`)
) ENGINE=InnoDB;
```

> Codice errore: 1215. Impossibile aggiungere un vincolo di chiave esterna

In questo caso fallisce a causa della mancanza di un indice nella tabella di *riferimento* `getTogethers` per gestire la rapida ricerca di un `eventDT` . Per essere risolto nella prossima dichiarazione.

```
CREATE INDEX `gt_eventdt` ON getTogethers (`eventDT`);
```

Table `getTogethers` è stato modificato e ora la creazione di `someOther` avrà successo.

Dalla pagina del manuale MySQL usando i vincoli FOREIGN KEY :

> MySQL richiede indici su chiavi esterne e chiavi di riferimento in modo che i controlli delle chiavi esterne possano essere veloci e non richiedere una scansione della tabella. Nella tabella di riferimento, deve esistere un indice in cui le colonne chiave esterna sono elencate come prime colonne nello stesso ordine. Tale indice viene creato automaticamente sulla tabella di riferimento se non esiste.
>
> Le colonne corrispondenti nella chiave esterna e la chiave di riferimento devono avere tipi di dati simili. La dimensione e il segno dei tipi interi devono essere uguali. La lunghezza dei tipi di stringa non deve essere la stessa. Per le colonne di stringa non binarie (carattere), il set di caratteri e le regole di confronto devono essere uguali.
>
> InnoDB consente a una chiave esterna di fare riferimento a qualsiasi colonna di indice o gruppo di colonne. Tuttavia, nella tabella di riferimento, deve esistere un indice in cui le colonne di riferimento sono elencate come prime colonne nello stesso ordine.

Si noti che l'ultimo punto sopra riguarda le prime (a sinistra) colonne e la mancanza di un requisito di chiave primaria (sebbene altamente consigliato).

Dopo aver creato correttamente una tabella di *riferimento* (figlio), tutte le chiavi che sono state create automaticamente per te sono visibili con un comando come il seguente:

```
SHOW CREATE TABLE someOther;
```

Altri casi comuni di questo errore includono, come menzionato sopra, ma dovrebbero essere evidenziati:

- Differenze apparentemente banali in `INT` che è firmata, che punta verso `INT UNSIGNED` .
- Sviluppatori che hanno difficoltà a comprendere i KEY a più colonne (compositi) e i primi requisiti di ordinazione (a sinistra).

1045 Accesso negato

Vedi le discussioni in "GRANT" e "Recovering root password".

1236 "posizione impossibile" nella replica

Di solito questo significa che il Master è `sync_binlog` crash e che `sync_binlog` era OFF. La soluzione è `CHANGE MASTER to POS=0` del prossimo file binlog (vedere Master) sullo Slave.

La causa: il Master invia elementi di replica allo Slave prima di eseguire il flushing al suo binlog (quando `sync_binlog=OFF`). Se il Master si blocca prima del flush, lo Slave ha già spostato logicamente la fine del file sul binlog. Quando il Master si riavvia, avvia un nuovo binlog, quindi CAMBIARE all'inizio di quel binlog è la migliore soluzione disponibile.

Una soluzione a lungo termine è `sync_binlog=ON` , se puoi permetterti l'I / O extra che causa.

(Se stai utilizzando GTID, ...?)

2002, 2003 Impossibile connettersi

Verificare la presenza di un problema del firewall che blocca la porta 3306.

Alcune possibili diagnosi e / o soluzioni

- Il server è effettivamente in esecuzione?
- "service firewalld stop" e "systemctl disable firewalld"
- telnet master 3306
- Controlla l' `bind-address`
- controlla `skip-name-resolve`
- controlla la presa.

1067, 1292, 1366, 1411 - Valore errato per numero, data, valore predefinito, ecc.

1067 Questo è probabilmente correlato ai valori predefiniti di `TIMESTAMP` , che sono cambiati nel tempo. Vedi i `TIMESTAMP defaults` nella pagina Date e orari. (che non esiste ancora)

1292/1366 DOUBLE / Integer Verifica la presenza di lettere o altri errori di sintassi. Verifica che le colonne siano allineate; forse pensi che stai mettendo in un `VARCHAR` ma è allineato con una colonna numerica.

1292 DATETIME Controlla troppo nel passato o nel futuro. Controlla tra le 2:00 e le 3:00 di mattina quando è stato modificato il risparmio di luce diurna. Verifica la sintassi errata, ad esempio `+00` fuso orario.

1292 VARIABLE Verificare i valori consentiti per la `VARIABLE` si sta tentando di `SET` .

1292 CARICHI DATI Guarda la linea che è "cattiva". Controlla i simboli di fuga, ecc. Guarda i tipi di dati.

1411 STR_TO_DATE Data di formattazione errata?

126, 127, 134, 144, 145

Quando provi ad accedere ai record dal database MySQL, potresti ricevere questi messaggi di errore. Questi messaggi di errore si sono verificati a causa della corruzione nel database MySQL. Di seguito sono riportati i tipi

```
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Bug di MySQL, attacco di virus, crash del server, arresto improprio, tabella danneggiata sono la

causa di questa corruzione. Quando viene danneggiato, diventa inaccessibile e non è più possibile accedervi. Per ottenere l'accessibilità, il modo migliore per recuperare i dati da un backup aggiornato. Tuttavia, se non si dispone di un backup aggiornato o di un backup valido, è possibile utilizzare MySQL Repair.

Se il tipo di motore tabella è `MyISAM` , applicare `CHECK TABLE` , quindi `REPAIR TABLE` su di esso.

Quindi pensa seriamente alla conversione in InnoDB, quindi questo errore non si ripeterà più.

Sintassi

```
CHECK TABLE <table name> ////To check the extent of database corruption
REPAIR TABLE <table name> ////To repair table
```

139

L'errore 139 può significare che il numero e la dimensione dei campi nella definizione della tabella superano alcuni limiti. soluzioni alternative:

- Rifletti lo schema
- Normalizza alcuni campi
- Partizionare verticalmente la tabella

1366

Questo di solito significa che la gestione del set di caratteri non era coerente tra client e server. Vedi ... per ulteriore assistenza.

126, 1054, 1146, 1062, 24

(prendersi una pausa) Con l'inclusione di quei 4 numeri di errore, penso che questa pagina avrà coperto circa il 50% degli errori tipici che gli utenti ottengono.

(Sì, questo 'Esempio' necessita di revisione.)

24 Impossibile aprire il file (Troppi file aperti)

`open_files_limit` proviene da un'impostazione del sistema operativo. `table_open_cache` deve essere inferiore a quello.

Questi possono causare quell'errore:

- Mancato `DEALLOCATE PREPARE` in una stored procedure.
- Tabelle PARTITIONed con un gran numero di partizioni e innodb_file_per_table = ON. Consiglia di non avere più di 50 partizioni in una data tabella (per vari motivi). (Quando "Native Partitions" diventa disponibile, questo consiglio può cambiare).

L'ovvia soluzione è di aumentare il limite del sistema operativo: per consentire più file, modificare

`ulimit` o `/etc/security/limits.conf` o in `sysctl.conf` (kern.maxfiles e kern.maxfilesperproc) o qualcos'altro (dipendente dal sistema operativo). Quindi aumenta `open_files_limit` e `table_open_cache` .

A partire da 5.6.8, `open_files_limit` è dimensionato automaticamente in base a `max_connections` , ma è OK cambiarlo dal valore predefinito.

1062 - Inserimento duplicato

Questo errore si verifica principalmente a causa dei seguenti due motivi

1. *Valore duplicato* - `Error Code: 1062. Duplicate entry '12' for key 'PRIMARY'`

   La colonna della chiave primaria è unica e non accetta la voce duplicata. Quindi, quando stai provando ad inserire una nuova riga che è già presente nella tua tabella, produrrà questo errore.

   Per risolvere questo, impostare la colonna chiave primaria come `AUTO_INCREMENT` . E quando stai provando ad inserire una nuova riga, ignora la colonna della chiave primaria o inserisci il valore `NULL` sulla chiave primaria.

```
CREATE TABLE userDetails(
  userId INT(10) NOT NULL AUTO_INCREMENT,
  firstName VARCHAR(50),
  lastName VARCHAR(50),
  isActive INT(1) DEFAULT 0,
  PRIMARY KEY (userId) );

--->and now while inserting
INSERT INTO userDetails VALUES (NULL ,'John', 'Doe', 1);
```

2. *Campo dati univoci* - `Error Code: 1062. Duplicate entry 'A' for key 'code'`

   È possibile assegnare una colonna come univoca e provare a inserire una nuova riga con un valore già esistente per quella colonna produrrà questo errore.

   Per superare questo errore, usa `INSERT IGNORE` invece di `INSERT` normale. Se la nuova riga che stai tentando di inserire non duplica un record esistente, MySQL lo inserisce come al solito. Se il record è un duplicato, la parola chiave `IGNORE` lo `IGNORE` senza generare alcun errore.

```
INSERT IGNORE INTO userDetails VALUES (NULL ,'John', 'Doe', 1);
```

Leggi Codici di errore online: https://riptutorial.com/it/mysql/topic/895/codici-di-errore

Capitolo 13: Commento Mysql

Osservazioni

L' `--` stile di commento, che richiede uno spazio finale, si differenzia nel comportamento dallo standard SQL , che non richiede lo spazio.

Examples

Aggiungere commenti

Esistono tre tipi di commenti:

```
 # This comment continues to the end of line

-- This comment continues to the end of line

/* This is an in-line comment */

/*
This is a
multiple-line comment
*/
```

Esempio:

```
SELECT * FROM t1; -- this is comment

CREATE TABLE stack(
    /*id_user int,
    username varchar(30),
    password varchar(30)
    */
    id int
);
```

Il metodo `--` richiede che uno spazio segua il `--` prima che inizi il commento, altrimenti verrà interpretato come un comando e di solito causa un errore.

```
#This comment works
/*This comment works.*/
--This comment does not.
```

Definizioni delle tabelle di commento

```
CREATE TABLE menagerie.bird (
    bird_id INT NOT NULL AUTO_INCREMENT,
    species VARCHAR(300) DEFAULT NULL COMMENT 'You can include genus, but never subspecies.',
    INDEX idx_species (species) COMMENT 'We must search on species often.',
```

```
    PRIMARY KEY (bird_id)
) ENGINE=InnoDB COMMENT 'This table was inaugurated on February 10th.';
```

Usare an `=` dopo `COMMENT` è opzionale. (Documenti ufficiali)

Questi commenti, a differenza degli altri, vengono salvati con lo schema e possono essere recuperati tramite `SHOW CREATE TABLE` o from `information_schema` .

Leggi Commento Mysql online: https://riptutorial.com/it/mysql/topic/2337/commento-mysql

Capitolo 14: Configurazione della connessione SSL

Examples

Installazione per sistemi basati su Debian

(Questo presuppone che MySQL sia stato installato e che `sudo` sia in uso.)

Generazione di una CA e chiavi SSL

Assicurarsi che OpenSSL e le librerie siano installate:

```
apt-get -y install openssl
apt-get -y install libssl-dev
```

Quindi fai e inserisci una directory per i file SSL:

```
mkdir /home/ubuntu/mysqlcerts
cd /home/ubuntu/mysqlcerts
```

Per generare chiavi, creare un'autorità di certificazione (CA) per firmare le chiavi (autofirmato):

```
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600 -key ca-key.pem -out ca.pem
```

I valori immessi in ciascun prompt non influiscono sulla configurazione. Quindi creare una chiave per il server e firmare utilizzando la CA di prima:

```
openssl req -newkey rsa:2048 -days 3600 -nodes -keyout server-key.pem -out server-req.pem
openssl rsa -in server-key.pem -out server-key.pem

openssl x509 -req -in server-req.pem -days 3600 -CA ca.pem -CAkey ca-key.pem -set_serial 01 -
out server-cert.pem
```

Quindi crea una chiave per un cliente:

```
openssl req -newkey rsa:2048 -days 3600 -nodes -keyout client-key.pem -out client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -req -in client-req.pem -days 3600 -CA ca.pem -CAkey ca-key.pem -set_serial 01 -
out client-cert.pem
```

Per assicurarti che tutto sia stato impostato correttamente, verifica i tasti:

```
openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

Aggiungere le chiavi a MySQL

Apri il file di configurazione di MySQL . Per esempio:

```
vim /etc/mysql/mysql.conf.d/mysqld.cnf
```

Sotto la sezione `[mysqld]` , aggiungi le seguenti opzioni:

```
ssl-ca = /home/ubuntu/mysqlcerts/ca.pem
ssl-cert = /home/ubuntu/mysqlcerts/server-cert.pem
ssl-key = /home/ubuntu/mysqlcerts/server-key.pem
```

Riavvia MySQL. Per esempio:

```
service mysql restart
```

Testare la connessione SSL

Connetti allo stesso modo, passando le opzioni extra `ssl-ca` , `ssl-cert` e `ssl-key` , usando la chiave del client generata. Ad esempio, assumendo `cd /home/ubuntu/mysqlcerts` :

```
mysql --ssl-ca=ca.pem --ssl-cert=client-cert.pem --ssl-key=client-key.pem -h 127.0.0.1 -u
superman -p
```

Dopo aver effettuato l'accesso, verificare che la connessione sia effettivamente sicura:

```
superman@127.0.0.1 [None]> SHOW VARIABLES LIKE '%ssl%';
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| have_openssl  | YES                                     |
| have_ssl      | YES                                     |
| ssl_ca        | /home/ubuntu/mysqlcerts/ca.pem          |
| ssl_capath    |                                         |
| ssl_cert      | /home/ubuntu/mysqlcerts/server-cert.pem |
| ssl_cipher    |                                         |
| ssl_crl       |                                         |
| ssl_crlpath   |                                         |
| ssl_key       | /home/ubuntu/mysqlcerts/server-key.pem  |
+---------------+-----------------------------------------+
```

Puoi anche controllare:

```
superman@127.0.0.1 [None]> STATUS;
...
SSL:                    Cipher in use is DHE-RSA-AES256-SHA
...
```

Applicare SSL

Questo è via `GRANT` , usando `REQUIRE SSL` :

```
GRANT ALL PRIVILEGES ON *.* TO 'superman'@'127.0.0.1' IDENTIFIED BY 'pass' REQUIRE SSL;
FLUSH PRIVILEGES;
```

Ora, `superman` *deve* connettersi tramite SSL.

Se non si desidera gestire le chiavi del client, utilizzare la chiave del client da prima e utilizzarla automaticamente per tutti i client. Aprire il file di configurazione MySQL , ad esempio:

```
vim /etc/mysql/mysql.conf.d/mysqld.cnf
```

Sotto la sezione `[client]` , aggiungi le seguenti opzioni:

```
ssl-ca = /home/ubuntu/mysqlcerts/ca.pem
ssl-cert = /home/ubuntu/mysqlcerts/client-cert.pem
ssl-key = /home/ubuntu/mysqlcerts/client-key.pem
```

Ora `superman` deve solo digitare quanto segue per accedere tramite SSL:

```
mysql -h 127.0.0.1 -u superman -p
```

Il collegamento da un altro programma, ad esempio in Python, richiede in genere solo un parametro aggiuntivo per la funzione di connessione. Un esempio di Python:

```
import MySQLdb
ssl = {'cert': '/home/ubuntu/mysqlcerts/client-cert.pem', 'key':
'/home/ubuntu/mysqlcerts/client-key.pem'}
conn = MySQLdb.connect(host='127.0.0.1', user='superman', passwd='imsoawesome', ssl=ssl)
```

Riferimenti e ulteriori letture:

- https://www.percona.com/blog/2013/06/22/setting-up-mysql-ssl-and-secure-connections/
- https://lowendbox.com/blog/getting-started-with-mysql-over-ssl/
- http://xmodulo.com/enable-ssl-mysql-server-client.html
- https://ubuntuforums.org/showthread.php?t=1121458

Installazione per CentOS7 / RHEL7

Questo esempio presuppone due server:

1. dbserver (dove vive il nostro database)
2. appclient (dove vivono le nostre applicazioni)

FWIW, entrambi i server sono Enforcing di SELinux.

Innanzitutto, accedere a dbserver

Creare una directory temporanea per la creazione dei certificati.

```
mkdir /root/certs/mysql/ && cd /root/certs/mysql/
```

Creare i certificati del server

```
openssl genrsa 2048 > ca-key.pem
openssl req -sha1 -new -x509 -nodes -days 3650 -key ca-key.pem > ca-cert.pem
openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout server-key.pem > server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
openssl x509 -sha1 -req -in server-req.pem -days 730  -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 > server-cert.pem
```

Sposta i certificati del server in / etc / pki / tls / certs / mysql /

Il percorso della directory assume CentOS o RHEL (regolare come necessario per altre distribuzioni):

```
mkdir /etc/pki/tls/certs/mysql/
```

Assicurati di impostare i permessi sulla cartella e sui file. mysql richiede piena proprietà e accesso.

```
chown -R mysql:mysql /etc/pki/tls/certs/mysql
```

Ora configura MySQL / MariaDB

```
# vi /etc/my.cnf
# i
[mysqld]
bind-address=*
ssl-ca=/etc/pki/tls/certs/ca-cert.pem
ssl-cert=/etc/pki/tls/certs/server-cert.pem
ssl-key=/etc/pki/tls/certs/server-key.pem
# :wq
```

Poi

```
systemctl restart mariadb
```

Non dimenticare di aprire il firewall per consentire le connessioni da appclient (usando IP 1.2.3.4)

```
firewall-cmd --zone=drop --permanent --add-rich-rule 'rule family="ipv4" source
address="1.2.3.4" service name="mysql" accept'
# I force everything to the drop zone.  Season the above command to taste.
```

Ora riavvia firewalld

```
service firewalld restart
```

Quindi, accedere al server mysql di dbserver:

```
mysql -uroot -p
```

Emettere quanto segue per creare un utente per il client. nota RICHIESTA SSL nell'istruzione GRANT.

```
GRANT ALL PRIVILEGES ON *.* TO 'iamsecure'@'appclient' IDENTIFIED BY 'dingdingding' REQUIRE
SSL;
FLUSH PRIVILEGES;
# quit mysql
```

Dovresti essere ancora in / root / certs / mysql dal primo passaggio. In caso contrario, ritorna ad esso per uno dei comandi seguenti.

Creare i certificati client

```
openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout client-key.pem > client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -sha1 -req -in client-req.pem -days 730 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 > client-cert.pem
```

Nota : ho usato lo stesso nome comune per entrambi i certificati server e client. YMMV.

Assicurati di essere ancora / root / certs / mysql / per questo comando successivo

Combina il certificato CA del server e del client in un singolo file:

```
cat server-cert.pem client-cert.pem > ca.pem
```

Assicurati di vedere due certificati:

```
cat ca.pem
```

FINE DEL SERVER LAVORO LATERALE PER ORA.

Aprire un altro terminale e

```
ssh appclient
```

Come prima, creare una casa permanente per i certificati client

```
mkdir /etc/pki/tls/certs/mysql/
```

Ora, posiziona i certificati client (creati su dbserver) su appclient. Puoi scoppiarli o semplicemente copiare e incollare i file uno per uno.

```
scp dbserver
# copy files from dbserver to appclient
# exit scp
```

Anche in questo caso, assicurarsi di impostare le autorizzazioni sulla cartella e sui file. mysql richiede piena proprietà e accesso.

```
chown -R mysql:mysql /etc/pki/tls/certs/mysql
```

Dovresti avere tre file, ognuno di proprietà dell'utente mysql:

```
/etc/pki/tls/certs/mysql/ca.pem
/etc/pki/tls/certs/mysql/client-cert.pem
/etc/pki/tls/certs/mysql/client-key.pem
```

Ora modifica la configurazione MariaDB / MySQL di appclient nella sezione `[client]` .

```
vi /etc/my.cnf
# i
[client]
ssl-ca=/etc/pki/tls/certs/mysql/ca.pem
ssl-cert=/etc/pki/tls/certs/mysql/client-cert.pem
ssl-key=/etc/pki/tls/certs/mysql/client-key.pem
# :wq
```

Riavvia il servizio mariadb dell'appclient:

```
systemctl restart mariadb
```

ancora sul cliente qui

Questo dovrebbe restituire: ssl TRUE

```
mysql --ssl --help
```

Ora, accedi all'istanza mysql di appclient

```
mysql -uroot -p
```

Dovrebbe vedere SÌ per entrambe le variabili di seguito

```
show variables LIKE '%ssl';
    have_openssl    YES
    have_ssl              YES
```

Inizialmente ho visto

```
have_openssl NO
```

Una rapida occhiata a mariadb.log ha rivelato:

> Errore SSL: impossibile ottenere il certificato da '/etc/pki/tls/certs/mysql/client-cert.pem'

Il problema era che root-owned client-cert.pem e la cartella contenente. La soluzione era impostare la proprietà di / etc / pki / tls / certs / mysql / su mysql.

```
chown -R mysql:mysql /etc/pki/tls/certs/mysql
```

Riavvia mariadb se necessario dal passaggio immediatamente precedente

ORA, SIAMO PRONTI A TESTARE IL COLLEGAMENTO SICURO

Siamo ancora in appclient qui

Tentativo di connessione all'istanza mysql di dbserver utilizzando l'account creato sopra.

```
mysql -h dbserver -u iamsecure -p
# enter password dingdingding (hopefully you changed that to something else)
```

Con un po 'di fortuna dovresti essere loggato senza errori.

Per confermare di essere connessi con SSL abilitato, emettere il seguente comando dal prompt MariaDB / MySQL:

```
\s
```

Questo è un backslash s, ovvero lo stato

Questo mostrerà lo stato della tua connessione, che dovrebbe assomigliare a questo:

```
Connection id:          4
Current database:
Current user:         iamsecure@appclient
SSL:             Cipher in use is DHE-RSA-AES256-GCM-SHA384
Current pager:          stdout
Using outfile:          ''
Using delimiter:    ;
Server:            MariaDB
Server version:         5.X.X-MariaDB MariaDB Server
Protocol version:    10
Connection:         dbserver via TCP/IP
```

```
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:        3306
Uptime:            42 min 13 sec
```

Se ti viene negata l'autorizzazione durante il tentativo di connessione, controlla la tua dichiarazione GRANT sopra per assicurarti che non ci siano caratteri vaganti o segni.

Se hai errori SSL, torna su questa guida per assicurarti che i passaggi siano ordinati.

Questo ha funzionato su RHEL7 e probabilmente funzionerà anche su CentOS7. Non posso confermare se questi passaggi esatti funzioneranno altrove.

Spero che questo salvi qualcun altro un po 'di tempo e di irritazione.

Leggi Configurazione della connessione SSL online:
https://riptutorial.com/it/mysql/topic/7563/configurazione-della-connessione-ssl

Capitolo 15: Configurazione e messa a punto

Osservazioni

La configurazione avviene in uno dei 3 modi:

- opzioni della riga di comando
- il file di configurazione `my.cnf`
- impostazione delle variabili dall'interno del server

Le opzioni della riga di comando hanno la forma `mysqld --long-parameter-name=value --another-parameter` . Gli stessi parametri possono essere inseriti nel file di configurazione `my.conf` . *Alcuni* parametri sono configurabili utilizzando le variabili di sistema da MySQL. Controlla la documentazione ufficiale per un elenco completo dei parametri.

Le variabili possono avere dash `-` o underscore `_` . Gli spazi possono esistere attorno al `=` . I numeri grandi possono essere suffissi da `K` , `M` , `G` per kilo-, mega- e giga-. Un'impostazione per riga.

Flags: solitamente `ON` e `1` sono sinonimi, idem per `OFF` e `0` . Alcune bandiere non hanno nulla dietro di loro.

Quando si inseriscono le impostazioni in `my.cnf` , tutte le impostazioni per il *server* devono essere nella sezione `[mysqld]` , quindi non aggiungere ciecamente le impostazioni alla fine del file. (Nota: per gli strumenti che consentono a più istanze di mysql di condividere un my.cnf, i nomi delle sezioni potrebbero essere diversi).

Examples

Prestazioni InnoDB

Ci sono centinaia di impostazioni che possono essere inserite in my.cnf. Per l'utente "lite" di MySQL, non contano tanto.

Una volta che il tuo database diventa non banale, è consigliabile impostare i seguenti parametri:

```
innodb_buffer_pool_size
```

Questo dovrebbe essere impostato su circa il 70% della RAM *disponibile* (se si dispone di almeno 4 GB di RAM, una percentuale inferiore se si dispone di una minuscola VM o macchina antica). L'impostazione controlla la quantità di cache utilizzata da InnoDB ENGINE. Quindi, è molto importante per le prestazioni di InnoDB.

Parametro per consentire l'inserimento di dati enormi

Se è necessario memorizzare immagini o video nella colonna, è necessario modificare il valore in

base alle esigenze dell'applicazione

max_allowed_packet = 10M

M è Mb, G in Gb, K in Kb

Aumentare il limite di stringa per group_concat

`group_concat` è usato per concatenare valori non nulli in un `group` . La lunghezza massima della stringa risultante può essere impostata utilizzando l'opzione `group_concat_max_len` :

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

L'impostazione della variabile `GLOBAL` garantirà un cambiamento permanente, mentre l'impostazione della variabile `SESSION` imposterà il valore per la sessione corrente.

Configurazione InnoDB minima

Questa è una configurazione minima per i server MySQL che utilizzano le tabelle InnoDB. Usando InnoDB, la cache delle query non è richiesta. Recupera spazio su disco quando una tabella o un database è `DROP` ed. Se si utilizzano SSD, lo svuotamento è un'operazione ridondante (gli SDD non sono sequenziali).

```
default_storage_engine = InnoDB
query_cache_type = 0
innodb_file_per_table = 1
innodb_flush_neighbors = 0
```

Concorrenza

Assicurati di poter creare più di 4 thread predefiniti impostando `innodb_thread_concurrency` su infinito (0); ciò consente a InnoDB di decidere in base all'esecuzione ottimale.

```
innodb_thread_concurrency = 0
innodb_read_io_threads = 64
innodb_write_io_threads = 64
```

Utilizzo del disco rigido

Impostare la capacità (carico normale) e capacity_max (massimo assoluto) di IOPS per MySQL. Il valore predefinito di 200 va bene per HDD, ma in questi giorni, con SSD in grado di migliaia di IOPS, è probabile che si desideri regolare questo numero. Esistono molti test che è possibile eseguire per determinare IOPS. I valori sopra indicati dovrebbero essere quasi quel limite *se si sta utilizzando un server MySQL dedicato* . Se stai eseguendo altri servizi sulla stessa macchina, dovresti ripartire come appropriato.

```
innodb_io_capacity = 2500
innodb_io_capacity_max = 3000
```

Utilizzo della RAM

Imposta la RAM disponibile su MySQL. Sebbene la regola empirica sia del 70-80%, ciò dipende in realtà dall'opportunità o meno dell'istanza di MySQL e dalla quantità di RAM disponibile. Non *sprecare* RAM (cioè risorse) se hai molto a disposizione.

```
innodb_buffer_pool_size = 10G
```

Proteggi la crittografia MySQL

La crittografia predefinita `aes-128-ecb` utilizza la modalità Electronic Codebook (ECB), che non è sicura e non dovrebbe mai essere utilizzata. Invece, aggiungi quanto segue al tuo file di configurazione:

```
block_encryption_mode = aes-256-cbc
```

Leggi Configurazione e messa a punto online:
https://riptutorial.com/it/mysql/topic/3134/configurazione-e-messa-a-punto

Capitolo 16: Connessione con UTF-8 utilizzando vari linguaggi di programmazione.

Examples

Pitone

Prima o seconda riga nel codice sorgente (per avere letterali nel codice utf8-encoded):

```
# -*- coding: utf-8 -*-
```

Connessione:

```
db = MySQLdb.connect(host=DB_HOST, user=DB_USER, passwd=DB_PASS, db=DB_NAME,
        charset="utf8mb4", use_unicode=True)
```

Per le pagine Web, uno di questi:

```
<meta charset="utf-8" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

PHP

In php.ini (questo è il valore predefinito dopo PHP 5.6):

```
default_charset UTF-8
```

Quando si costruisce una pagina Web:

```
header('Content-type: text/plain; charset=UTF-8');
```

Quando ci si connette a MySQL:

```
(for mysql:)   Do not use the mysql_* API!
(for mysqli:)  $mysqli_obj->set_charset('utf8mb4');
(for PDO:)     $db = new PDO('dblib:host=host;dbname=db;charset=utf8', $user, $pwd);
```

Nel codice, non utilizzare alcuna routine di conversione.

Per l'inserimento dei dati,

```
<form accept-charset="UTF-8">
```

Per JSON, per evitare `\uxxxx` :

```
$t = json_encode($s, JSON_UNESCAPED_UNICODE);
```

Leggi Connessione con UTF-8 utilizzando vari linguaggi di programmazione. online: https://riptutorial.com/it/mysql/topic/7332/connessione-con-utf-8-utilizzando-vari-linguaggi-di-programmazione-

Capitolo 17: Conversione da MyISAM a InnoDB

Examples

Conversione di base

```
ALTER TABLE foo ENGINE=InnoDB;
```

Questo converte la tabella, ma non si cura delle differenze tra i motori. La maggior parte delle differenze non ha importanza, specialmente per i tavoli di piccole dimensioni. Ma per le tabelle più impegnative, dovrebbero essere prese in considerazione altre considerazioni. *Considerazioni di conversione*

Conversione di tutte le tabelle in un database

Per convertire facilmente tutte le tabelle in un unico database, utilizzare quanto segue:

```
SET @DB_NAME = DATABASE();

SELECT  CONCAT('ALTER TABLE `', table_name, '` ENGINE=InnoDB;') AS sql_statements
FROM    information_schema.tables
WHERE   table_schema = @DB_NAME
AND     `ENGINE` = 'MyISAM'
AND     `TABLE_TYPE` = 'BASE TABLE';
```

> **NOTA:** per funzionare, dovresti essere connesso al database affinché funzioni la funzione `DATABASE()` , altrimenti restituirà `NULL` . Questo si applica principalmente al client mysql standard fornito con il server in quanto consente di connettersi senza specificare un database.

Esegui questa istruzione SQL per recuperare tutte le tabelle `MyISAM` nel tuo database.

Infine, copia l'output ed esegui query SQL da esso.

Leggi Conversione da MyISAM a InnoDB online:
https://riptutorial.com/it/mysql/topic/3135/conversione-da-myisam-a-innodb

Capitolo 18: Crea nuovo utente

Osservazioni

Per visualizzare un elenco di utenti MySQL, utilizziamo il seguente comando:

```
SELECT User,Host FROM mysql.user;
```

Examples

Crea un utente MySQL

Per creare un nuovo utente, dobbiamo seguire semplici passaggi come di seguito:

Step 1: Accedi a `MySQL` come `root`

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

Passo 2: Vedremo il prompt dei comandi di mysql

```
mysql> CREATE USER 'my_new_user'@'localhost' IDENTIFIED BY 'test_password';
```

Qui, abbiamo creato con successo un nuovo utente, ma questo utente non avrà alcuna `permissions` , quindi per assegnare le `permissions` all'utente usa il seguente comando:

```
mysql> GRANT ALL PRIVILEGES ON my_db.* TO 'my_new_user'@'localhost' identified by
'my_password';
```

Specifica la password

L'utilizzo di base è:

```
mysql> CREATE USER 'my_new_user'@'localhost' IDENTIFIED BY 'test_password';
```

Tuttavia, per situazioni in cui non è consigliabile codificare la password in chiaro, è anche possibile specificare direttamente, utilizzando la direttiva `PASSWORD` , il valore hash restituito dalla funzione `PASSWORD()` :

```
mysql> select PASSWORD('test_password'); -- returns *4414E26EDED6D661B5386813EBBA95065DBC4728
mysql> CREATE USER 'my_new_user'@'localhost' IDENTIFIED BY PASSWORD
'*4414E26EDED6D661B5386813EBBA95065DBC4728';
```

Crea un nuovo utente e concedi tutti i privilegi allo schema

```
grant all privileges on schema_name.* to 'new_user_name'@'%' identified by 'newpassword';
```

Attenzione: questo può essere usato per creare un nuovo utente root

Rinominare l'utente

```
rename user 'user'@'%' to 'new_name`@'%';
```

Se crei un utente per errore, puoi cambiare il suo nome

Leggi Crea nuovo utente online: https://riptutorial.com/it/mysql/topic/3508/crea-nuovo-utente

Capitolo 19: Creazione della tabella

Sintassi

- CREATE TABLE table_name (column_name1 data_type (size), column_name2 data_type (size), column_name3 data_type (size), ...); // Creazione della tabella di base
- CREATE TABLE nome_tabella [IF NOT EXISTS] (column_name1 data_type (size), column_name2 data_type (size), column_name3 data_type (size), ...); // Controllo creazione tabelle esistente
- CREATE [TEMPORARY] TABLE nome_tabella [IF NOT EXISTS] (column_name1 data_type (size), column_name2 data_type (size), column_name3 data_type (size), ...); // Creazione di tabelle temporanee
- CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl; // Creazione di tabelle da SELECT

Osservazioni

L'istruzione `CREATE TABLE` dovrebbe terminare con una specifica `ENGINE` :

```
CREATE TABLE table_name ( column_definitions ) ENGINE=engine;
```

Alcune opzioni sono:

- `InnoDB` : (Impostazione predefinita dalla versione 5.5.5) È un motore sicuro per la transizione (compatibile ACID). Dispone di commit delle transazioni e rollback, funzionalità di ripristino in caso di arresto anomalo e blocco a livello di riga.
- `MyISAM` : (predefinito prima della versione 5.5.5) È un motore semplice veloce. Non supporta le transazioni, né le chiavi esterne, ma è utile per il data-warehousing.
- `Memory` : `Memory` tutti i dati nella RAM per operazioni estremamente veloci, ma la data della tabella andrà persa al riavvio del database.

Altre opzioni di motore qui .

Examples

Creazione di una tabella di base

L'istruzione `CREATE TABLE` viene utilizzata per creare una tabella in un database MySQL.

```
CREATE TABLE Person (
    `PersonID`      INTEGER NOT NULL PRIMARY KEY,
    `LastName`      VARCHAR(80),
    `FirstName`     VARCHAR(80),
    `Address`       TEXT,
```

```
    `City`          VARCHAR(100)
) Engine=InnoDB;
```

Ogni definizione di campo deve avere:

1. Nome campo: un nome campo valido. Assicurati di codificare i nomi in `-chars. Ciò garantisce che è possibile utilizzare ad esempio caratteri di spazio nel campo nomecampo.
2. Tipo di dati [Lunghezza]: se il campo è `CHAR` o `VARCHAR` , è obbligatorio specificare una lunghezza del campo.
3. Attributi `NULL` | `NOT NULL` : se viene specificato `NOT NULL` , qualsiasi tentativo di memorizzare un valore `NULL` in quel campo avrà esito negativo.
4. Vedi di più sui tipi di dati e i loro attributi qui .

`Engine=...` è un parametro facoltativo utilizzato per specificare il motore di archiviazione della tabella. Se non viene specificato alcun motore di archiviazione, la tabella verrà creata utilizzando il motore di memorizzazione della tabella predefinito del server (solitamente InnoDB o MyISAM).

Impostazione delle impostazioni predefinite

Inoltre, se ha senso, è possibile impostare un valore predefinito per ogni campo utilizzando `DEFAULT` :

```
CREATE TABLE Address (
    `AddressID`    INTEGER NOT NULL PRIMARY KEY,
    `Street`       VARCHAR(80),
    `City`         VARCHAR(80),
    `Country`      VARCHAR(80) DEFAULT "United States",
    `Active`       BOOLEAN DEFAULT 1,
) Engine=InnoDB;
```

Se durante gli inserimenti non è specificato `Street` , quel campo sarà `NULL` quando recuperato. Quando non viene specificato alcun `Country` momento dell'introduzione, per impostazione predefinita verrà impostato "Stati Uniti".

È possibile impostare valori predefiniti per tutti i tipi di colonna, ad eccezione dei campi `BLOB` , `TEXT` , `GEOMETRY` e `JSON` .

Creazione di tabelle con chiave primaria

```
CREATE TABLE Person (
    PersonID     INT UNSIGNED NOT NULL,
    LastName     VARCHAR(66) NOT NULL,
    FirstName    VARCHAR(66),
    Address      VARCHAR(255),
    City         VARCHAR(66),
    PRIMARY KEY (PersonID)
);
```

Una **chiave primaria** è un singolo `NOT NULL` o un identificatore a più colonne che identifica in modo

univoco una riga di una tabella. Viene creato un indice e, se non dichiarato esplicitamente come `NOT NULL` , MySQL lo dichiarerà in modo così silenzioso e implicito.

Una tabella può avere solo una `PRIMARY KEY` e ad ogni tabella si consiglia di averne una. InnoDB ne creerà automaticamente uno in sua assenza (come visto nella documentazione di MySQL) anche se questo è meno desiderabile.

Spesso, un `AUTO_INCREMENT INT` noto anche come "chiave surrogata", viene utilizzato per l'ottimizzazione dell'indice sottile e le relazioni con altre tabelle. Questo valore aumenterà (normalmente) di 1 ogni volta che viene aggiunto un nuovo record, a partire da un valore predefinito di 1.

Tuttavia, nonostante il suo nome, non è suo scopo garantire che i valori siano incrementali, ma semplicemente che siano sequenziali e unici.

Un valore `INT` autoincremento non verrà ripristinato al valore iniziale predefinito se tutte le righe della tabella vengono eliminate, a meno che la tabella non venga troncata utilizzando l'istruzione `TRUNCATE TABLE` .

Definizione di una colonna come chiave primaria (definizione in linea)

Se la chiave primaria è costituita da una singola colonna, la clausola `PRIMARY KEY` può essere posizionata in linea con la definizione della colonna:

```
CREATE TABLE Person (
    PersonID     INT UNSIGNED NOT NULL PRIMARY KEY,
    LastName     VARCHAR(66) NOT NULL,
    FirstName    VARCHAR(66),
    Address      VARCHAR(255),
    City         VARCHAR(66)
);
```

Questa forma del comando è più breve e più facile da leggere.

Definizione di una chiave primaria a più colonne

È anche possibile definire una chiave primaria comprendente più di una colonna. Questo potrebbe essere fatto ad esempio sulla tabella figlio di una relazione di chiave esterna. Una chiave primaria a più colonne viene definita elencando le colonne partecipanti in una clausola `PRIMARY KEY` separata. La sintassi inline non è consentita qui, poiché solo una colonna può essere dichiarata `PRIMARY KEY` linea. Per esempio:

```
CREATE TABLE invoice_line_items (
```

```
    LineNum      SMALLINT UNSIGNED NOT NULL,
    InvoiceNum   INT UNSIGNED NOT NULL,
    -- Other columns go here
    PRIMARY KEY (InvoiceNum, LineNum),
    FOREIGN KEY (InvoiceNum) REFERENCES -- references to an attribute of a table
);
```

Si noti che le colonne della chiave primaria *devono* essere specificate nell'ordine di ordinamento logico, che *potrebbe* essere diverso dall'ordine in cui sono state definite le colonne, come nell'esempio sopra riportato.

Gli indici più grandi richiedono più spazio su disco, memoria e I / O. Pertanto le chiavi dovrebbero essere il più piccole possibile (specialmente per quanto riguarda le chiavi composte). In InnoDB, ogni "indice secondario" include una copia delle colonne del `PRIMARY KEY` .

Creazione di tabelle con chiave esterna

```
CREATE TABLE Account (
    AccountID     INT UNSIGNED NOT NULL,
    AccountNo     INT UNSIGNED NOT NULL,
    PersonID    INT UNSIGNED,
    PRIMARY KEY (AccountID),
    FOREIGN KEY (PersonID) REFERENCES Person (PersonID)
) ENGINE=InnoDB;
```

Chiave esterna: una chiave esterna (`FK`) è una singola colonna o un composto composto da più colonne di una tabella di *riferimento* . Questo `FK` è confermato per esistere nella tabella di *riferimento* . Si consiglia vivamente che la chiave della tabella di *riferimento che* conferma l' `FK` sia una chiave primaria, ma non viene applicata. È usato come ricerca rapida nel *riferimento* dove non ha bisogno di essere unico, e in effetti può essere l'indice più a sinistra lì.

Le relazioni con le chiavi esterne coinvolgono una tabella padre che contiene i valori dei dati centrali e una tabella figlio con valori identici che risalgono al suo genitore. La clausola FOREIGN KEY è specificata nella tabella figlio. Le tabelle padre e figlio devono utilizzare lo stesso motore di archiviazione. Non devono essere tavoli TEMPORANEA .

Le colonne corrispondenti nella chiave esterna e la chiave di riferimento devono avere tipi di dati simili. La dimensione e il segno dei tipi interi devono essere uguali. La lunghezza dei tipi di stringa non deve essere la stessa. Per le colonne di stringa non binarie (carattere), il set di caratteri e le regole di confronto devono essere uguali.

Nota: i vincoli di chiave esterna sono supportati dal motore di archiviazione InnoDB (non da MyISAM o MEMORY). Le configurazioni DB che utilizzano altri motori accettano questa istruzione `CREATE TABLE` ma non rispetteranno i vincoli di chiave esterna. (Anche se le versioni più recenti di MySQL sono predefinite in `InnoDB` , ma è buona prassi essere esplicite.)

Clonazione di una tabella esistente

Una tabella può essere replicata come segue:

```
CREATE TABLE ClonedPersons LIKE Persons;
```

La nuova tabella avrà esattamente la stessa struttura della tabella originale, inclusi gli indici e gli attributi di colonna.

Oltre a creare manualmente una tabella, è anche possibile creare una tabella selezionando i dati da un'altra tabella:

```
CREATE TABLE ClonedPersons SELECT * FROM Persons;
```

È possibile utilizzare una qualsiasi delle normali funzionalità di un'istruzione `SELECT` per modificare i dati man mano che si procede:

```
CREATE TABLE ModifiedPersons
SELECT PersonID, FirstName + LastName AS FullName FROM Persons
WHERE LastName IS NOT NULL;
```

Le chiavi e gli indici primari non verranno conservati durante la creazione di tabelle da `SELECT` . È necessario ridichiarli:

```
CREATE TABLE ModifiedPersons (PRIMARY KEY (PersonID))
SELECT PersonID, FirstName + LastName AS FullName FROM Persons
WHERE LastName IS NOT NULL;
```

CREA TABELLA DA SELEZIONA

È possibile creare una tabella da un'altra aggiungendo un'istruzione `SELECT` alla fine `CREATE TABLE` :

```
CREATE TABLE stack (
    id_user INT,
    username VARCHAR(30),
    password VARCHAR(30)
);
```

Crea una tabella nello stesso database:

```
-- create a table from another table in the same database with all attributes
CREATE TABLE stack2 AS SELECT * FROM stack;

-- create a table from another table in the same database with some attributes
CREATE TABLE stack3 AS SELECT username, password FROM stack;
```

Crea tabelle da diversi database:

```
-- create a table from another table from another database with all attributes
CREATE TABLE stack2 AS SELECT * FROM second_db.stack;

-- create a table from another table from another database with some attributes
CREATE TABLE stack3 AS SELECT username, password FROM second_db.stack;
```

NB

Per creare una tabella uguale a un'altra tabella esistente in un altro database, è necessario specificare il nome del database in questo modo:

```
FROM NAME_DATABASE.name_table
```

Mostra struttura tabella

Se si desidera visualizzare le informazioni sullo schema della tabella, è possibile utilizzare uno dei seguenti:

```
SHOW CREATE TABLE child; -- Option 1

CREATE TABLE `child` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fullName` varchar(100) NOT NULL,
  `myParent` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `mommy_daddy` (`myParent`),
  CONSTRAINT `mommy_daddy` FOREIGN KEY (`myParent`) REFERENCES `parent` (`id`)
      ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Se usato dallo strumento della riga di comando mysql, questo è meno dettagliato:

```
SHOW CREATE TABLE child \G
```

Un modo meno descrittivo di mostrare la struttura della tabella:

```
mysql> CREATE TABLE Tab1(id int, name varchar(30));
Query OK, 0 rows affected (0.03 sec)

mysql> DESCRIBE Tab1; -- Option 2

+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

> Sia **DESCRIBE** che **DESC** danno lo stesso risultato.

Per vedere `DESCRIBE` eseguito su tutte le tabelle di un database contemporaneamente, vedere questo esempio .

Tabella Crea con colonna TimeStamp per mostrare l'ultimo aggiornamento

La colonna TIMESTAMP mostrerà quando la riga è stata aggiornata l'ultima volta.

```
CREATE TABLE `TestLastUpdate` (
```

```
    `ID` INT NULL,
    `Name` VARCHAR(50) NULL,
    `Address` VARCHAR(50) NULL,
    `LastUpdate` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
COMMENT='Last Update'
;
```

Leggi Creazione della tabella online: https://riptutorial.com/it/mysql/topic/795/creazione-della-tabella

Capitolo 20: Creazione di database

Sintassi

- CREATE {DATABASE | SCHEMA} [SE NON ESISTE] db_name [create_specification] /// Per creare un database
- DROP {DATABASE | SCHEMA} [IF EXISTS] db_name /// Per eliminare il database

Parametri

Parametro	Dettagli
CREA DATABASE	Crea un database con il nome specificato
CREA SCHEMA	Questo è un sinonimo per `CREATE DATABASE`
SE NON ESISTE	Utilizzato per evitare errori di esecuzione, se il database specificato esiste già
create_specification	`create_specification` opzioni `create_specification` specificano le caratteristiche del database come `CHARACTER SET` e `COLLATE` (confronto dei database)

Examples

Crea database, utenti e concessioni

Crea un DATABASE. Si noti che la parola abbreviata SCHEMA può essere utilizzata come sinonimo.

```
CREATE DATABASE Baseball; -- creates a database named Baseball
```

Se il database esiste già, viene restituito l'errore 1007. Per aggirare questo errore, prova:

```
CREATE DATABASE IF NOT EXISTS Baseball;
```

Allo stesso modo,

```
DROP DATABASE IF EXISTS Baseball; -- Drops a database if it exists, avoids Error 1008
DROP DATABASE xyz; -- If xyz does not exist, ERROR 1008 will occur
```

A causa delle precedenti possibilità di errore, le istruzioni DDL vengono spesso utilizzate con `IF EXISTS` .

Si può creare un database con un SET CARATTERE e regole di confronto predefiniti. Per esempio:

```
CREATE DATABASE Baseball CHARACTER SET utf8 COLLATE utf8_general_ci;

SHOW CREATE DATABASE Baseball;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| Baseball | CREATE DATABASE `Baseball` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
```

Vedi i tuoi attuali database:

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| ajax_stuff         |
| Baseball           |
+--------------------+
```

Imposta il database attualmente attivo e vedi alcune informazioni:

```
USE Baseball; -- set it as the current database
SELECT @@character_set_database as cset,@@collation_database as col;
+------+-----------------+
| cset | col             |
+------+-----------------+
| utf8 | utf8_general_ci |
+------+-----------------+
```

Quanto sopra mostra il SET CARATTERE e le regole di confronto predefiniti per il database.

Crea un utente:

```
CREATE USER 'John123'@'%' IDENTIFIED BY 'OpenSesame';
```

Quanto sopra crea un utente John123, in grado di connettersi con qualsiasi nome host a causa della `%` jolly. La password per l'utente è impostata su 'OpenSesame', che è hash.

E creane un altro:

```
CREATE USER 'John456'@'%' IDENTIFIED BY 'somePassword';
```

Mostra che gli utenti sono stati creati esaminando lo speciale database `mysql` :

```
SELECT user,host,password from mysql.user where user in ('John123','John456');
+---------+------+-------------------------------------------+
| user    | host | password                                  |
+---------+------+-------------------------------------------+
| John123 | %    | *E6531C342ED87 ...................        |
```

```
| John456 | %    | *B04E11FAAAE9A ...................       |
+---------+------+-------------------------------------------+
```

Si noti che a questo punto, gli utenti sono stati creati, ma senza alcuna autorizzazione per utilizzare il database Baseball.

Lavora con permessi per utenti e database. Concedere i diritti all'utente John123 per avere i privilegi completi sul database Baseball e solo i diritti SELECT per l'altro utente:

```
GRANT ALL ON Baseball.* TO 'John123'@'%';
GRANT SELECT ON Baseball.* TO 'John456'@'%';
```

Verifica quanto sopra:

```
SHOW GRANTS FOR 'John123'@'%';
+-----------------------------------------------------------------------------------------------
-----------+
| Grants for John123@%
|
+-----------------------------------------------------------------------------------------------
-----------+
| GRANT USAGE ON *.* TO 'John123'@'%' IDENTIFIED BY PASSWORD '*E6531C342ED87
.....................        |
| GRANT ALL PRIVILEGES ON `baseball`.* TO 'John123'@'%'
|
+-----------------------------------------------------------------------------------------------
-----------+

SHOW GRANTS FOR 'John456'@'%';
+-----------------------------------------------------------------------------------------------
-----------+
| Grants for John456@%
|
+-----------------------------------------------------------------------------------------------
-----------+
| GRANT USAGE ON *.* TO 'John456'@'%' IDENTIFIED BY PASSWORD '*B04E11FAAAE9A
.....................        |
| GRANT SELECT ON `baseball`.* TO 'John456'@'%'
|
+-----------------------------------------------------------------------------------------------
-----------+
```

Notare che l' `GRANT USAGE` che si vedrà sempre significa semplicemente che l'utente può accedere. Questo è tutto ciò che significa.

MyDatabase

È *necessario* creare il proprio database e non utilizzare la scrittura in nessuno dei database esistenti. Questa è probabilmente una delle prime cose da fare dopo essere stata connessa la prima volta.

```
CREATE DATABASE my_db;
USE my_db;
CREATE TABLE some_table;
```

```
INSERT INTO some_table ...;
```

È possibile fare riferimento alla tabella qualificandosi con il nome del database: `my_db.some_table` .

Database di sistema

I seguenti database esistono per l'uso di MySQL. Puoi leggerli (`SELECT`), ma non devi scrivere (`INSERT` / `UPDATE` / `DELETE`) le tabelle al loro interno. (Ci sono alcune eccezioni.)

- `mysql` - repository per informazioni `GRANT` e alcune altre cose.
- `information_schema` : le tabelle qui sono 'virtuali' nel senso che sono effettivamente manifestate da strutture in memoria. Il loro contenuto include lo schema per tutte le tabelle.
- `performance_schema` - ?? [si prega di accettare, quindi modificare]
- altri?? (per MariaDB, Galera, TokuDB, ecc.)

Creazione e selezione di un database

Se l'amministratore crea il tuo database al momento della configurazione delle autorizzazioni, puoi iniziare a usarlo. Altrimenti, devi crearlo da solo:

```
mysql> CREATE DATABASE menagerie;
```

Sotto Unix, i nomi dei database fanno distinzione tra maiuscole e minuscole (a differenza delle parole chiave SQL), quindi è necessario fare sempre riferimento al proprio database come serraglio, non come Menagerie, MENAGERIE o qualche altra variante. Questo vale anche per i nomi delle tabelle. (In Windows, questa limitazione non si applica, anche se è necessario fare riferimento a database e tabelle utilizzando la stessa lettera maiuscola in una determinata query. Tuttavia, per una serie di motivi, la best practice consigliata è sempre quella di utilizzare la stessa lettera maiuscola utilizzata quando il database è stato creato.)

La creazione di un database non lo seleziona per l'uso; devi farlo esplicitamente. Per rendere il serraglio il database attuale, utilizzare questa dichiarazione:

```
mysql> USE menagerie
Database changed
```

Il tuo database deve essere creato una sola volta, ma devi selezionarlo per l'uso ogni volta che inizi una sessione mysql. Puoi farlo inviando una dichiarazione USE come mostrato nell'esempio. In alternativa, puoi selezionare il database sulla riga di comando quando invochi mysql. Basta specificare il suo nome dopo ogni parametro di connessione che potrebbe essere necessario fornire. Per esempio:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Leggi Creazione di database online: https://riptutorial.com/it/mysql/topic/600/creazione-di-database

Capitolo 21: Drop Table

Sintassi

- DROP TABLE nome_tabella;
- DROP TABLE IF EXISTS nome_tabella; - per evitare fastidiosi errori nello script automatico
- DROP TABLE t1, t2, t3; - DROP più tabelle
- DROP TEMPORARY TABLE t; - DROP una tabella da CREATE TABELLA TEMPORANEA ...

Parametri

parametri	Dettagli
TEMPORANEO	Opzionale. Specifica che solo le tabelle temporanee devono essere eliminate dall'istruzione DROP TABLE.
SE ESISTE	Opzionale. Se specificato, l'istruzione DROP TABLE non genera un errore se una delle tabelle non esiste.

Examples

Drop Table

Drop Table è usato per cancellare la tabella dal database.

Creazione della tabella:

Creazione di una tabella denominata tbl e quindi eliminazione della tabella creata

```
CREATE TABLE tbl(
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(100) NOT NULL,
    author VARCHAR(40) NOT NULL,
    submission_date DATE,
    PRIMARY KEY (id)
);
```

Tabella di lancio:

```
DROP TABLE tbl;
```

> **NOTARE CHE**
>
> La tabella di eliminazione eliminerà completamente la tabella dal database e tutte le

relative informazioni e non verrà ripristinata.

Eliminare tabelle dal database

DROP TABLE Database.table_name

Leggi Drop Table online: https://riptutorial.com/it/mysql/topic/4123/drop-table

Capitolo 22: ELIMINA

Sintassi

- ELIMINA [LOW_PRIORITY] [QUICK] [IGNORA] FROM table [WHERE conditions] [ORDER BY expression [ASC | DESC]] [LIMIT number_rows]; /// Sintassi per eliminare le righe da una singola tabella

Parametri

Parametro	Dettagli
BASSA PRIORITÀ	Se viene fornito `LOW_PRIORITY` , l'eliminazione verrà ritardata fino a quando non ci saranno processi in lettura dalla tabella
IGNORARE	Se viene fornito `IGNORE` , tutti gli errori incontrati durante l'eliminazione vengono ignorati
tavolo	La tabella da cui si intende eliminare i record
Dove condizioni	Le condizioni che devono essere soddisfatte per i record da eliminare. Se non vengono fornite condizioni, tutti i record dalla tabella verranno eliminati
ORDINA PER espressione	Se viene fornito `ORDER BY` , i record verranno eliminati nell'ordine indicato
LIMITE	Controlla il numero massimo di record da eliminare dalla tabella. Dato `number_rows` sarà cancellato.

Examples

Elimina con clausola Where

```
DELETE FROM `table_name` WHERE `field_one` = 'value_one'
```

Questo cancellerà tutte le righe dalla tabella in cui il contenuto di `field_one` per quella riga corrisponde a 'value_one'

La clausola `WHERE` funziona allo stesso modo di una selezione, quindi è possibile utilizzare cose come `>` , `<` , `<>` o `LIKE` .

Avviso: è necessario utilizzare le clausole condizionali (WHERE, LIKE) nella query di eliminazione. Se non si usano clausole condizionali, tutti i dati da quella tabella verranno eliminati.

Elimina tutte le righe da una tabella

```
DELETE FROM table_name ;
```

Questo cancellerà tutto, tutte le righe dalla tabella. È l'esempio più semplice della sintassi. Mostra anche che le istruzioni `DELETE` dovrebbero essere utilizzate con estrema attenzione in quanto potrebbero svuotare una tabella, se la clausola `WHERE` viene omessa.

LIMITAZIONE delle eliminazioni

```
DELETE FROM `table_name` WHERE `field_one` = 'value_one' LIMIT 1
```

Funziona allo stesso modo dell'esempio "Elimina con clausola Where", ma interromperà la cancellazione una volta rimosso il numero limitato di righe.

Se stai limitando le righe per la cancellazione in questo modo, tieni presente che eliminerà la prima riga che corrisponde ai criteri. Potrebbe non essere quello che ci si aspetterebbe, in quanto i risultati possono tornare non ordinati se non sono esplicitamente ordinati.

Eliminazioni di più tabelle

L'istruzione `DELETE` di MySQL può utilizzare il costrutto `JOIN` , consentendo anche di specificare da quali tabelle eliminare. Questo è utile per evitare query annidate. Dato lo schema:

```
create table people
(    id int primary key,
    name varchar(100) not null,
    gender char(1) not null
);
insert people (id,name,gender) values
(1,'Kathy','f'),(2,'John','m'),(3,'Paul','m'),(4,'Kim','f');

create table pets
(    id int auto_increment primary key,
    ownerId int not null,
    name varchar(100) not null,
    color varchar(100) not null
);
insert pets(ownerId,name,color) values
(1,'Rover','beige'),(2,'Bubbles','purple'),(3,'Spot','black and white'),
(1,'Rover2','white');
```

id	nome	Genere
1	Kathy	f
2	John	m
3	Paolo	m
4	Kim	f

id	ownerid	nome	colore
1	1	vagabondo	beige
2	2	bolle	viola
4	1	Rover2	bianca

Se vogliamo rimuovere gli animali domestici di Paul, la dichiarazione

```
DELETE p2
FROM pets p2
WHERE p2.ownerId in (
    SELECT p1.id
    FROM people p1
    WHERE p1.name = 'Paul');
```

può essere riscritto come:

```
DELETE p2    -- remove only rows from pets
FROM people p1
JOIN pets p2
ON p2.ownerId = p1.id
WHERE p1.name = 'Paul';
```

1 riga cancellata
Spot viene eliminato dagli animali domestici

`p1` e `p2` sono alias per i nomi delle tabelle, particolarmente utili per nomi di tabelle lunghe e facilità di lettura.

Per rimuovere sia la persona che l'animale domestico:

```
DELETE p1, p2     -- remove rows from both tables
FROM people p1
JOIN pets p2
ON p2.ownerId = p1.id
WHERE p1.name = 'Paul';
```

2 righe cancellate
Spot viene eliminato dagli animali domestici
Paul viene cancellato dalle persone

chiavi esterne

Quando l'istruzione DELETE esegue il richiamo di tabelle con una chiave esterna, l'ottimizzatore può elaborare le tabelle in un ordine che non segue la relazione. Aggiungendo ad esempio una chiave esterna alla definizione di `pets`

```
ALTER TABLE pets ADD CONSTRAINT `fk_pets_2_people` FOREIGN KEY (ownerId) references people(id)
```

```
ON DELETE CASCADE;
```

il motore può provare a eliminare le voci dalle `people` prima degli `pets` , provocando il seguente errore:

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
(`test`.`pets`, CONSTRAINT `pets_ibfk_1` FOREIGN KEY (`ownerId`) REFERENCES `people` (`id`))
```

La soluzione in questo caso è eliminare la riga dalle `people` e fare affidamento sulle funzionalità `ON DELETE InnoDB` per propagare la cancellazione:

```
DELETE FROM people
WHERE name = 'Paul';
```

2 righe cancellate
Paul viene cancellato dalle persone
Spot viene eliminato su cascata da Animali domestici

Un'altra soluzione consiste nel disabilitare temporaneamente il controllo sulle chiavi esterne:

```
SET foreign_key_checks = 0;
DELETE p1, p2 FROM people p1 JOIN pets p2 ON p2.ownerId = p1.id WHERE p1.name = 'Paul';
SET foreign_key_checks = 1;
```

Eliminazione di base

```
DELETE FROM `myTable` WHERE `someColumn` = 'something'
```

La clausola `WHERE` è facoltativa, ma senza di essa tutte le righe vengono eliminate.

DELETE vs TRUNCATE

```
TRUNCATE tableName;
```

Questo cancellerà tutti i dati e resetterà l'indice `AUTO_INCREMENT` . È molto più veloce di `DELETE FROM tableName` su un enorme set di dati. Può essere molto utile durante lo sviluppo / test.

Quando si ***tronca*** una tabella, il server SQL non elimina i dati, elimina la tabella e la ricrea, in modo da deallocare le pagine in modo che vi sia la possibilità di ripristinare i dati troncati prima delle pagine sovrascritte. (Lo spazio non può essere recuperato immediatamente per `innodb_file_per_table=OFF` .)

DELETE multi tabella

MySQL consente di specificare da quale tabella devono essere cancellate le righe corrispondenti

```
    -- remove only the employees
```

```
DELETE e
FROM Employees e JOIN Department d ON e.department_id = d.department_id
WHERE d.name = 'Sales'
```

```
-- remove employees and department
DELETE e, d
FROM Employees e JOIN Department d ON e.department_id = d.department_id
WHERE d.name = 'Sales'
```

```
-- remove from all tables (in this case same as previous)
DELETE
FROM Employees e JOIN Department d ON e.department_id = d.department_id
WHERE d.name = 'Sales'
```

Leggi ELIMINA online: https://riptutorial.com/it/mysql/topic/1487/elimina

Capitolo 23: ENUM

Examples

Perché ENUM?

ENUM fornisce un modo per fornire un attributo per una riga. Gli attributi con un numero limitato di opzioni non numeriche funzionano meglio. Esempi:

```
reply ENUM('yes', 'no')
gender ENUM('male', 'female', 'other', 'decline-to-state')
```

I valori sono stringhe:

```
INSERT ... VALUES ('yes', 'female')
SELECT ... --> yes female
```

TINYINT come alternativa

Diciamo che abbiamo

```
type ENUM('fish','mammal','bird')
```

Un'alternativa è

```
type TINYINT UNSIGNED
```

più

```
CREATE TABLE AnimalTypes (
    type TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL  COMMENT "('fish','mammal','bird')",
    PRIMARY KEY(type),
    INDEX(name)
) ENGINE=InnoDB
```

che è molto simile a una tabella molti-a-molti.

Confronto, e se migliore o peggiore di ENUM:

- (peggio) INSERISCI: è necessario cercare il `type`
- (peggio) SELEZIONA: è necessario unire per ottenere la stringa (ENUM ti dà la stringa senza sforzo)
- (migliore) Aggiunta di nuovi tipi: basta inserire in questa tabella. Con ENUM, è necessario eseguire una ALTER TABLE.
- (stessa) Ciascuna tecnica (per un massimo di 255 valori) richiede solo 1 byte.

- (misto) C'è anche un problema di integrità dei dati: `TINYINT` ammetterà valori non validi; mentre `ENUM` li imposta su un valore di stringa vuota speciale (a meno che non sia abilitata la modalità SQL rigorosa, nel qual caso vengono rifiutati). È possibile ottenere una migliore integrità dei dati con `TINYINT` rendendola una chiave esterna in una tabella di ricerca: con query / join appropriate, ma il costo per raggiungere l'altra tabella è ancora limitato. (Le `FOREIGN KEYs` non sono gratuite.)

VARCHAR come alternativa

Diciamo che abbiamo

```
type ENUM('fish','mammal','bird')
```

Un'alternativa è

```
type VARCHAR(20)  COMENT "fish, bird, etc"
```

Questo è abbastanza aperto in quanto i nuovi tipi sono banalmente aggiunti.

Confronto, e se migliore o peggiore di ENUM:

- (stesso) INSERT: fornire semplicemente la stringa
- (peggio?) In INSERT un errore di battitura passerà inosservato
- (stesso) SELECT: viene restituita la stringa effettiva
- (peggio) Viene consumato molto più spazio

Aggiunta di una nuova opzione

```
ALTER TABLE tbl MODIFY COLUMN type ENUM('fish','mammal','bird','insect');
```

Gli appunti

- Come in tutti i casi di MODIFY COLUMN, devi includere `NOT NULL` e qualsiasi altro qualificatore originariamente esistente, altrimenti andranno persi.
- *Se* si aggiunge alla *fine* dell'elenco *e* l'elenco è inferiore a 256 elementi, `ALTER` viene eseguito semplicemente cambiando lo schema. Questo non ci sarà una copia da tavolo lunga. (Le vecchie versioni di MySQL non avevano questa ottimizzazione.)

NULL vs NOT NULL

Esempi di ciò che accade quando NULL e 'bad-value' sono memorizzati in colonne nullable e not nullable. Mostra anche l'utilizzo del cast in numerico tramite `+0` .

```
CREATE TABLE enum (
    e     ENUM('yes', 'no')   NOT NULL,
    enull ENUM('x', 'y', 'z')     NULL
        );
INSERT INTO enum (e, enull)
```

```
    VALUES
        ('yes', 'x'),
        ('no',  'y'),
        (NULL,  NULL),
        ('bad-value', 'bad-value');
Query OK, 4 rows affected, 3 warnings (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 3

mysql>SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1048 | Column 'e' cannot be null                 |
| Warning | 1265 | Data truncated for column 'e' at row 4    |
| Warning | 1265 | Data truncated for column 'enull' at row 4 |
+---------+------+-------------------------------------------+
3 rows in set (0.00 sec)
```

Cosa c'è nella tabella dopo questi inserti. Questo utilizza "+0" per eseguire il cast a valori numerici per vedere cosa viene memorizzato.

```
mysql>SELECT e, e+0 FROM enum;
+-----+-----+
| e   | e+0 |
+-----+-----+
| yes |   1 |
| no  |   2 |
|     |   0 |  -- NULL
|     |   0 |  -- 'bad-value'
+-----+-----+
4 rows in set (0.00 sec)

mysql>SELECT enull, enull+0 FROM enum;
+-------+---------+
| enull | enull+0 |
+-------+---------+
| x     |       1 |
| y     |       2 |
| NULL  |    NULL |
|       |       0 |  -- 'bad-value'
+-------+---------+
4 rows in set (0.00 sec)
```

Leggi ENUM online: https://riptutorial.com/it/mysql/topic/4425/enum

Capitolo 24: Errore 1055: ONLY_FULL_GROUP_BY: qualcosa non è nella clausola GROUP BY ...

introduzione

Di recente, le nuove versioni dei server MySQL hanno iniziato a generare errori 1055 per le query che prima funzionavano. Questo argomento spiega quegli errori. Il team di MySQL ha lavorato per ritirare l'estensione non standard a `GROUP BY` , o almeno per rendere più difficile per gli sviluppatori di query di scrittura da masterizzare.

Osservazioni

Per molto tempo, MySQL ha contenuto una nota estensione non standard di `GROUP BY` , che consente comportamenti stravaganti in nome dell'efficienza. Questa estensione ha permesso a innumerevoli sviluppatori in tutto il mondo di utilizzare `GROUP BY` nel codice di produzione senza capire completamente cosa stessero facendo.

In particolare, è una cattiva idea utilizzare `SELECT *` in una query `GROUP BY` , perché una clausola `GROUP BY` standard richiede l'enumerazione delle colonne. Sfortunatamente, molti sviluppatori lo hanno fatto.

Leggi questo. https://dev.mysql.com/doc/refman/5.7/en/group-by-handling.html

Il team di MySQL ha cercato di risolvere questa disfunzione senza incasinare il codice di produzione. Hanno aggiunto un flag `sql_mode` in 5.7.5 denominato `ONLY_FULL_GROUP_BY` per forzare il comportamento standard. In una recente versione, hanno attivato quel flag per impostazione predefinita. Quando hai aggiornato MySQL locale alla 5.7.14, la flag è stata attivata e il tuo codice di produzione, dipendente dalla vecchia estensione, ha smesso di funzionare.

Se hai recentemente iniziato a ottenere errori 1055, quali sono le tue scelte?

1. correggi le query SQL offende o chiedi ai loro autori di farlo.
2. ripristinare la versione di MySQL compatibile immediatamente con il software applicativo utilizzato.
3. cambia la `sql_mode` del tuo server per sbarazzarti della nuova modalità `ONLY_FULL_GROUP_BY` .

È possibile modificare la modalità eseguendo un comando `SET` .

```
SET  sql_mode =
'STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENG
```

dovrebbe fare il trucco se lo fai subito dopo che l'applicazione si connette a MySQL.

Oppure, puoi trovare il file init nella tua installazione MySQL , localizzare la riga `sql_mode=` e cambiarlo per omettere `ONLY_FULL_GROUP_BY` e riavviare il tuo server.

Examples

Utilizzo e abuso di GROUP BY

```
SELECT item.item_id, item.name,     /* not SQL-92 */
       COUNT(*) number_of_uses
 FROM item
 JOIN uses ON item.item_id, uses.item_id
GROUP BY item.item_id
```

mostrerà le righe in una tabella chiamata `item` e mostrerà il conteggio delle righe correlate in una tabella denominata `uses` . Funziona bene, ma sfortunatamente non è SQL-92 standard.

Perchè no? poiché la clausola `SELECT` (e la clausola `ORDER BY`) nelle query `GROUP BY` deve contenere colonne

1. menzionato nella clausola `GROUP BY` , o
2. funzioni di aggregazione come `COUNT()` , `MIN()` e simili.

La clausola `SELECT` questo esempio menziona `item.name` , una colonna che non soddisfa uno di questi criteri. MySQL 5.6 e precedenti rifiuteranno questa query se la modalità SQL contiene `ONLY_FULL_GROUP_BY` .

Questa query di esempio può essere eseguita per conformarsi allo standard SQL-92 modificando la clausola `GROUP BY` , in questo modo.

```
SELECT item.item_id, item.name,
       COUNT(*) number_of_uses
  FROM item
  JOIN uses ON item.item_id, uses.item_id
 GROUP BY item.item_id, item.name
```

Lo standard SQL-99 successivo consente a un'istruzione `SELECT` di omettere le colonne non aggregate dalla chiave di gruppo se il DBMS può dimostrare una dipendenza funzionale tra di esse e le colonne chiave di gruppo. Poiché `item.name` dipende dal punto di `item.item_id` funzionale su `item.item_id` , l'esempio iniziale è valido SQL-99. MySQL ha ottenuto un test di dipendenza funzionale nella versione 5.7. L'esempio originale funziona con `ONLY_FULL_GROUP_BY` .

Utilizzo improprio di GROUP BY per restituire risultati imprevedibili: la legge di Murphy

```
SELECT item.item_id, uses.category,   /* nonstandard */
       COUNT(*) number_of_uses
  FROM item
  JOIN uses ON item.item_id, uses.item_id
 GROUP BY item.item_id
```

mostrerà le righe in una tabella chiamata item e mostrerà il conteggio delle righe correlate in una tabella denominata uses. `uses.category` anche il valore di una colonna chiamata `uses.category` .

Questa query funziona in MySQL (prima che `ONLY_FULL_GROUP_BY` flag `ONLY_FULL_GROUP_BY`). Usa l'estensione non standard di MySQL a `GROUP BY` .

Ma la query ha un problema: se più righe nella tabella `uses` corrispondono alla condizione `ON` nella clausola `JOIN` , MySQL restituisce la colonna della `category` da una sola di quelle righe. Quale fila? Lo scrittore della query e l'utente dell'applicazione non lo sanno in anticipo. Formalmente parlando, è *imprevedibile* : MySQL può restituire qualsiasi valore desiderato.

Imprevedibile è come *casuale,* con una differenza significativa. Ci si potrebbe aspettare che una scelta *casuale* cambi di volta in volta. Pertanto, se una scelta fosse casuale, potreste rilevarla durante il debug o il test. Il risultato *imprevedibile* è peggiore: MySQL restituisce lo stesso risultato ogni volta che si utilizza la query, *finché non lo fa.* A volte è una nuova versione del server MySQL che causa un risultato diverso. A volte è una tabella in crescita a causare il problema. Cosa può andare storto, andrà male, e quando non te lo aspetti. Si chiama Murphy's Law .

Il team di MySQL ha lavorato per rendere più difficile per gli sviluppatori commettere questo errore. Le versioni più recenti di MySQL nella sequenza 5.7 hanno un flag `sql_mode` chiamato `ONLY_FULL_GROUP_BY` . Quando viene impostato questo flag, il server MySQL restituisce l'errore 1055 e si rifiuta di eseguire questo tipo di query.

Utilizzo errato di GROUP BY con SELECT * e come risolverlo.

A volte una query appare come questa, con un `*` nella clausola `SELECT` .

```
SELECT item.*,     /* nonstandard */
       COUNT(*) number_of_uses
 FROM item
 JOIN uses ON item.item_id, uses.item_id
GROUP BY item.item_id
```

Tale query deve essere sottoposta a refactoring per essere conforme allo standard `ONLY_FULL_GROUP_BY` .

Per fare ciò, abbiamo bisogno di una subquery che usi correttamente `GROUP BY` per restituire il valore `number_of_uses` per ogni `item_id` . Questa sottoquery è breve e dolce, perché ha solo bisogno di guardare la tabella degli `uses` .

```
                            SELECT item_id, COUNT(*) number_of_uses
                              FROM  uses
                             GROUP BY item_id
```

Quindi, possiamo unirmi a quella sottoquery con la tabella degli `item` .

```
SELECT item.*, usecount.number_of_uses
  FROM item
  JOIN (
                            SELECT item_id, COUNT(*) number_of_uses
```

```
                              FROM  uses
                             GROUP BY item_id
        ) usecount ON item.item_id = usecount.item_id
```

Ciò consente alla clausola `GROUP BY` di essere semplice e corretta e ci consente anche di usare l'identificatore `*` .

Nota: tuttavia, gli sviluppatori saggi evitano di utilizzare l'identificatore `*` in ogni caso. Di solito è meglio elencare le colonne che desideri in una query.

ANY_VALUE ()

```
SELECT item.item_id, ANY_VALUE(uses.tag) tag,
       COUNT(*) number_of_uses
 FROM item
 JOIN uses ON item.item_id, uses.item_id
GROUP BY item.item_id
```

mostra le righe in una tabella chiamata `item` , il conteggio delle righe correlate e uno dei valori nella tabella correlata chiamata `uses` .

Puoi pensare a questa funzione `ANY_VALUE()` come a uno strano tipo di funzione aggregata. Invece di restituire un conteggio, somma o massimo, istruisce il server MySQL a scegliere, arbitrariamente, un valore dal gruppo in questione. È un modo per aggirare l'errore 1055.

`ANY_VALUE()` attenzione quando usi `ANY_VALUE()` nelle query nelle applicazioni di produzione.

Dovrebbe davvero essere chiamato `SURPRISE_ME()` . Restituisce il valore di alcune righe nel gruppo GROUP BY. Quale riga restituisce è indeterminata. Ciò significa che dipende interamente dal server MySQL. Formalmente, restituisce un valore imprevedibile.

Il server non sceglie un valore casuale, è peggio di quello. Restituisce lo stesso valore ogni volta che si esegue la query, finché non lo fa. Può cambiare o meno quando una tabella cresce o si riduce, o quando il server ha più o meno RAM, o quando la versione del server cambia, o quando Marte è in retrogrado (qualunque cosa ciò significhi), o per nessuna ragione.

Sei stato avvertito.

Leggi Errore 1055: ONLY_FULL_GROUP_BY: qualcosa non è nella clausola GROUP BY ... online: https://riptutorial.com/it/mysql/topic/8245/errore-1055--only-full-group-by--qualcosa-non-e-nella-clausola-group-by----

Capitolo 25: Espressioni regolari

introduzione

Un'espressione regolare è un modo efficace per specificare un modello per una ricerca complessa.

Examples

REGEXP / RLIKE

L' `REGEXP` (o il suo sinonimo, `RLIKE`) consente la corrispondenza del modello in base alle espressioni regolari.

Considera la seguente tabella dei `employee` :

```
+-------------+-------------+-------------+--------------+----------+
| EMPLOYEE_ID | FIRST_NAME  | LAST_NAME   | PHONE_NUMBER | SALARY   |
+-------------+-------------+-------------+--------------+----------+
|         100 | Steven      | King        | 515.123.4567 | 24000.00 |
|         101 | Neena       | Kochhar     | 515.123.4568 | 17000.00 |
|         102 | Lex         | De Haan     | 515.123.4569 | 17000.00 |
|         103 | Alexander   | Hunold      | 590.423.4567 |  9000.00 |
|         104 | Bruce       | Ernst       | 590.423.4568 |  6000.00 |
|         105 | David       | Austin      | 590.423.4569 |  4800.00 |
|         106 | Valli       | Pataballa   | 590.423.4560 |  4800.00 |
|         107 | Diana       | Lorentz     | 590.423.5567 |  4200.00 |
|         108 | Nancy       | Greenberg   | 515.124.4569 | 12000.00 |
|         109 | Daniel      | Faviet      | 515.124.4169 |  9000.00 |
|         110 | John        | Chen        | 515.124.4269 |  8200.00 |
+-------------+-------------+-------------+--------------+----------+
```

Modello ^

Seleziona tutti i dipendenti con `FIRST_NAME` iniziano con **N.**

domanda

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^N'
-- Pattern start with---------------------------^
```

Modello $ **

Seleziona tutti i dipendenti il cui `PHONE_NUMBER` termina con **4569** .

domanda

```
SELECT * FROM employees WHERE PHONE_NUMBER REGEXP '4569$'
-- Pattern end with---------------------------------^
```

NON REGEXP

Seleziona tutti i dipendenti il cui `FIRST_NAME` *non* inizia con **N.**

domanda

```
SELECT * FROM employees WHERE FIRST_NAME NOT REGEXP '^N'
-- Pattern does not start with---------------^
```

Regex Contenere

Seleziona tutti i dipendenti il cui `LAST_NAME` contiene **e** la cui `FIRST_NAME` contiene `a` .

domanda

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP 'a' AND LAST_NAME REGEXP 'in'
-- No ^ or $, pattern can be anywhere ------------------------------------^
```

Qualsiasi personaggio tra []

Seleziona tutti i dipendenti il cui `FIRST_NAME` inizia con **A** o **B** o **C.**

domanda

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^[ABC]'
-------------------------------------------------^^---^
```

Modello o |

Seleziona tutti i dipendenti il cui `FIRST_NAME` inizia con **A** o **B** o **C** e termina con **r** , **e** o **i** .

domanda

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^[ABC]|[rei]$'
-- ---------------------------------------------^^---^^^---^^
```

Conteggio delle corrispondenze di espressioni regolari

Considera la seguente query:

```
SELECT FIRST_NAME, FIRST_NAME REGEXP '^N' as matching FROM employees
```

`FIRST_NAME REGEXP '^N'` è *1* o *0 a* seconda del fatto che `FIRST_NAME` corrisponda a `^N`

Per visualizzarlo meglio:

```
SELECT
FIRST_NAME,
IF(FIRST_NAME REGEXP '^N', 'matches ^N', 'does not match ^N') as matching
FROM employees
```

Infine, conta il numero totale di righe corrispondenti e non corrispondenti con:

```
SELECT
IF(FIRST_NAME REGEXP '^N', 'matches ^N', 'does not match ^N') as matching,
COUNT(*)
FROM employees
GROUP BY matching
```

Leggi Espressioni regolari online: https://riptutorial.com/it/mysql/topic/9444/espressioni-regolari

Capitolo 26: Estrai valori dal tipo JSON

introduzione

MySQL 5.7.8+ supporta il tipo JSON nativo. Mentre hai diversi modi per creare oggetti JSON, puoi anche accedere e leggere i membri in modi diversi.

La funzione principale è `JSON_EXTRACT` , quindi gli operatori `->` e `->>` sono più amichevoli.

Sintassi

- JSON_EXTRACT (json_doc, percorso [...])
- JSON_EXTRACT (json_doc, percorso)
- JSON_EXTRACT (json_doc, path1, path2)

Parametri

Parametro	Descrizione
json_doc	documento JSON valido
sentiero	percorso dei membri

Osservazioni

Citato in MySQL 5.7 Manuale di riferimento

- Più valori abbinati per argomento / i di percorso

  Se è possibile che tali argomenti possano restituire più valori, i valori corrispondenti vengono autowrapped come array, nell'ordine corrispondente ai percorsi che li hanno prodotti. Altrimenti, il valore di ritorno è il singolo valore corrispondente.

- `NULL` Risultato quando:
  - qualsiasi argomento è NULL
  - percorso non corrispondente

  Restituisce NULL se qualsiasi argomento è NULL o nessun percorso trova un valore nel documento.

Examples

Leggi il valore dell'array JSON

Crea la variabile @myjson come tipo JSON (leggi di più):

```
SET @myjson = CAST('["A","B",{"id":1,"label":"C"}]' as JSON) ;
```

`SELECT` alcuni membri!

```
SELECT
  JSON_EXTRACT( @myjson , '$[1]' ) ,
  JSON_EXTRACT( @myjson , '$[*].label') ,
  JSON_EXTRACT( @myjson , '$[1].*' ) ,
  JSON_EXTRACT( @myjson , '$[2].*')
;
-- result values:
'\"B\"', '[\"C\"]', NULL, '[1, \"C\"]'
-- visually:
"B", ["C"], NULL, [1, "C"]
```

Operatori di estrazione JSON

Estrai `path` da `->` o `->>` Operatori, mentre `->>` è UNQUOTED valore:

```
SELECT
  myjson_col->>'$[1]' , myjson_col->'$[1]' ,
  myjson_col->>'$[*].label' ,
  myjson_col->>'$[1].*' ,
  myjson_col->>'$[2].*'
FROM tablename ;
 -- visuall:
    B, "B" , ["C"], NULL, [1, "C"]
 __^^^ ^^^
```

Quindi `col->>path` è uguale a `JSON_UNQUOTE(JSON_EXTRACT(col,path))` :

> Come con ->, l'operatore - >> è sempre espanso nell'output di EXPLAIN, come dimostra il seguente esempio:

```
mysql> EXPLAIN SELECT c->>'$.name' AS name
    ->     FROM jemp WHERE g > 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: jemp
   partitions: NULL
         type: range
possible_keys: i
          key: i
      key_len: 5
          ref: NULL
         rows: 2
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
```

```
  Level: Note
   Code: 1003
Message: /* select#1 */ select
json_unquote(json_extract(`jtest`.`jemp`.`c`,'$.name')) AS `name` from
`jtest`.`jemp` where (`jtest`.`jemp`.`g` > 2)
1 row in set (0.00 sec)
```

Leggi l'estratto del percorso inline (+)

Leggi Estrai valori dal tipo JSON online: https://riptutorial.com/it/mysql/topic/9042/estrai-valori-dal-tipo-json

Capitolo 27: eventi

Examples

Crea un evento

Mysql ha la sua funzionalità EVENT per evitare complicate interazioni di cron quando gran parte di ciò che si sta programmando è correlato a SQL e meno relativo ai file. Vedi la pagina del manuale qui . Pensa agli eventi come stored procedure pianificate per essere eseguite su intervalli ricorrenti.

Per risparmiare tempo nel debug dei problemi relativi agli eventi, tenere presente che il gestore eventi globale deve essere attivato per elaborare gli eventi.

```
SHOW VARIABLES WHERE variable_name='event_scheduler';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | OFF   |
+-----------------+-------+
```

Con esso OFF, non si innescherà nulla. Quindi accendilo:

```
SET GLOBAL event_scheduler = ON;
```

Schema per i test

```
create table theMessages
(   id INT AUTO_INCREMENT PRIMARY KEY,
    userId INT NOT NULL,
    message VARCHAR(255) NOT NULL,
    updateDt DATETIME NOT NULL,
    KEY(updateDt)
);

INSERT theMessages(userId,message,updateDt) VALUES (1,'message 123','2015-08-24 11:10:09');
INSERT theMessages(userId,message,updateDt) VALUES (7,'message 124','2015-08-29');
INSERT theMessages(userId,message,updateDt) VALUES (1,'message 125','2015-09-03 12:00:00');
INSERT theMessages(userId,message,updateDt) VALUES (1,'message 126','2015-09-03 14:00:00');
```

Gli inserti di cui sopra sono forniti per mostrare un punto di partenza. Tieni presente che i 2 eventi creati di seguito eliminano le righe.

Crea 2 eventi, 1a corsa al giorno, 2a corsa ogni 10 minuti

Ignora ciò che stanno effettivamente facendo (giocando l'uno contro l'altro). Il punto è su INTERVAL e pianificazione.

```
DROP EVENT IF EXISTS `delete7DayOldMessages`;
DELIMITER $$
CREATE EVENT `delete7DayOldMessages`
  ON SCHEDULE EVERY 1 DAY STARTS '2015-09-01 00:00:00'
  ON COMPLETION PRESERVE
DO BEGIN
   DELETE FROM theMessages
   WHERE datediff(now(),updateDt)>6; -- not terribly exact, yesterday but <24hrs is still 1
day

  -- Other code here

END$$
DELIMITER ;
```

...

```
DROP EVENT IF EXISTS `Every_10_Minutes_Cleanup`;
DELIMITER $$
CREATE EVENT `Every_10_Minutes_Cleanup`
  ON SCHEDULE EVERY 10 MINUTE STARTS '2015-09-01 00:00:00'
  ON COMPLETION PRESERVE
DO BEGIN
   DELETE FROM theMessages
   WHERE TIMESTAMPDIFF(HOUR, updateDt, now())>168; -- messages over 1 week old (168 hours)

   -- Other code here
END$$
DELIMITER ;
```

Mostra stati degli eventi (approcci diversi)

```
SHOW EVENTS FROM my_db_name; -- List all events by schema name (db name)
SHOW EVENTS;
SHOW EVENTS\G; -- <--------- I like this one from mysql> prompt

*************************** 1. row ***************************
                  Db: my_db_name
                Name: delete7DayOldMessages
             Definer: root@localhost
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 1
      Interval field: DAY
              Starts: 2015-09-01 00:00:00
                Ends: NULL
              Status: ENABLED
          Originator: 1
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
*************************** 2. row ***************************
                  Db: my_db_name
                Name: Every_10_Minutes_Cleanup
             Definer: root@localhost
           Time zone: SYSTEM
```

```
               Type: RECURRING
          Execute at: NULL
      Interval value: 10
      Interval field: MINUTE
             Starts: 2015-09-01 00:00:00
               Ends: NULL
             Status: ENABLED
          Originator: 1
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
2 rows in set (0.06 sec)
```

Roba a caso da considerare

`DROP EVENT someEventName;` - Elimina l'evento e il suo codice

`ON COMPLETION PRESERVE` - Al termine dell'elaborazione dell'evento, conservarlo. Altrimenti, viene eliminato.

Gli eventi sono come trigger. Non sono chiamati dal programma di un utente. Piuttosto, sono programmati. In quanto tali, riescono o falliscono silenziosamente.

Il collegamento alla pagina di manuale mostra un po 'di flessibilità con le opzioni di intervallo, mostrate di seguito:

> intervallo:
>
> ```
> quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
>           WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
>           DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
> ```

Gli eventi sono potenti meccanismi che gestiscono attività ricorrenti e pianificate per il tuo sistema. Possono contenere tante istruzioni, routine DDL e DML e join complicati come si potrebbe ragionevolmente desiderare. Si prega di consultare la pagina del manuale MySQL intitolata Restrizioni sui programmi memorizzati .

Leggi eventi online: https://riptutorial.com/it/mysql/topic/4319/eventi

Capitolo 28: Gestione dei fusi orari

Osservazioni

Quando hai bisogno di gestire le informazioni sul tempo per una base di utenti in tutto il mondo in MySQL, usa il tipo di dati TIMESTAMP nelle tue tabelle.

Per ogni utente, memorizzare una colonna del fuso orario delle preferenze dell'utente. VARCHAR (64) è un buon tipo di dati per quella colonna. Quando un utente si registra per utilizzare il sistema, chiedere il valore del fuso orario. Il mio è Atlantic Time, `America/Edmonton` . Il vostro potrebbe o potrebbe non essere l' `Asia/Kolkata` o l' `Australia/NSW` . Per un'interfaccia utente per questa impostazione delle preferenze dell'utente, il software WordPress.org ha un buon esempio.

Infine, ogni volta che si stabilisce una connessione dal proprio programma host (Java, php, qualunque) al proprio DBMS per conto di un utente, emettere il comando SQL

```
SET SESSION time_zone='(whatever tz string the user gave you)'
```

prima di gestire qualsiasi dato utente che coinvolga tempi. Quindi tutte le volte `TIMESTAMP` installate verranno visualizzate nell'ora locale dell'utente.

Ciò causerà la conversione di tutti i tempi nelle tue tabelle in UTC e tutte le volte che verranno tradotte in locale. Funziona correttamente per NOW () e CURDATE (). Anche in questo caso, è necessario utilizzare TIMESTAMP e non i tipi di dati DATETIME o DATE per questo.

Assicurati che il tuo sistema operativo server e i fusi orari MySQL predefiniti siano impostati su UTC. Se non lo fai prima di iniziare a caricare le informazioni nel tuo database, sarà quasi impossibile correggerle. Se si utilizza un fornitore per eseguire MySQL, insistere sul fatto che abbia ragione.

Examples

Recupera la data e l'ora correnti in un determinato fuso orario.

Questo recupera il valore di `NOW()` in ora locale, in India, ora standard e poi di nuovo in UTC.

```
SELECT NOW();
SET time_zone='Asia/Kolkata';
SELECT NOW();
SET time_zone='UTC';
SELECT NOW();
```

Converti un valore `DATE` o` DATETIME` memorizzato in un altro fuso orario.

Se hai un `DATE` o `DATETIME` memorizzato (in una colonna da qualche parte) è stato memorizzato rispetto ad un certo fuso orario, ma in MySQL il fuso orario *non* è memorizzato con il valore.

Quindi, se vuoi convertirlo in un altro fuso orario, puoi, ma devi conoscere il fuso orario originale. L'utilizzo di `CONVERT_TZ()` esegue la conversione. Questo esempio mostra le righe vendute in California in ora locale.

```
SELECT CONVERT_TZ(date_sold,'UTC','America/Los_Angeles') date_sold_local
  FROM sales
 WHERE state_sold = 'CA'
```

Recupera i valori memorizzati di `TIMESTAMP` in un determinato fuso orario

Questo è veramente facile Tutti i valori `TIMESTAMP` sono memorizzati in tempo universale e sempre convertiti all'impostazione `time_zone` attuale ogni volta che vengono renderizzati.

```
SET SESSION time_zone='America/Los_Angeles';
SELECT timestamp_sold
  FROM sales
 WHERE state_sold = 'CA'
```

Perchè è questo? `TIMESTAMP` valori `TIMESTAMP` sono basati sul venerabile tipo di dati `time_t` UNIX . Questi timestamp UNIX sono memorizzati come un numero di secondi dal `1970-01-01 00:00:00` UTC.

I valori `TIMESTAMP` sono memorizzati nel tempo universale. `DATE` valori `DATE` e `DATETIME` sono memorizzati in qualunque ora locale era in vigore al momento della memorizzazione.

Qual è l'impostazione del fuso orario locale del mio server?

Ogni server ha un'impostazione globale time_zone predefinita, configurata dal proprietario della macchina server. Puoi trovare il fuso orario corrente in questo modo:

```
SELECT @@time_zone
```

Sfortunatamente, questo di solito dà il valore `SYSTEM` , il che significa che il tempo di MySQL è regolato dall'impostazione del fuso orario del SO del server.

Questa sequenza di query (sì, è un hack) restituisce l'offset in minuti tra l'impostazione del fuso orario del server e l'ora UTC.

```
CREATE TEMPORARY TABLE times (dt DATETIME, ts TIMESTAMP);
SET time_zone = 'UTC';
INSERT INTO times VALUES(NOW(), NOW());
SET time_zone = 'SYSTEM';
SELECT dt, ts, TIMESTAMPDIFF(MINUTE, dt, ts)offset FROM times;
DROP TEMPORARY TABLE times;
```

Come funziona? Le due colonne nella tabella temporanea con diversi tipi di dati sono la chiave. `DATETIME` tipi di dati `DATETIME` sono sempre memorizzati in ora locale nelle tabelle e `TIMESTAMP` in UTC. Quindi l'istruzione `INSERT` , eseguita quando time_zone è impostata su UTC, memorizza due valori identici di data / ora.

Quindi, l'istruzione SELECT, viene eseguita quando time_zone è impostato sull'ora locale del server. `TIMESTAMP` vengono sempre tradotti dal loro modulo UTC memorizzato in ora locale nelle istruzioni SELECT. `DATETIME` s non lo sono. Quindi l'operazione `TIMESTAMPDIFF(MINUTE...)` calcola la differenza tra ora locale e universale.

Quali valori time_zone sono disponibili nel mio server?

Per ottenere un elenco dei possibili valori time_zone nell'istanza del server MySQL, utilizzare questo comando.

```
SELECT mysql.time_zone_name.name
```

Normalmente, questo mostra l' elenco ZoneInfo dei fusi orari gestito da Paul Eggert presso l' Internet Numbers Authority Assigned . In tutto il mondo ci sono approssimativamente 600 fusi orari.

Sistemi operativi simil-Unix (distribuzioni Linux, distribuzioni BSD e moderne distribuzioni Mac OS, ad esempio) ricevono aggiornamenti di routine. L'installazione di questi aggiornamenti su un sistema operativo consente alle istanze MySQL che eseguono lì di tenere traccia delle modifiche apportate ai cambi di fuso orario e di ora legale / ora solare.

Se si ottiene un elenco molto più breve di nomi di fuso orario, il server è configurato in modo incompleto o in esecuzione su Windows. Ecco le istruzioni per l'amministratore del tuo server per installare e gestire l'elenco ZoneInfo.

Leggi Gestione dei fusi orari online: https://riptutorial.com/it/mysql/topic/7849/gestione-dei-fusi-orari

Capitolo 29: Gestione di dati sparsi o mancanti

Examples

Lavorare con le colonne che contengono valori NULL

In MySQL e in altri dialetti SQL, i valori `NULL` hanno proprietà speciali.

Considera la seguente tabella contenente i candidati al lavoro, le società per le quali hanno lavorato e la data in cui hanno lasciato la società. `NULL` indica che un richiedente lavora ancora presso l'azienda:

```
CREATE TABLE example
(`applicant_id` INT, `company_name` VARCHAR(255), `end_date` DATE);

+--------------+-----------------+------------+
| applicant_id | company_name    | end_date   |
+--------------+-----------------+------------+
|            1 | Google          | NULL       |
|            1 | Initech         | 2013-01-31 |
|            2 | Woodworking.com | 2016-08-25 |
|            2 | NY Times        | 2013-11-10 |
|            3 | NFL.com         | 2014-04-13 |
+--------------+-----------------+------------+
```

Il tuo compito è quello di comporre una query che restituisce tutte le righe dopo il `2016-01-01` , compresi i dipendenti che lavorano ancora in un'azienda (quelli con date di fine `NULL`). Questa affermazione selezionata:

```
SELECT * FROM example WHERE end_date > '2016-01-01';
```

non riesce a includere alcuna riga con valori `NULL` :

```
+--------------+-----------------+------------+
| applicant_id | company_name    | end_date   |
+--------------+-----------------+------------+
|            2 | Woodworking.com | 2016-08-25 |
+--------------+-----------------+------------+
```

Secondo la documentazione MySQL , i confronti usando gli operatori aritmetici <,>, = e <> restituiscono `NULL` invece di un valore booleano `TRUE` o `FALSE` . Pertanto una riga con un end_date `NULL` non è né maggiore del 2016-01-01 né inferiore al 2016-01-01.

Questo può essere risolto utilizzando le parole chiave IS NULL:

```
SELECT * FROM example WHERE end_date > '2016-01-01' OR end_date IS NULL;
```

```
+--------------+-----------------+------------+
| applicant_id | company_name    | end_date   |
+--------------+-----------------+------------+
|            1 | Google          | NULL       |
|            2 | Woodworking.com | 2016-08-25 |
+--------------+-----------------+------------+
```

Lavorare con NULL diventa più complesso quando l'attività coinvolge funzioni di aggregazione come `MAX()` e una clausola `GROUP BY` . Se la tua attività dovesse selezionare la data di assunzione più recente per ciascun candidato id, la seguente query sembrerebbe un primo tentativo logico:

```
SELECT applicant_id, MAX(end_date) FROM example GROUP BY applicant_id;

+--------------+---------------+
| applicant_id | MAX(end_date) |
+--------------+---------------+
|            1 | 2013-01-31    |
|            2 | 2016-08-25    |
|            3 | 2014-04-13    |
+--------------+---------------+
```

Tuttavia, sapendo che `NULL` indica che un candidato è ancora impiegato in un'azienda, la prima riga del risultato è imprecisa. L'utilizzo di `CASE WHEN` fornisce una soluzione alternativa per il problema `NULL` :

```
 SELECT
    applicant_id,
    CASE WHEN MAX(end_date is null) = 1 THEN 'present' ELSE MAX(end_date) END
    max_date
FROM example
GROUP BY applicant_id;

+--------------+------------+
| applicant_id | max_date   |
+--------------+------------+
|            1 | present    |
|            2 | 2016-08-25 |
|            3 | 2014-04-13 |
+--------------+------------+
```

Questo risultato può essere ricollocato nella tabella di `example` originale per determinare l'azienda a cui ha lavorato per ultimo un richiedente:

```
SELECT
  data.applicant_id,
  data.company_name,
  data.max_date
FROM (
  SELECT
    *,
    CASE WHEN end_date is null THEN 'present' ELSE end_date END max_date
  FROM example
) data
INNER JOIN (
 SELECT
   applicant_id,
```

```
   CASE WHEN MAX(end_date is null) = 1 THEN 'present' ELSE MAX(end_date) END max_date
 FROM
   example
 GROUP BY applicant_id
) j
ON data.applicant_id = j.applicant_id AND data.max_date = j.max_date;

+--------------+-----------------+------------+
| applicant_id | company_name    | max_date   |
+--------------+-----------------+------------+
|            1 | Google          | present    |
|            2 | Woodworking.com | 2016-08-25 |
|            3 | NFL.com         | 2014-04-13 |
+--------------+-----------------+------------+
```

Questi sono solo alcuni esempi di come lavorare con valori `NULL` in MySQL.

Leggi Gestione di dati sparsi o mancanti online: https://riptutorial.com/it/mysql/topic/5866/gestione-di-dati-sparsi-o-mancanti

Capitolo 30: Indici e chiavi

Sintassi

- - Crea un indice semplice

  CREATE INDEX *nome_indice* ON *nome_tabella* (*nome_colonna1* [, *nome_colonna2* , ...])

- - Crea un indice univoco

  CREATE UNICO INDICE *nome_indice* ON *nome_tabella* (*nome_colonna1* [, *nome_colonna2* , ...]

- - Indice di caduta

  DROP INDEX **nome_indice** ON **nome_tabella [algorithm_option** | **lock_option**] ...

  algorithm_option: ALGORITHM [=] {DEFAULT | INPLACE | COPY}

  lock_option: LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}

Osservazioni

concetti

Un indice in una tabella MySQL funziona come un indice in un libro.

Diciamo che hai un libro sui database e vuoi trovare alcune informazioni su, per esempio, sull'archiviazione. Senza un indice (supponendo che non ci siano altri aiuti, come un sommario) dovresti passare da una pagina all'altra, finché non trovi l'argomento (è una "scansione completa della tabella"). D'altra parte, un indice ha un elenco di parole chiave, quindi è necessario consultare l'indice e vedere che lo spazio di archiviazione è menzionato nelle pagine 113-120, 231 e 354. Quindi è possibile passare direttamente a quelle pagine, senza cercare (che è una ricerca con un indice, un po 'più veloce).

Ovviamente, l'utilità dell'indice dipende da molte cose: alcuni esempi, usando la similitudine sopra:

- Se hai un libro sui database e indicizzato la parola "database", potresti vedere che è menzionato alle pagine 1-59, 61-290 e 292-400. Sono un sacco di pagine e, in tal caso, l'indice non è di grande aiuto e potrebbe essere più veloce scorrere le pagine una alla volta. (In un database, questa è "scarsa selettività".)
- Per un libro di 10 pagine, non ha senso fare un indice, poiché potresti finire con un libro di 10 pagine preceduto da un indice di 5 pagine, il che è semplicemente sciocco: basta scansionare le 10 pagine e farle con esso .
- L'indice deve anche essere utile - non c'è in genere alcun punto di indicizzazione, ad

esempio la frequenza della lettera "L" per pagina.

Examples

Crea indice

```
-- Create an index for column 'name' in table 'my_table'
CREATE INDEX idx_name ON my_table(name);
```

Crea un indice univoco

Un indice univoco impedisce l'inserimento di dati duplicati in una tabella. `NULL` valori `NULL` possono essere inseriti nelle colonne che fanno parte dell'indice univoco (poiché, per definizione, un valore `NULL` è diverso da qualsiasi altro valore, incluso un altro valore `NULL`)

```
-- Creates a unique index for column 'name' in table 'my_table'
CREATE UNIQUE INDEX idx_name ON my_table(name);
```

Indice di caduta

```
-- Drop an index for column 'name' in table 'my_table'
DROP INDEX idx_name ON my_table;
```

Crea indice composito

Questo creerà un indice composito di entrambe le chiavi, `mystring` e `mydatetime` e accelererà le query con entrambe le colonne nella clausola `WHERE` .

```
CREATE INDEX idx_mycol_myothercol ON my_table(mycol, myothercol)
```

Nota: l'ordine è importante! Se la query di ricerca non include entrambe le colonne nella clausola `WHERE` , può utilizzare solo l'indice più a sinistra. In questo caso, una query con `mycol` nel `WHERE` utilizzerà l'indice, una query alla ricerca di `myothercol` **senza** cercare anche `mycol` **non lo** farà. Per maggiori informazioni consulta questo post sul blog .

Nota: a causa del modo in cui funziona BTREE, le colonne che vengono solitamente interrogate in intervalli dovrebbero andare nel valore più a destra. Ad esempio, le colonne `DATETIME` vengono normalmente interrogate come `WHERE datecol > '2016-01-01 00:00:00'` . Gli indici BTREE gestiscono gli intervalli in modo molto efficiente, ma solo se la colonna interrogata come intervallo è l'ultima dell'indice composito.

Tasto AUTO_INCREMENT

```
CREATE TABLE (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    ...
```

```
    PRIMARY KEY(id),
    ...  );
```

Note principali:

- Inizia con 1 e incrementi di 1 automaticamente quando non si riesce a specificarlo su `INSERT` o specificarlo come `NULL` .
- Gli ID sono sempre distinti l'uno dall'altro, ma ...
- Non fare alcuna ipotesi (nessuna lacuna, generata consecutivamente, non riutilizzata, ecc.) Sui valori dell'id tranne che essere univoci in un dato istante.

Note sottili:

- Al riavvio del server, il valore 'successivo' è 'calcolato' come `MAX(id)+1` .
- Se l'ultima operazione prima dello spegnimento o crash andava a cancellare l'id più alto, quell'id *può* essere riutilizzato (dipende dal motore). Quindi, *non fidarti di auto_increments essere permanentemente univoci* ; sono unici in ogni momento.
- Per soluzioni multi-master o cluster, vedere `auto_increment_offset` e `auto_increment_increment` .
- Va bene avere qualcos'altro come `PRIMARY KEY` e semplicemente `INDEX(id)` . (Questa è un'ottimizzazione in alcune situazioni).
- L'utilizzo di `AUTO_INCREMENT` come "tasto `PARTITION` " è raramente utile; fare qualcosa di diverso.
- Varie operazioni *possono* "bruciare" i valori. Ciò accade quando pre-allocano i valori, quindi non li usano: `INSERT IGNORE` (con tasto dup), `REPLACE` (che è `DELETE` più `INSERT`) e altri. `ROLLBACK` è un'altra causa di lacune negli ID.
- In Replica, non è possibile fidarsi degli ID per arrivare allo / agli schiavo / i in ordine crescente. Sebbene gli ID siano assegnati in ordine consecutivo, le istruzioni InnoDB vengono inviate agli slave in ordine `COMMIT` .

Leggi Indici e chiavi online: https://riptutorial.com/it/mysql/topic/1748/indici-e-chiavi

Capitolo 31: Informazioni sul server

Parametri

parametri	Spiegazione
GLOBALE	Mostra le variabili così come sono configurate per l'intero server. Opzionale.
SESSIONE	Mostra le variabili che sono configurate solo per questa sessione. Opzionale.

Examples

MOSTRA L'esempio VARIABILI

Per ottenere tutte le variabili del server, esegui questa query nella finestra SQL della tua interfaccia preferita (PHPMyAdmin o altro) o nell'interfaccia CLI MySQL

```
SHOW VARIABLES;
```

Puoi specificare se vuoi le variabili di sessione o le variabili globali come segue:

Variabili di sessione:

```
SHOW SESSION VARIABLES;
```

Variabili globali:

```
SHOW GLOBAL VARIABLES;
```

Come qualsiasi altro comando SQL è possibile aggiungere parametri alla query come il comando LIKE:

```
SHOW [GLOBAL | SESSION] VARIABLES LIKE 'max_join_size';
```

Oppure, usando i caratteri jolly:

```
SHOW [GLOBAL | SESSION] VARIABLES LIKE '%size%';
```

È inoltre possibile filtrare i risultati della query SHOW utilizzando un parametro WHERE come segue:

```
SHOW [GLOBAL | SESSION] VARIABLES WHERE VALUE > 0;
```

SHOW STATUS esempio

Per ottenere lo stato del server database, eseguire questa query nella finestra SQL dell'interfaccia preferita (PHPMyAdmin o altro) o nell'interfaccia CLI MySQL.

```
SHOW STATUS;
```

È possibile specificare se si desidera ricevere lo stato SESSION o GLOBAL del proprio server in questo modo: Stato della sessione:

```
SHOW SESSION STATUS;
```

Stato globale:

```
SHOW GLOBAL STATUS;
```

Come qualsiasi altro comando SQL è possibile aggiungere parametri alla query come il comando LIKE:

```
SHOW [GLOBAL | SESSION] STATUS LIKE 'Key%';
```

O il comando Where:

```
SHOW [GLOBAL | SESSION] STATUS WHERE VALUE > 0;
```

La principale differenza tra GLOBAL e SESSION è che con il modificatore GLOBAL il comando visualizza informazioni aggregate sul server e su tutte le sue connessioni, mentre il modificatore SESSION mostrerà solo i valori per la connessione corrente.

Leggi Informazioni sul server online: https://riptutorial.com/it/mysql/topic/9924/informazioni-sul-server

Capitolo 32: INSERIRE

Sintassi

1. INSERISCI [LOW_PRIORITY | RITARDATO | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (partition_name, ...)] [(col_name, ...)] {VALUES | VALUE} ({expr | DEFAULT}, ...), (...), ... [ON DUPLICATE KEY UPDATE col_name = expr [, nome_col = expr] ...]

2. INSERISCI [LOW_PRIORITY | RITARDATO | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (partition_name, ...)] SET col_name = {expr | DEFAULT}, ... [ON DUPLICATE KEY UPDATE col_name = expr [, nome_col = expr] ...]

3. INSERISCI [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (partition_name, ...)] [(nome_col, ...)] SELECT ... [ON DUPLICATE KEY UPDATE col_name = expr [, nome_col = expr] ...]

4. Un'espr espressione può riferirsi a qualsiasi colonna che è stata impostata in precedenza in una lista valori. Ad esempio, puoi farlo perché il valore di col2 si riferisce a col1, che è stato precedentemente assegnato:
   INSERISCI IN VALORI tbl_name (col1, col2) (15, col1 * 2);

5. Le istruzioni INSERT che utilizzano la sintassi VALUES possono inserire più righe. Per fare ciò, includere più elenchi di valori di colonna, ciascuno racchiuso tra parentesi e separati da virgole. Esempio:
   INSERISCI IN VALORI tbl_name (a, b, c) (1,2,3), (4,5,6), (7,8,9);

6. L'elenco dei valori per ogni riga deve essere racchiuso tra parentesi. La seguente dichiarazione è illegale perché il numero di valori nell'elenco non corrisponde al numero di nomi di colonna:
   INSERISCI IN VALORI tbl_name (a, b, c) (1,2,3,4,5,6,7,8,9);

7. INSERIRE ... SELEZIONA Sintassi
   INSERISCI [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (partition_name, ...)] [(nome_col, ...)] SELECT ... [ON DUPLICATE KEY UPDATE col_name = expr, ...]

8. Con INSERT ... SELECT, puoi inserire rapidamente molte righe in una tabella da una o più tabelle. Per esempio:
   INSERISCI IN tbl_temp2 (fld_id) SELECT tbl_temp1.fld_order_id FROM tbl_temp1 WHERE tbl_temp1.fld_order_id> 100;

Osservazioni

Sintassi ufficiale INSERT

Examples

Inserto di base

```
INSERT INTO `table_name` (`field_one`, `field_two`) VALUES ('value_one', 'value_two');
```

In questo banale esempio, `table_name` è dove i dati devono essere aggiunti, `field_one` e `field_two` sono campi per impostare i dati contro, `value_one` e `value_two` sono i dati da fare rispettivamente contro `field_one` e `field_two` .

È buona norma elencare i campi in cui stai inserendo i dati all'interno del tuo codice, come se la tabella cambi e nuove colonne vengano aggiunte, il tuo inserto si spezzerebbe se non fossero lì

INSERT, ON DUPLICATE KEY UPDATE

```
INSERT INTO `table_name`
    (`index_field`, `other_field_1`, `other_field_2`)
    VALUES
    ('index_value', 'insert_value', 'other_value')
    ON DUPLICATE KEY UPDATE
        `other_field_1` = 'update_value',
        `other_field_2` = VALUES(`other_field_2`);
```

Questo `INSERT` in `table_name` i valori specificati, ma se la chiave univoca esiste già, aggiornerà il `other_field_1` per avere un nuovo valore.
A volte, quando si aggiorna la chiave duplicata, è utile usare `VALUES()` per accedere al valore originale che è stato passato a `INSERT` invece di impostare direttamente il valore. In questo modo, puoi impostare valori diversi usando `INSERT` e `UPDATE` . Vedere l'esempio sopra dove `other_field_1` è impostato su `insert_value` su `INSERT` o su `update_value` su `UPDATE` mentre `other_field_2` è sempre impostato su `other_value` .

Fondamentale per l'Insert su Duplicate Key Update (IODKU) da utilizzare è lo schema contenente una chiave univoca che segnalerà uno scontro duplicato. Questa chiave univoca può essere una chiave primaria o meno. Può essere una chiave univoca su una singola colonna o una multi-colonna (chiave composita).

Inserimento di più righe

```
INSERT INTO `my_table` (`field_1`, `field_2`) VALUES
    ('data_1', 'data_2'),
    ('data_1', 'data_3'),
    ('data_4', 'data_5');
```

Questo è un modo semplice per aggiungere più righe contemporaneamente con una sola istruzione `INSERT` .

Questo tipo di inserto "batch" è molto più veloce dell'inserimento di righe una per una. In genere, l'inserimento di 100 righe in un singolo batch in questo modo è 10 volte più veloce di inserirle tutte

singolarmente.

Ignorando le righe esistenti

Quando si importano insiemi di dati di grandi dimensioni, in determinate circostanze potrebbe essere preferibile saltare le righe che di solito causano il fallimento della query a causa di un vincolo di colonna, ad esempio chiavi primarie duplicate. Questo può essere fatto usando `INSERT IGNORE` .

Considera il seguente database di esempio:

```
SELECT * FROM `people`;
--- Produces:
+----+------+
| id | name |
+----+------+
|  1 | john |
|  2 | anna |
+----+------+


INSERT IGNORE INTO `people` (`id`, `name`) VALUES
    ('2', 'anna'), --- Without the IGNORE keyword, this record would produce an error
    ('3', 'mike');

SELECT * FROM `people`;
--- Produces:
+----+--------+
| id |  name  |
+----+--------+
|  1 |  john  |
|  2 |  anna  |
|  3 |  mike  |
+----+--------+
```

La cosa importante da ricordare è che *INSERTO IGNORE* salterà anche silenziosamente altri errori, ecco cosa dice la documentazione ufficiale di Mysql:

> Le conversioni di dati che potrebbero causare errori interrompono l'istruzione se IGNORE non è> specificato. Con IGNORE, i valori non validi vengono adattati ai valori più vicini e> inseriti; gli avvertimenti sono prodotti ma la dichiarazione non abortisce.

Nota: - La sezione seguente viene aggiunta per completezza, ma non è considerata la migliore pratica (questo fallirebbe, ad esempio, se un'altra colonna fosse stata aggiunta alla tabella).

Se si specifica il valore della colonna corrispondente per tutte le colonne nella tabella, è possibile ignorare l'elenco delle colonne `INSERT` come segue:

```
INSERT INTO `my_table` VALUES
        ('data_1', 'data_2'),
        ('data_1', 'data_3'),
        ('data_4', 'data_5');
```

INSERT SELECT (Inserimento di dati da un'altra tabella)

Questo è il modo base per inserire dati da un'altra tabella con l'istruzione SELECT.

```
INSERT INTO `tableA` (`field_one`, `field_two`)
   SELECT `tableB`.`field_one`, `tableB`.`field_two`
   FROM `tableB`
   WHERE `tableB`.clmn <> 'someValue'
   ORDER BY `tableB`.`sorting_clmn`;
```

È possibile selezionare `SELECT * FROM` , ma la `tableA` e la `tableB` *devono* avere il conteggio delle colonne corrispondente e i tipi di dati corrispondenti.

Le colonne con `AUTO_INCREMENT` sono trattate come nella clausola `INSERT` con `VALUES` .

Questa sintassi semplifica il riempimento di tabelle (temporanee) con dati di altre tabelle, ancor più quando i dati devono essere filtrati sull'inserto.

INSERISCI con AUTO_INCREMENT + LAST_INSERT_ID ()

Quando una tabella ha una `PRIMARY KEY AUTO_INCREMENT` , normalmente non viene inserita in quella colonna. Invece, specifica tutte le altre colonne, quindi chiedi quale fosse il nuovo ID.

```
CREATE TABLE t (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    this ...,
    that ...,
    PRIMARY KEY(id) );

INSERT INTO t (this, that) VALUES (..., ...);
SELECT LAST_INSERT_ID() INTO @id;
INSERT INTO another_table (..., t_id, ...) VALUES (..., @id, ...);
```

Nota che `LAST_INSERT_ID()` è legato alla sessione, quindi anche se più connessioni si stanno inserendo nella stessa tabella, ognuna con il proprio ID.

La tua API client probabilmente ha un modo alternativo per ottenere `LAST_INSERT_ID()` senza eseguire effettivamente una `SELECT` e restituire il valore al client invece di lasciarlo in una `@variable` all'interno di MySQL. Di solito è preferibile.

Esempio più lungo e dettagliato

L'utilizzo "normale" di IODKU è di attivare "chiave duplicata" in base a una chiave `UNIQUE` , non alla `AUTO_INCREMENT PRIMARY KEY` . Il seguente dimostra tale. Si noti che *non* fornisce l' `id` nell'INSERT.

Installazione per gli esempi da seguire:

```
CREATE TABLE iodku (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    misc INT NOT NULL,
    PRIMARY KEY(id),
```

```
    UNIQUE(name)
) ENGINE=InnoDB;

INSERT INTO iodku (name, misc)
    VALUES
    ('Leslie', 123),
    ('Sally', 456);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  |  456 |
+----+--------+------+
```

Il caso di IODKU che esegue un "aggiornamento" e `LAST_INSERT_ID()` recupera l' `id` rilevante:

```
INSERT INTO iodku (name, misc)
    VALUES
    ('Sally', 3333)            -- should update
    ON DUPLICATE KEY UPDATE    -- `name` will trigger "duplicate key"
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
SELECT LAST_INSERT_ID();       -- picking up existing value
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                2 |
+------------------+
```

Il caso in cui IODKU esegue un "inserimento" e `LAST_INSERT_ID()` recupera il nuovo `id` :

```
INSERT INTO iodku (name, misc)
    VALUES
    ('Dana', 789)          -- Should insert
    ON DUPLICATE KEY UPDATE
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
SELECT LAST_INSERT_ID();   -- picking up new value
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                3 |
+------------------+
```

Contenuto della tabella risultante:

```
SELECT * FROM iodku;
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  | 3333 |  -- IODKU changed this
|  3 | Dana   |  789 |  -- IODKU added this
+----+--------+------+
```

ID_INCREMENT perso

Diverse funzioni di "inserimento" possono "bruciare" id. Ecco un esempio, utilizzando InnoDB (altri motori potrebbero funzionare in modo diverso):

```
CREATE TABLE Burn (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
        ) ENGINE=InnoDB;

INSERT IGNORE INTO Burn (name) VALUES ('first'), ('second');
SELECT LAST_INSERT_ID();          -- 1
SELECT * FROM Burn ORDER BY id;
  +----+--------+
  |  1 | first  |
  |  2 | second |
  +----+--------+

INSERT IGNORE INTO Burn (name) VALUES ('second');  -- dup 'IGNOREd', but id=3 is burned
SELECT LAST_INSERT_ID();          -- Still "1" -- can't trust in this situation
SELECT * FROM Burn ORDER BY id;
  +----+--------+
  |  1 | first  |
  |  2 | second |
  +----+--------+

INSERT IGNORE INTO Burn (name) VALUES ('third');
SELECT LAST_INSERT_ID();           -- now "4"
SELECT * FROM Burn ORDER BY id;    -- note that id=3 was skipped over
  +----+--------+
  |  1 | first  |
  |  2 | second |
  |  4 | third  |    -- notice that id=3 has been 'burned'
  +----+--------+
```

Pensaci (all'incirca) in questo modo: prima l'inserto cerca di vedere quante righe *potrebbero* essere inserite. Quindi prendi tutti quei valori dall'auto_increment per quella tabella. Infine, inserisci le righe, usando gli ID secondo necessità e masterizzando gli eventuali rimanenti.

L'unica volta che i rimanenti sono recuperabili è se il sistema viene arrestato e riavviato. Al riavvio, viene effettivamente eseguito `MAX(id)` . Questo può riutilizzare id che sono stati masterizzati o liberati da `DELETEs` con l'id più alto.

Essenzialmente qualsiasi sapore di `INSERT` (incluso `REPLACE` , che è `DELETE` + `INSERT`) può masterizzare id. In InnoDB, la variabile globale (non di sessione!) `innodb_autoinc_lock_mode` può essere utilizzata per controllare parte di ciò che sta accadendo.

Quando si "normalizzano" le lunghe stringhe in un `AUTO INCREMENT id` , la masterizzazione può facilmente verificarsi. Ciò *potrebbe* portare a un sovraccarico delle dimensioni `INT` hai scelto.

Leggi INSERIRE online: https://riptutorial.com/it/mysql/topic/866/inserire

Capitolo 33: Installa il contenitore Mysql con Docker-Compose

Examples

Semplice esempio con docker-compose

Questo è un semplice esempio per creare un server mysql con finestra mobile

1.- crea **docker-compose.yml** :

Nota: se desideri utilizzare lo stesso contenitore per tutti i tuoi progetti, devi creare un PERCORSO nel tuo HOME_PATH. Se si desidera crearlo per ogni progetto, è possibile creare una directory **mobile** nel progetto.

```
version: '2'
services:
  cabin_db:
    image: mysql:latest
    volumes:
      - "./.mysql-data/db:/var/lib/mysql"
    restart: always
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: rootpw
      MYSQL_DATABASE: cabin
      MYSQL_USER: cabin
      MYSQL_PASSWORD: cabinpw
```

2.- eseguilo:

```
cd PATH_TO_DOCKER-COMPOSE.YML
docker-compose up -d
```

3.- connettersi al server

```
mysql -h 127.0.0.1 -u root -P 3306 -p rootpw
```

Evviva !!

4.- Arresta il server

```
docker-compose stop
```

Leggi Installa il contenitore Mysql con Docker-Compose online:
https://riptutorial.com/it/mysql/topic/4458/installa-il-contenitore-mysql-con-docker-compose

Capitolo 34: JSON

introduzione

A partire da MySQL 5.7.8, MySQL supporta un tipo di dati JSON nativo che consente un accesso efficiente ai dati nei documenti JSON (JavaScript Object Notation).
https://dev.mysql.com/doc/refman/5.7/en/json.html

Osservazioni

A partire da MySQL 5.7.8, MySQL viene fornito con un tipo JSON. Un sacco di sviluppatori hanno salvato i dati JSON nelle colonne di testo per un tempo di registrazione ma il tipo JSON è diverso, i dati vengono salvati in formato binario dopo la convalida. Questo evita il sovraccarico di analizzare il testo su ogni lettura.

Examples

Crea una tabella semplice con una chiave primaria e un campo JSON

```
CREATE TABLE table_name (
    id INT NOT NULL AUTO_INCREMENT,
    json_col JSON,
    PRIMARY KEY(id)
);
```

Inserisci un semplice JSON

```
INSERT INTO
    table_name (json_col)
VALUES
    ('{"City": "Galle", "Description": "Best damn city in the world"}');
```

È semplice come si può ottenere, ma si noti che, poiché le chiavi del dizionario JSON devono essere circondate da virgolette doppie, l'intera cosa deve essere racchiusa tra virgolette singole. Se la query ha esito positivo, i dati verranno archiviati in un formato binario.

Inserisci dati misti in un campo JSON.

Questo inserisce un dizionario json in cui uno dei membri è una matrice di stringhe nella tabella creata in un altro esempio.

```
INSERT INTO myjson(dict)
VALUES('{"opening":"Sicilian","variations":["pelikan","dragon","najdorf"]}');
```

Nota, ancora una volta, che devi fare attenzione con l'uso di virgolette singole e doppie. Il tutto

deve essere racchiuso tra virgolette singole.

Aggiornamento di un campo JSON

Nell'esempio precedente abbiamo visto come tipi di dati misti possono essere inseriti in un campo JSON. Cosa succede se vogliamo aggiornare quel campo? Stiamo andando ad aggiungere *scheveningen* alla matrice denominata `variations` nell'esempio precedente.

```
UPDATE
    myjson
SET
    dict=JSON_ARRAY_APPEND(dict,'$.variations','scheveningen')
WHERE
    id = 2;
```

Gli appunti:

1. L'array `$.variations` nel nostro dizionario json. Il simbolo $ rappresenta la documentazione di JSON. Per una spiegazione completa dei percorsi json riconosciuti da mysql, consultare https://dev.mysql.com/doc/refman/5.7/en/json-path-syntax.html
2. Poiché non abbiamo ancora un esempio sull'interrogazione utilizzando i campi JSON, questo esempio utilizza la chiave primaria.

Ora se facciamo `SELECT * FROM myjson` vedremo

```
+----+-------------------------------------------------------------------------------------------
-+
| id | dict
|
+---+--------------------------------------------------------------------------------------------
+
| 2  | {"opening": "Sicilian", "variations": ["pelikan", "dragon", "najdorf", "scheveningen"]}
|
+----+-------------------------------------------------------------------------------------------
-+
1 row in set (0.00 sec)
```

Dati CAST a tipo JSON

Converte stringhe json valide in tipo JSON MySQL:

```
 SELECT CAST('[1,2,3]' as JSON) ;
 SELECT CAST('{"opening":"Sicilian","variations":["pelikan","dragon","najdorf"]}' as JSON);
```

Crea oggetti e matrici Json

`JSON_OBJECT` crea oggetti JSON:

```
 SELECT JSON_OBJECT('key1',col1 , 'key2',col2 , 'key3','col3') as myobj;
```

`JSON_ARRAY` crea anche JSON Array:

```
 SELECT JSON_ARRAY(col1,col2,'col3') as myarray;
```

Nota: myobj.key3 e myarray [2] sono "col3" come stringa fissa.

Anche dati JSON misti:

```
 SELECT JSON_OBJECT("opening","Sicilian",
"variations",JSON_ARRAY("pelikan","dragon","najdorf") ) as mymixed ;
```

Leggi JSON online: https://riptutorial.com/it/mysql/topic/2985/json

Capitolo 35: Limite e offset

Sintassi

- SELEZIONA colonna_1 [, colonna_2]
  FROM table_1
  ORDINA PER order_column
  LIMIT row_count [OFFSET row_offset]
- SELEZIONA colonna_1 [, colonna_2]
  FROM table_1
  ORDINA PER order_column
  LIMIT [row_offset,] row_count

Osservazioni

"Limite" potrebbe significare "Numero massimo di righe in una tabella".

"Offset" significa scelta dal numero di `row` (da non confondere con il valore della chiave primaria o qualsiasi valore dei dati del campo)

Examples

Rapporto limite e offset

Considerando la seguente tabella `users` :

id	nome utente
1	Utente1
2	Utente2
3	user3
4	Utente4
5	User5

Per limitare il numero di righe nel set di risultati di una query `SELECT` , la clausola `LIMIT` può essere utilizzata insieme a uno o due numeri interi positivi come argomenti (incluso lo zero).

Clausola `LIMIT` con un argomento

Quando viene utilizzato un argomento, il set di risultati sarà limitato al numero specificato nel

seguente modo:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2
```

id	nome utente
1	Utente1
2	Utente2

Se il valore dell'argomento è `0` , il set di risultati sarà vuoto.

Si noti inoltre che la clausola `ORDER BY` può essere importante per specificare le prime righe del set di risultati che verrà presentato (quando si ordina da un'altra colonna).

Clausola `LIMIT` con due argomenti

Quando vengono utilizzati due argomenti in una clausola `LIMIT` :

- il **primo** argomento rappresenta la riga da cui verranno presentate le righe del set di risultati; questo numero viene spesso indicato come **offset** , poiché rappresenta la riga precedente alla riga iniziale del set di risultati vincolati. Ciò consente all'argomento di ricevere `0` come valore e quindi di prendere in considerazione la prima riga del set di risultati non vincolato.
- il **secondo** argomento specifica il numero massimo di righe da restituire nel set di risultati (in modo simile all'esempio di un argomento).

Pertanto la query:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2, 3
```

Presenta il seguente set di risultati:

id	nome utente
3	user3
4	Utente4
5	User5

Si noti che quando l'argomento di **offset** è `0` , il set di risultati sarà equivalente a una clausola `LIMIT` un argomento. Ciò significa che le seguenti 2 query:

```
SELECT * FROM users ORDER BY id ASC LIMIT 0, 2

SELECT * FROM users ORDER BY id ASC LIMIT 2
```

Produce lo stesso set di risultati:

id	nome utente
1	Utente1
2	Utente2

`OFFSET` chiave `OFFSET` : sintassi alternativa

Una sintassi alternativa per la clausola `LIMIT` con due argomenti consiste nell'utilizzo della parola chiave `OFFSET` dopo il primo argomento nel seguente modo:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2 OFFSET 3
```

Questa query restituirebbe il seguente set di risultati:

id	nome utente
3	user3
4	Utente4

Si noti che in questa sintassi alternativa gli argomenti hanno le loro posizioni commutate:

- il **primo** argomento rappresenta il numero di righe da restituire nel set di risultati;
- il **secondo** argomento rappresenta l'offset.

Leggi Limite e offset online: https://riptutorial.com/it/mysql/topic/548/limite-e-offset

Capitolo 36: Log files

Examples

Una lista

- Registro generale - tutte le query - vedi VARIABLE general_log
- Registro lento: query più lente di long_query_time - slow_query_log_file
- Binlog: per la replica e il backup: log_bin_basename
- Log di inoltro - anche per la replica
- errori generali - mysqld.err
- start / stop - mysql.log (non molto interessante) - log_error
- InnoDB redo log - iblog *

Vedere le variabili `basedir` e `datadir` per la posizione predefinita per molti registri

Alcuni registri sono attivati / disattivati da altre VARIABILI. Alcuni sono scritti su un file o su un tavolo.

(Nota per i revisori: questo richiede più dettagli e maggiori spiegazioni).

Documentatori : includere la posizione e il nome predefiniti per ciascun tipo di registro, sia per Windows che per * nix. (O almeno il più possibile.)

Log delle query lente

Il registro delle query lente è costituito da eventi di registro per le query che `long_query_time` fino a `long_query_time` secondi per terminare. Ad esempio, fino a 10 secondi per il completamento. Per vedere la soglia temporale attualmente impostata, emettere il seguente messaggio:

```
SELECT @@long_query_time;
+--------------------+
| @@long_query_time |
+--------------------+
|          10.000000 |
+--------------------+
```

Può essere impostato come variabile GLOBAL, nel file `my.cnf` o `my.ini` . Oppure può essere impostato dalla connessione, anche se questo è insolito. Il valore può essere impostato tra 0 e 10 (secondi). Quale valore usare?

- 10 è così alto da essere quasi inutile;
- 2 è un compromesso;
- 0,5 e altre frazioni sono possibili;
- 0 cattura tutto; questo potrebbe riempire il disco pericolosamente velocemente, ma può essere molto utile.

L'acquisizione di query lente è attivata o disattivata. E anche il file registrato è specificato. Il sotto cattura questi concetti:

```
SELECT @@slow_query_log; -- Is capture currently active? (1=On, 0=Off)
SELECT @@slow_query_log_file; -- filename for capture. Resides in datadir
SELECT @@datadir; -- to see current value of the location for capture file

SET GLOBAL slow_query_log=0; -- Turn Off
-- make a backup of the Slow Query Log capture file. Then delete it.
SET GLOBAL slow_query_log=1; -- Turn it back On (new empty file is created)
```

Per ulteriori informazioni, consultare la pagina del manuale MySQL. Il registro delle query lente

Nota: le informazioni di cui sopra sull'accensione / spegnimento del registro lento sono state modificate in 5.6 (?); la versione precedente aveva un altro meccanismo.

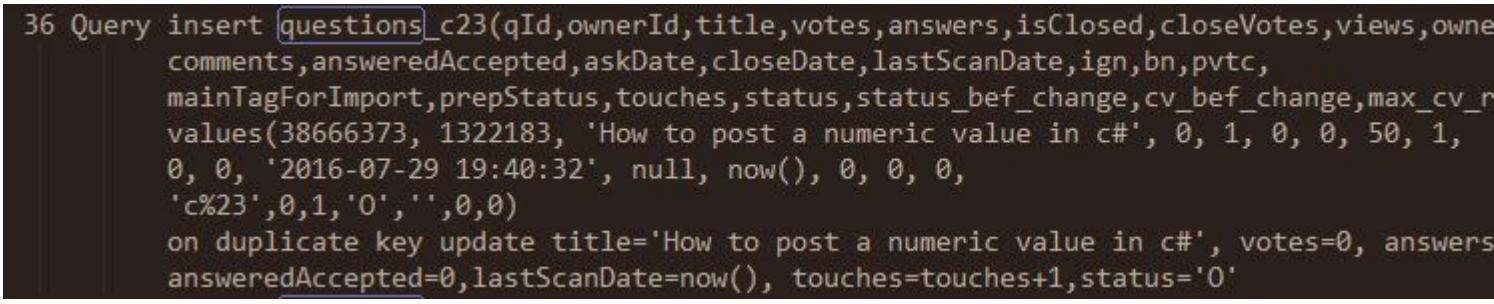
Il modo "migliore" per vedere cosa sta rallentando il tuo sistema:

```
long_query_time=...
turn on the slowlog
run for a few hours
turn off the slowlog (or raise the cutoff)
run pt-query-digest to find the 'worst' couple of queries.  Or mysqldumpslow -s t
```

Log di query generale

Il registro delle query generali contiene un elenco di informazioni generali da client si connette, disconnette e query. È di valore inestimabile per il debug, eppure si pone come un ostacolo alle prestazioni (citazione?).

Di seguito è riportata una vista di esempio di un log di query generale:



Per determinare se il registro generale è attualmente in fase di acquisizione:

```
SELECT @@general_log; -- 1 = Capture is active; 0 = It is not.
```

Per determinare il nome file del file di acquisizione:

```
SELECT @@general_log_file; -- Full path to capture file
```

Se il percorso completo del file non viene mostrato, il file esiste nel `datadir` .

Esempio di Windows:

```
+-----------------------------------------------------------+
| @@general_log_file                                        |
+-----------------------------------------------------------+
| C:\ProgramData\MySQL\MySQL Server 5.7\Data\GuySmiley.log |
+-----------------------------------------------------------+
```

Linux:

```
+----------------------------------+
| @@general_log_file               |
+----------------------------------+
| /var/lib/mysql/ip-ww-xx-yy-zz.log |
+----------------------------------+
```

Quando vengono apportate modifiche alla variabile GLOBAL `general_log_file` , il nuovo log viene salvato nel `datadir` . Tuttavia, il percorso completo potrebbe non riflettersi più esaminando la variabile.

Nel caso di nessuna voce per `general_log_file` nel file di configurazione, per impostazione predefinita sarà `@@hostname` .log nel `datadir` .

Le migliori pratiche sono per disattivare l'acquisizione. Salvare il file di registro in una directory di backup con un nome file che riflette il tempo di inizio / fine della cattura. Cancellare il file precedente se non si verificava un *trasferimento del* filesystem di quel file. Stabilire un nuovo nome file per il file di registro e attivare la cattura su ON (tutti vengono visualizzati di seguito). Le migliori pratiche includono anche un'attenta determinazione se vuoi anche catturare al momento. In genere, l'acquisizione è attiva solo a scopo di debug.

Un tipico nome file del filesystem per un registro di backup potrebbe essere:

```
/LogBackup/GeneralLog_20160802_1520_to_20160802_1815.log
```

dove la data e l'ora sono parte del nome del file come intervallo.

Per Windows, osserva la seguente sequenza con le modifiche alle impostazioni.

```
SELECT @@general_log; -- 0. Not being captured
SELECT @@general_log_file; -- C:\ProgramData\MySQL\MySQL Server 5.6\Data\GuySmiley.log
SELECT @@datadir; -- C:\ProgramData\MySQL\MySQL Server 5.7\Data\
SET GLOBAL general_log_file='GeneralLogBegin_20160803_1420.log'; -- datetime clue
SET GLOBAL general_log=1; -- Turns on actual log capture. File is created under `datadir`
SET GLOBAL general_log=0; -- Turn logging off
```

Linux è simile. Questi rappresenterebbero cambiamenti dinamici. Qualsiasi riavvio del server avrebbe rilevato le impostazioni del file di configurazione.

Per quanto riguarda il file di configurazione, considerare le seguenti impostazioni delle variabili rilevanti:

```
[mysqld]
general_log_file = /path/to/currentquery.log
general_log      = 1
```

Inoltre, la variabile `log_output` può essere configurata per l'output `TABLE` , non solo `FILE` . Per quello, per favore vedi Destinazioni .

Si prega di consultare la pagina del manuale MySQL Il registro generale delle query .

Registro errori

Il registro degli errori è popolato con le informazioni di avvio e arresto e gli eventi critici incontrati dal server.

Quello che segue è un esempio dei suoi contenuti:

```
2016-08-02 20:40:39 2420 [Note] Shutting down plugin 'binlog'
2016-08-02 20:40:39 2420 [Note] mysqld: Shutdown complete

2016-08-02 20:43:11 2888 [Note] Plugin 'FEDERATED' is disabled.
2016-08-02 20:43:11 2888 [Note] InnoDB: Using atomics to ref count buffer pool pages
2016-08-02 20:43:11 2888 [Note] InnoDB: The InnoDB memory heap is disabled
```

La variabile `log_error` contiene il percorso del file di registro per la registrazione degli errori.

In assenza di una voce del file di configurazione per `log_error` , il sistema `log_error` i suoi valori su `@@hostname` .err nel `datadir` . Nota che `log_error` non è una variabile dinamica. Di conseguenza, le modifiche vengono apportate tramite modifiche al file cnf o ini e al riavvio del server (oppure visualizzando "Flushing e Renaming the Error Log File" nel collegamento Pagina manuale in basso qui).

La registrazione non può essere disabilitata per errori. Sono importanti per la salute del sistema durante la risoluzione dei problemi. Inoltre, le voci non sono frequenti rispetto al registro delle query generali.

La variabile GLOBAL `log_warnings` imposta il livello di verbosità che varia in base alla versione del server. Il seguente frammento illustra:

```
SELECT @@log_warnings; -- make a note of your prior setting
SET GLOBAL log_warnings=2; -- setting above 1 increases output (see server version)
```

`log_warnings` come visto sopra è una variabile dinamica.

Le modifiche del file di configurazione nei file `cnf` e `ini` potrebbero essere le seguenti.

```
[mysqld]
log_error        = /path/to/CurrentError.log
log_warnings     = 2
```

MySQL 5.7.2 ha espanso il livello di avviso verbosità a 3 e ha aggiunto GLOBAL `log_error_verbosity`

. Ancora una volta, è stato introdotto in 5.7.2. Può essere impostato dinamicamente e controllato come variabile o impostato tramite le impostazioni del file di configurazione `cnf` o `ini` .

A partire da MySQL 5.7.2:

```
[mysqld]
log_error           = /path/to/CurrentError.log
log_warnings        = 2
log_error_verbosity = 3
```

Si prega di consultare la pagina del manuale MySQL intitolata The Error Log, in particolare per Flushing e Renaming del file di log degli errori, e la sua sezione *Verbosità del registro degli errori* con le versioni relative a `log_warnings` e `error_log_verbosity` .

Leggi Log files online: https://riptutorial.com/it/mysql/topic/5102/log-files

Capitolo 37: Motore MyISAM

Osservazioni

Nel corso degli anni, InnoDB è migliorato al punto in cui è quasi sempre migliore di MyISAM, almeno le versioni attualmente supportate. Conclusione: non utilizzare MyISAM, tranne forse per le tabelle che sono veramente temporanee.

Resta un vantaggio di MyISAM rispetto a InnoDB: è 2x-3x più piccolo nello spazio richiesto sul disco.

Quando InnoDB è uscito, MyISAM era ancora un motore redditizio. Ma con l'avvento di XtraDB e 5.6, InnoDB è diventato "migliore" di MyISAM nella maggior parte dei benchmark.

Si dice che la prossima versione principale eliminerà la necessità di MyISAM realizzando tabelle InnoDB veramente temporanee e spostando le tabelle di sistema in InnoDB.

Examples

MOTORE = MyISAM

```
CREATE TABLE foo (
     ...
) ENGINE=MyISAM;
```

Leggi Motore MyISAM online: https://riptutorial.com/it/mysql/topic/4710/motore-myisam

Capitolo 38: MySQL LOCK TABLE

Sintassi

- LOCK TABLES nome_tabella [LEGGI | SCRIVI]; // Lock Table
- TABELLE SBLOCCA; // Sblocca tabelle

Osservazioni

Il blocco viene utilizzato per risolvere problemi di concorrenza. Il blocco è richiesto solo quando si esegue una transazione, che prima legge un valore da un database e successivamente lo scrive nel database. I blocchi non sono mai necessari per l'inserimento, l'aggiornamento o l'eliminazione di operazioni autonome.

Sono disponibili due tipi di serrature

LEGGI BLOCCO: quando un utente legge solo da una tabella.

WRITE LOCK - quando un utente sta eseguendo sia la lettura che la scrittura su un tavolo.

Quando un utente tiene un `WRITE LOCK` su una tabella, nessun altro utente può leggere o scrivere su quella tabella. Quando un utente dispone di un `READ LOCK` su un tavolo, altri utenti possono anche leggere o sospendere un `READ LOCK` , ma nessun utente può scrivere o mantenere un `WRITE LOCK` su tale tabella.

Se il motore di archiviazione predefinito è InnoDB, MySQL utilizza automaticamente il blocco a livello di riga in modo che più transazioni possano utilizzare la stessa tabella contemporaneamente per la lettura e la scrittura, senza aspettarsi reciprocamente.

Per tutti i motori di memorizzazione diversi da InnoDB, MySQL utilizza il blocco delle tabelle.

Per maggiori dettagli sul blocco del tavolo Vedi qui

Examples

Serrature Mysql

I blocchi tabella possono essere uno strumento importante per `ENGINE=MyISAM` , ma raramente sono utili per `ENGINE=InnoDB` . Se si è tentati di utilizzare blocchi di tabelle con InnoDB, è necessario riconsiderare il modo in cui si sta lavorando con le transazioni.

MySQL consente alle sessioni client di acquisire esplicitamente blocchi di tabelle allo scopo di collaborare con altre sessioni per l'accesso alle tabelle o di impedire ad altre sessioni di modificare le tabelle durante i periodi in cui una sessione richiede l'accesso esclusivo a tali sessioni. Una sessione può acquisire o rilasciare i blocchi solo per se stesso. Una sessione non può acquisire

blocchi per un'altra sessione o rilasciare blocchi bloccati da un'altra sessione.

I blocchi possono essere utilizzati per emulare transazioni o per ottenere maggiore velocità durante l'aggiornamento delle tabelle. Questo è spiegato più in dettaglio più avanti in questa sezione.

Comando: `LOCK TABLES table_name READ|WRITE;`

è possibile assegnare solo il tipo di blocco a una singola tabella;

Esempio (READ LOCK):

```
LOCK TABLES table_name READ;
```

Esempio (WRITE LOCK):

```
LOCK TABLES table_name WRITE;
```

Per vedere il blocco è applicato o meno, utilizzare il seguente comando

```
SHOW OPEN TABLES;
```

Per svuotare / rimuovere tutti i blocchi, utilizzare il seguente comando:

```
UNLOCK TABLES;
```

ESEMPIO:

```
LOCK TABLES products WRITE:
INSERT INTO products(id,product_name) SELECT id,old_product_name FROM old_products;
UNLOCK TABLES;
```

Sopra l'esempio qualsiasi connessione esterna non può scrivere alcun dato nella tabella prodotti fino al prodotto della tabella di sblocco

ESEMPIO:

```
LOCK TABLES products READ:
INSERT INTO products(id,product_name) SELECT id,old_product_name FROM old_products;
UNLOCK TABLES;
```

Sopra l'esempio qualsiasi connessione esterna non può leggere alcun dato dalla tabella prodotti fino al prodotto della tabella di sblocco

Blocco a livello di riga

Se le tabelle utilizzano InnoDB, MySQL utilizza automaticamente il blocco a livello di riga in modo che più transazioni possano utilizzare la stessa tabella contemporaneamente per la lettura e la scrittura, senza aspettarsi reciprocamente.

Se due transazioni cercano di modificare la stessa riga e entrambe utilizzano il blocco a livello di riga, una delle transazioni attende che l'altro venga completato.

Il blocco del livello di riga può anche essere ottenuto utilizzando `SELECT ... FOR UPDATE` per ogni riga che si prevede venga modificata.

Prendi in considerazione due connessioni per spiegare il blocco del livello di riga in dettaglio

Connessione 1

```
START TRANSACTION;
SELECT ledgerAmount FROM accDetails WHERE id = 1 FOR UPDATE;
```

Nella connessione 1, il blocco del livello di riga ottenuto `SELECT ... FOR UPDATE` .

Connessione 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1;
```

Quando qualcuno tenta di aggiornare la stessa riga nella connessione 2, attenderà che la connessione 1 termini la transazione o che venga visualizzato un messaggio di errore in base all'impostazione `innodb_lock_wait_timeout` , che per impostazione predefinita è 50 secondi.

```
Error Code: 1205. Lock wait timeout exceeded; try restarting transaction
```

Per visualizzare i dettagli su questo blocco, eseguire `SHOW ENGINE INNODB STATUS`

```
---TRANSACTION 1973004, ACTIVE 7 sec updating
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 4, OS thread handle 0x7f996beac700, query id 30 localhost root update
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1
------- TRX HAS BEEN WAITING 7 SEC FOR THIS LOCK TO BE GRANTED:
```

Connessione 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 250 WHERE id=2;
1 row(s) affected
```

Ma mentre l'aggiornamento di un'altra riga nella connessione 2 verrà eseguita senza alcun errore.

Connessione 1

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 750 WHERE id=1;
COMMIT;
1 row(s) affected
```

Ora il blocco riga viene rilasciato, perché la transazione è impegnata in Connection 1.

Connessione 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1;
1 row(s) affected
```

L'aggiornamento viene eseguito senza errori in Connection 2 dopo che Connection 1 ha rilasciato il blocco riga terminando la transazione.

Leggi MySQL LOCK TABLE online: https://riptutorial.com/it/mysql/topic/5233/mysql-lock-table

Capitolo 39: mysqlimport

Parametri

Parametro	Descrizione
`--delete -D`	svuota la tabella prima di importare il file di testo
`--fields-optionally-enclosed-by`	definire il carattere che cita i campi
`--fields-terminated-by`	terminatore di campo
`--ignore -i`	ignora la riga ingerita in caso di chiavi duplicate
`--lines-terminated-by`	definire il terminatore di riga
`--password -p`	parola d'ordine
`--port -P`	porta
`--replace -r`	sovrascrivere la vecchia riga di immissione in caso di chiavi duplicate
`--user -u`	nome utente
`--where -w`	specificare una condizione

Osservazioni

`mysqlimport` utilizzerà il nome del file importato, dopo aver rimosso l'estensione, per determinare la tabella di destinazione.

Examples

Utilizzo di base

Dato il file `employee.txt` tablatura `employee.txt`

1 `\t` Arthur Dent
2 `\t` Marvin
3 `\t` Zaphod Beeblebrox

```
$ mysql --user=user --password=password mycompany -e 'CREATE TABLE employee(id INT, name
VARCHAR(100), PRIMARY KEY (id))'
```

```
$ mysqlimport --user=user --password=password mycompany employee.txt
```

Utilizzando un delimitatore di campo personalizzato

Dato il file di testo employee.txt

1 | Arthur Dent
2 | Marvin
3 | Zaphod Beeblebrox

```
$ mysqlimport --fields-terminated-by='|' mycompany employee.txt
```

Utilizzo di un delimitatore di riga personalizzato

Questo esempio è utile per le desinenze di tipo windows:

```
$ mysqlimport --lines-terminated-by='\r\n' mycompany employee.txt
```

Gestire chiavi duplicate

Dato il tavolo `Employee`

id	Nome
3	Yooden Vranx

E il file `employee.txt`

1 `\t` Arthur Dent
2 `\t` Marvin
3 `\t` Zaphod Beeblebrox

L'opzione `--ignore` ignorerà la voce su chiavi duplicate

```
$ mysqlimport --ignore mycompany employee.txt
```

id	Nome
1	Arthur Dent
2	Marvin
3	Yooden Vranx

L'opzione `--replace` sovrascrive la vecchia voce

```
$ mysqlimport --replace mycompany employee.txt
```

id	Nome
1	Arthur Dent
2	Marvin
3	Zaphod Beeblebrox

Importazione condizionale

```
$ mysqlimport --where="id>2" mycompany employee.txt
```

Importa un csv standard

```
$ mysqlimport
    --fields-optionally-enclosed-by='"'
    --fields-terminated-by=,
    --lines-terminated-by="\r\n"
    mycompany employee.csv
```

Leggi mysqlimport online: https://riptutorial.com/it/mysql/topic/5215/mysqlimport

Capitolo 40: NULLO

Examples

Utilizza per NULL

- Dati non ancora noti - come `end_date` , `rating`
- Dati facoltativi, ad esempio `middle_initial` (anche se potrebbe essere preferibile come stringa vuota)
- 0/0 - Il risultato di alcuni calcoli, come zero diviso per zero.
- NULL non è uguale a "" (stringa vuota) o 0 (in caso di numero intero).
- altri?

Test NULL

- `IS NULL` / `IS NOT NULL` - `= NULL` non funziona come previsto.
- `x <=> y` è un confronto "nullo-sicuro".

In un `LEFT JOIN` test per righe di `a` per cui *non* v'è una riga corrispondente nella `b` .

```
SELECT ...
    FROM a
    LEFT JOIN b ON ...
    WHERE b.id IS NULL
```

Leggi NULLO online: https://riptutorial.com/it/mysql/topic/6757/nullo

Capitolo 41: Operazioni di data e ora

Examples

Adesso()

```
Select Now();
```

Mostra la data e l'ora del server corrente.

```
Update `footable` set mydatefield = Now();
```

Questo aggiornerà il campo `mydatefield` con la data e l'ora del server corrente nel fuso orario configurato del server, ad es

```
'2016-07-21 12:00:00'
```

Data aritmetica

```
NOW() + INTERVAL 1 DAY  -- This time tomorrow

CURDATE() - INTERVAL 4 DAY -- Midnight 4 mornings ago
```

Mostra le domande mysql memorizzate che sono state poste da 3 a 10 ore fa (da 180 a 600 minuti fa):

```
SELECT qId,askDate,minuteDiff
FROM
(   SELECT qId,askDate,
    TIMESTAMPDIFF(MINUTE,askDate,now()) as minuteDiff
    FROM questions_mysql
) xDerived
WHERE minuteDiff BETWEEN 180 AND 600
ORDER BY qId DESC
LIMIT 50;

+----------+---------------------+------------+
| qId      | askDate             | minuteDiff |
+----------+---------------------+------------+
| 38546828 | 2016-07-23 22:06:50 |        182 |
| 38546733 | 2016-07-23 21:53:26 |        195 |
| 38546707 | 2016-07-23 21:48:46 |        200 |
| 38546687 | 2016-07-23 21:45:26 |        203 |
| ...      |                     |            |
+----------+---------------------+------------+
```

Pagine di manuale MySQL per `TIMESTAMPDIFF()` .

Attenzione Non cercare di usare espressioni come `CURDATE() + 1` per l'aritmetica della data in

MySQL. Non restituiscono ciò che ti aspetti, soprattutto se sei abituato al prodotto del database Oracle. Utilizzare invece `CURDATE() + INTERVAL 1 DAY` .

Test contro un intervallo di date

Anche se è molto allettante usare `BETWEEN` ... `AND` ... per un intervallo di date, è problematico. Invece, questo schema evita la maggior parte dei problemi:

```
WHERE x >= '2016-02-25'
  AND x  < '2016-02-25' + INTERVAL 5 DAY
```

vantaggi:

- `BETWEEN` è "incluso" quindi includendo la data finale o il secondo.
- `23:59:59` è maldestro e sbagliato se hai una risoluzione di microsecondi su `DATETIME` .
- Questo modello evita di gestire gli anni bisestili e altri calcoli di dati.
- Funziona se `x` è `DATE` , `DATETIME` o `TIMESTAMP` .

SYSDATE (), NOW (), CURDATE ()

```
  SELECT SYSDATE();
```

Questa funzione restituisce la data e l'ora correnti come valore nel formato `'YYYY-MM-DD HH:MM:SS'` o `YYYYMMDDHHMMSS` , a seconda che la funzione venga utilizzata in una stringa o in un contesto numerico. Restituisce la data e l'ora nel fuso orario corrente.

```
  SELECT NOW();
```

Questa funzione è un sinonimo di `SYSDATE()` .

```
  SELECT CURDATE();
```

Questa funzione restituisce la data corrente, senza alcun orario, come valore nel formato `'YYYY-MM-DD'` o `YYYYMMDD` , a seconda che la funzione venga utilizzata in una stringa o in un contesto numerico. Restituisce la data nel fuso orario corrente.

Data di estrazione dalla data data o dall'espressione data / ora

```
SELECT DATE('2003-12-31 01:02:03');
```

L'output sarà:

```
2003-12-31
```

Utilizzando un indice per una ricerca di data e ora

Molte tabelle di database reali hanno molte righe con valori di colonna `DATETIME` OR `TIMESTAMP` che coprono molto tempo, inclusi anni o addirittura decenni. Spesso è necessario utilizzare una clausola `WHERE` per recuperare alcuni sottoinsiemi di quel periodo di tempo. Ad esempio, potremmo voler recuperare le righe per la data 1 settembre 2016 da una tabella.

Un modo inefficiente per farlo è questo:

```
WHERE DATE(x) = '2016-09-01'   /* slow! */
```

È inefficiente perché applica una funzione, `DATE()` , ai valori di una colonna. Ciò significa che MySQL deve esaminare ciascun valore di `x` e non è possibile utilizzare un indice.

Questo è un modo migliore di fare l'operazione

```
WHERE x >= '2016-09-01'
  AND x <  '2016-09-01' + INTERVAL 1 DAY
```

Questo seleziona un intervallo di valori di `x` giace in qualsiasi punto del giorno in questione, fino a ma *non compreso* (quindi `<`) a mezzanotte del giorno successivo.

Se la tabella ha un indice sulla colonna `x` , il server di database può eseguire una scansione di intervallo sull'indice. Ciò significa che può trovare rapidamente il primo valore rilevante di x, quindi eseguire la scansione dell'indice in modo sequenziale finché non trova l'ultimo valore pertinente. Una scansione dell'intervallo dell'indice è molto più efficiente della scansione completa della tabella richiesta da `DATE(x) = '2016-09-01` .

Non essere tentato di usarlo, anche se sembra più efficiente.

```
 WHERE x BETWEEN '2016-09-01' AND '2016-09-01' + INTERVAL 1 DAY  /*  wrong! */
```

Ha la stessa efficienza della scansione dell'intervallo, ma selezionerà le righe con valori di `x` scendono esattamente a mezzanotte del 2-Sept-2016, che non è quello che vuoi.

Leggi Operazioni di data e ora online: https://riptutorial.com/it/mysql/topic/1882/operazioni-di-data-e-ora

Capitolo 42: Operazioni di stringa

Parametri

Nome	Descrizione
ASCII ()	Restituisce il valore numerico del carattere più a sinistra
BIDONE()	Restituisce una stringa contenente la rappresentazione binaria di un numero
BIT_LENGTH ()	Restituisce la lunghezza dell'argomento in bit
CHAR ()	Restituisce il carattere per ogni intero passato
CHAR_LENGTH ()	Restituisce il numero di caratteri in argomento
Character_length ()	Sinonimo di CHAR_LENGTH ()
CONCAT ()	Restituisci stringa concatenata
CONCAT_WS ()	Ritorno concatenato con separatore
ELT ()	Restituisce la stringa al numero di indice
EXPORT_SET ()	Restituisce una stringa in modo tale che per ogni bit impostato nei bit del valore, si ottiene una stringa e per ogni bit non impostato si ottiene una stringa off
CAMPO()	Restituisce l'indice (posizione) del primo argomento negli argomenti successivi
FIND_IN_SET ()	Restituisce la posizione dell'indice del primo argomento all'interno del secondo argomento
FORMATO()	Restituisce un numero formattato al numero specificato di posizioni decimali
FROM_BASE64 ()	Decodifica su una stringa base 64 e restituisce il risultato
ESADECIMALE()	Restituisce una rappresentazione esadecimale di un valore decimale o stringa
INSERIRE()	Inserire una sottostringa nella posizione specificata fino al numero di caratteri specificato
INSTR ()	Restituisce l'indice della prima occorrenza della sottostringa

Nome	Descrizione
LCASE ()	Sinonimo di LOWER ()
SINISTRA()	Restituisce il numero più a sinistra dei caratteri come specificato
LUNGHEZZA()	Restituisce la lunghezza di una stringa in byte
PIACE	Semplice abbinamento di motivi
LOAD_FILE ()	Carica il file indicato
INDIVIDUARE()	Restituisce la posizione della prima occorrenza della sottostringa
INFERIORE()	Restituisce l'argomento in minuscolo
LPAD ()	Restituisce l'argomento stringa, a sinistra con la stringa specificata
LTRIM ()	Rimuovi gli spazi iniziali
MAKE_SET ()	Restituisce un insieme di stringhe separate da virgola che hanno il bit corrispondente in bit impostati
INCONTRO	Eseguire la ricerca full-text
MID ()	Restituisce una sottostringa a partire dalla posizione specificata
NON COME	Negazione della corrispondenza semplice del modello
NON REGEXP	Negazione di REGEXP
OCT ()	Restituisce una stringa contenente la rappresentazione ottale di un numero
OCTET_LENGTH ()	Sinonimo di LENGTH ()
ORD ()	Restituisce il codice del carattere per il carattere all'estrema sinistra dell'argomento
POSIZIONE()	Sinonimo di LOCATE ()
CITAZIONE()	Sfuggi all'argomento da usare in un'istruzione SQL
REGEXP	Corrispondenza dei modelli usando le espressioni regolari
RIPETERE()	Ripeti una stringa il numero specificato di volte
SOSTITUIRE()	Sostituisci le occorrenze di una stringa specificata
INVERSO()	Invertire i caratteri in una stringa
DESTRA()	Restituisce il numero più a destra specificato di caratteri

Nome	Descrizione
RLIKE	Sinonimo di REGEXP
RPAD ()	Aggiungi stringa il numero specificato di volte
RTRIM ()	Rimuovi gli spazi finali
SOUNDEX ()	Restituisce una stringa soundex
SUONO MI PIACE	Confronta i suoni
SPAZIO()	Restituisce una stringa del numero specificato di spazi
STRCMP ()	Confronta due stringhe
SUBSTR ()	Restituisce la sottostringa come specificato
SUBSTRING ()	Restituisce la sottostringa come specificato
SUBSTRING_INDEX ()	Restituisce una sottostringa da una stringa prima del numero specificato di occorrenze del delimitatore
TO_BASE64 ()	Restituisce l'argomento convertito in una stringa base 64
TRIM ()	Rimuovi gli spazi iniziali e finali
UCASE ()	Sinonimo di UPPER ()
UNHEX ()	Restituisce una stringa contenente la rappresentazione esadecimale di un numero
SUPERIORE()	Converti in maiuscolo
WEIGHT_STRING ()	Restituisce la stringa di peso per una stringa

Examples

Trova elemento in elenco separato da virgole

```
SELECT FIND_IN_SET('b','a,b,c');
```

Valore di ritorno:

2

```
SELECT FIND_IN_SET('d','a,b,c');
```

Valore di ritorno:

0

STR_TO_DATE - Converti la stringa in data

Con una colonna di uno dei tipi di stringa, denominato `my_date_field` con un valore come [la stringa] `07/25/2016` , la seguente dichiarazione dimostra l'uso della funzione `STR_TO_DATE` :

```
SELECT STR_TO_DATE(my_date_field, '%m/%d/%Y') FROM my_table;
```

È possibile utilizzare questa funzione anche come parte della clausola `WHERE` .

LOWER () / LCASE ()

Convertire in minuscolo l'argomento della stringa

Sintassi: LOWER (str)

```
LOWER('fOoBar') -- 'foobar'
LCASE('fOoBar') -- 'foobar'
```

SOSTITUIRE()

Convertire in minuscolo l'argomento della stringa

Sintassi: REPLACE (str, from_str, to_str)

```
REPLACE('foobarbaz', 'bar', 'BAR') -- 'fooBARbaz'
REPLACE('foobarbaz', 'zzz', 'ZZZ') -- 'foobarbaz'
```

SUBSTRING ()

SUBSTRING (o equivalente: SUBSTR) restituisce la sottostringa a partire dalla posizione specificata e, facoltativamente, con la lunghezza specificata

Sintassi: `SUBSTRING(str, start_position)`

```
SELECT SUBSTRING('foobarbaz', 4); -- 'barbaz'
SELECT SUBSTRING('foobarbaz' FROM 4); -- 'barbaz'

-- using negative indexing
SELECT SUBSTRING('foobarbaz', -6); -- 'barbaz'
SELECT SUBSTRING('foobarbaz' FROM -6); -- 'barbaz'
```

Sintassi: `SUBSTRING(str, start_position, length)`

```
SELECT SUBSTRING('foobarbaz', 4, 3); -- 'bar'
SELECT SUBSTRING('foobarbaz', FROM 4 FOR 3); -- 'bar'

-- using negative indexing
```

```
SELECT SUBSTRING('foobarbaz', -6, 3); -- 'bar'
SELECT SUBSTRING('foobarbaz' FROM -6 FOR 3); -- 'bar'
```

UPPER () / UCASE ()

Convertire in maiuscolo l'argomento della stringa

Sintassi: UPPER (str)

```
UPPER('fOoBar') -- 'FOOBAR'
UCASE('fOoBar') -- 'FOOBAR'
```

LUNGHEZZA()

Restituisce la lunghezza della stringa in byte. Poiché alcuni caratteri possono essere codificati usando più di un byte, se vuoi che la lunghezza dei caratteri veda CHAR_LENGTH ()

Sintassi: LENGTH (str)

```
LENGTH('foobar') -- 6
LENGTH('fööbar') -- 8 -- contrast with CHAR_LENGTH(...) = 6
```

CHAR_LENGTH ()

Restituisce il numero di caratteri nella stringa

Sintassi: CHAR_LENGTH (str)

```
CHAR_LENGTH('foobar') -- 6
CHAR_LENGTH('fööbar') -- 6 -- contrast with LENGTH(...) = 8
```

HEX (str)

Convertire l'argomento in esadecimale. Questo è usato per le stringhe.

```
HEX('fööbar') -- 66F6F6626172 -- in "CHARACTER SET latin1" because "F6" is hex for ö
HEX('fööbar') -- 66C3B6C3B6626172  -- in "CHARACTER SET utf8 or utf8mb4" because "C3B6" is hex
for ö
```

Leggi Operazioni di stringa online: https://riptutorial.com/it/mysql/topic/1399/operazioni-di-stringa

Capitolo 43: ORDINATO DA

Examples

contesti

Le clausole in un `SELECT` hanno un ordine specifico:

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...
    ORDER BY ...  -- goes here
    LIMIT ... OFFSET ...;

( SELECT ... ) UNION ( SELECT ... ) ORDER BY ...  -- for ordering the result of the UNION.

SELECT ... GROUP_CONCAT(DISTINCT x ORDER BY ... SEPARATOR ...) ...

ALTER TABLE ... ORDER BY ... -- probably useful only for MyISAM; not for InnoDB
```

Di base

ORDINA PER x

`x` può essere qualsiasi tipo di dati.

- `NULL`s precedono i non NULL.
- Il valore predefinito è `ASC` (dal più basso al più alto)
- Le stringhe (`VARCHAR` , ecc.) Sono ordinate secondo la `COLLATION` della dichiarazione
- `ENUM`s sono ordinati in base all'ordine di dichiarazione delle stringhe.

ASCending / DESCending

```
ORDER BY x ASC  -- same as default
ORDER BY x DESC  -- highest to lowest
ORDER BY lastname, firstname  -- typical name sorting; using two columns
ORDER BY submit_date DESC  -- latest first
ORDER BY submit_date DESC, id ASC  -- latest first, but fully specifying order.
```

- `ASC` = `ASCENDING` , `DESC` = `DESCENDING`
- `NULL`s vengono prima anche per `DESC` .
- Negli esempi precedenti, `INDEX(x)` , `INDEX(lastname, firstname)` , `INDEX(submit_date)` possono migliorare significativamente le prestazioni.

Ma ... Il mixaggio di `ASC` e `DESC` , come nell'ultimo esempio, non può utilizzare un indice composito a vantaggio. Né `INDEX(submit_date DESC, id ASC)` help - " `DESC` " è riconosciuto sintatticamente nella dichiarazione `INDEX` , ma ignorato.

Alcuni trucchi

```
ORDER BY FIND_IN_SET(card_type, "MASTER-CARD,VISA,DISCOVER") -- sort 'MASTER-CARD' first.
ORDER BY x IS NULL, x  -- order by `x`, but put `NULLs` last.
```

Ordinazione personalizzata

```
SELECT * FROM some_table WHERE id IN (118, 17, 113, 23, 72)
ORDER BY FIELD(id, 118, 17, 113, 23, 72);
```

Restituisce il risultato nell'ordine di id specificato.

id	...
118	...
17	...
113	...
23	...
72	...

Utile se gli ID sono già ordinati e devi solo recuperare le righe.

Leggi ORDINATO DA online: https://riptutorial.com/it/mysql/topic/5469/ordinato-da

Capitolo 44: Ottimizzazione delle prestazioni

Sintassi

- Non utilizzare DISTINCT e GROUP BY nella stessa SELECT.
- Non impaginare tramite OFFSET, "ricorda dove hai lasciato".
- WHERE (a, b) = (22,33) non ottimizza affatto.
- Esplicitamente dire TUTTO o DISTINTIVO dopo UNION - ti ricorda di scegliere tra il più veloce ALL o il più lento DISTINCT.
- Non usare SELECT *, specialmente se hai colonne TEXT o BLOB che non ti servono. C'è sovraccarico nelle tabelle e nella trasmissione del tmp.
- È più veloce quando GROUP BY e ORDER BY possono avere esattamente la stessa lista.
- Non utilizzare FORCE INDEX; potrebbe essere d'aiuto oggi, ma probabilmente farà male domani.

Osservazioni

Vedi anche le discussioni su ORDER BY, LIKE, REGEXP, ecc. Nota: questo richiede modifiche con link e altri argomenti.

Ricettario sulla costruzione di indici ottimali .

Examples

Aggiungi l'indice corretto

Questo è un argomento enorme, ma è anche il più importante problema di "rendimento".

La lezione principale per un principiante è quella di imparare gli indici "compositi". Ecco un rapido esempio:

```
INDEX(last_name, first_name)
```

è eccellente per questi:

```
WHERE last_name = '...'
WHERE first_name = '...' AND last_name = '...'   -- (order in WHERE does not matter)
```

ma non per

```
WHERE first_name = '...'   -- order in INDEX _does_ matter
WHERE last_name = '...' OR first_name = '...'   -- "OR" is a killer
```

Imposta la cache correttamente

`innodb_buffer_pool_size` dovrebbe essere circa il 70% della RAM disponibile.

Evitare costrutti inefficienti

```
x IN ( SELECT ... )
```

trasformarsi in un `JOIN`

Quando possibile, evita `OR` .

Non "nascondere" una colonna indicizzata in una funzione, ad esempio `WHERE DATE(x) = ...` ; riformulare come `WHERE x = ...`

In generale, è possibile evitare `WHERE LCASE(name1) = LCASE(name2)` con un confronto appropriato.

Non utilizzare `OFFSET` per "paginazione", invece "ricorda dove hai lasciato".

Evita `SELECT * ...` (a meno che non si esegua il debug).

Nota a Maria Deleva, Barranka, Batsu: Questo è un segnaposto; ti preghiamo di rimuovere questi elementi mentre crei esempi completi. Dopo aver fatto quello che puoi, mi trasferirò per approfondire il resto e / o lanciarlo.

negativi

Ecco alcune cose che non sono in grado di aiutare le prestazioni. Derivano da informazioni non aggiornate e / o ingenuità.

- InnoDB è migliorato al punto in cui è improbabile che MyISAM sia migliore.
- `PARTITIONing` offre raramente vantaggi prestazionali; può anche danneggiare le prestazioni.
- L'impostazione di `query_cache_size` superiore a 100M di solito *danneggia le* prestazioni.
- Aumentare molti valori in `my.cnf` può portare a "swapping", che è un *serio* problema di prestazioni.
- "Gli indici di prefisso" (come `INDEX(foo(20))`) sono generalmente inutili.
- `OPTIMIZE TABLE` è quasi sempre inutile. (E comporta il blocco del tavolo.)

Avere un INDICE

La cosa più importante per velocizzare una query su qualsiasi tabella non piccola è avere un indice adatto.

```
WHERE a = 12  --> INDEX(a)
WHERE a > 12  --> INDEX(a)
```

```
WHERE a = 12 AND b > 78  --> INDEX(a,b) is more useful than INDEX(b,a)
WHERE a > 12 AND b > 78  --> INDEX(a) or INDEX(b); no way to handle both ranges

ORDER BY x  --> INDEX(x)
ORDER BY x, y  --> INDEX(x,y) in that order
ORDER BY x DESC, y ASC  --> No index helps - because of mixing ASC and DESC
```

Non nasconderti nella funzione

Un errore comune è quello di nascondere una colonna indicizzata all'interno di una chiamata di funzione. Ad esempio, questo non può essere aiutato da un indice:

```
WHERE DATE(dt) = '2000-01-01'
```

Invece, dato `INDEX(dt)` questi possono usare l'indice:

```
WHERE dt = '2000-01-01'  -- if `dt` is datatype `DATE`
```

Funziona per `DATE` , `DATETIME` , `TIMESTAMP` e anche `DATETIME(6)` (microsecondi):

```
WHERE dt >= '2000-01-01'
  AND dt  < '2000-01-01' + INTERVAL 1 DAY
```

O

In generale `OR` uccide l'ottimizzazione.

```
WHERE a = 12 OR b = 78
```

non può usare `INDEX(a,b)` , e può o non può usare `INDEX(a), INDEX(b)` tramite "indice unione". La fusione indice è meglio di niente, ma solo a malapena.

```
WHERE x = 3 OR x = 5
```

è trasformato in

```
WHERE x IN (3, 5)
```

che *può* usare un indice con `x` in esso.

subquery

Le sottoquery hanno diversi gusti e hanno un potenziale di ottimizzazione diverso. Innanzitutto, nota che le sottoquery possono essere "correlate" o "non correlate". Correlati significa che dipendono da un valore esterno alla subquery. Ciò implica in genere che la sottoquery *deve* essere rivalutata per ciascun valore esterno.

Questa subquery correlata è spesso piuttosto buona. Nota: deve restituire al massimo 1 valore. È spesso utile come alternativa a, anche se non necessariamente più veloce, un `LEFT JOIN` .

```
SELECT a, b, ( SELECT ... FROM t WHERE t.x = u.x ) AS c
    FROM u ...
SELECT a, b, ( SELECT MAX(x) ... ) AS c
    FROM u ...
SELECT a, b, ( SELECT x FROM t ORDER BY ... LIMIT 1 ) AS c
    FROM u ...
```

Questo di solito non è correlato:

```
SELECT ...
    FROM ( SELECT ... ) AS a
    JOIN b ON ...
```

Note su `FROM-SELECT` :

- Se restituisce 1 riga, ottimo.
- Un buon paradigma (di nuovo "1 riga") è per la sottoquery `( SELECT @n := 0 )` , inizializzando così una variabile @ per l'uso nel resto o nella query.
- Se restituisce molte righe *e* anche `JOIN` è `( SELECT ... )` con molte righe, l'efficienza può essere terribile. Pre-5.6, non c'era alcun indice, quindi è diventato un `CROSS JOIN` ; 5.6+ comporta la deduzione del miglior indice sulle tabelle temporanee e quindi la generazione, solo per buttarlo via una volta terminato con `SELECT` .

ISCRIVITI + GRUPPO DI

Un problema comune che porta a una query inefficiente è simile a questo:

```
SELECT ...
    FROM a
    JOIN b  ON ...
    WHERE ...
    GROUP BY a.id
```

Innanzitutto, il `JOIN` espande il numero di righe; quindi `GROUP BY` ridimensiona il numero di righe in `a` .

Non ci possono essere buone scelte per risolvere questo problema implode-implode. Una possibile opzione è trasformare il `JOIN` in una subquery correlata in `SELECT` . Questo elimina anche `GROUP BY` .

Leggi Ottimizzazione delle prestazioni online:
https://riptutorial.com/it/mysql/topic/4292/ottimizzazione-delle-prestazioni

Capitolo 45: Parole riservate

introduzione

MySQL ha alcuni nomi speciali chiamati *parole riservate* . Una parola riservata può essere usata come identificatore di una tabella, colonna, ecc. Solo se è racchiusa tra apici inversi (`), altrimenti causerà un errore.

Per evitare tali errori, non utilizzare parole riservate come identificatori o avvolgere l'identificatore offendente nei backtick.

Osservazioni

Di seguito sono elencate tutte le parole riservate (dalla documentazione ufficiale):

- ACCESSIBILE
- INSERISCI
- TUTTI
- ALTER
- ANALIZZARE
- E
- COME
- ASC
- ASENSITIVE
- PRIMA
- FRA
- BIGINT
- BINARIO
- BLOB
- TUTTI E DUE
- DI
- CHIAMATA
- CASCATA
- ASTUCCIO
- MODIFICARE
- CHAR
- PERSONAGGIO
- DAI UN'OCCHIATA
- FASCICOLARE
- COLONNA
- CONDIZIONE
- VINCOLO
- CONTINUA
- CONVERTIRE
- CREARE

- ATTRAVERSARE
- DATA ODIERNA
- ORA ATTUALE
- CURRENT_TIMESTAMP
- UTENTE CORRENTE
- CURSORE
- BANCA DATI
- BANCHE DATI
- DAY_HOUR
- DAY_MICROSECOND
- DAY_MINUTE
- DAY_SECOND
- dicembre
- DECIMALE
- DICHIARARE
- PREDEFINITO
- RITARDATO
- ELIMINA
- DESC
- DESCRIVERE
- DETERMINISTICO
- DISTINCT
- DISTINCTROW
- DIV
- DOPPIO
- FAR CADERE
- DUAL
- OGNI
- ALTRO
- ELSEIF
- CHIUSO
- SFUGGITO
- ESISTE
- USCITA
- SPIEGARE
- FALSE
- FETCH
- GALLEGGIANTE
- float4
- float8
- PER
- VIGORE
- STRANIERO
- A PARTIRE DAL
- TESTO INTERO
- GENERATO

- OTTENERE
- CONCEDERE
- GRUPPO
- VISTA
- PRIORITÀ ALTA
- HOUR_MICROSECOND
- HOUR_MINUTE
- HOUR_SECOND
- SE
- IGNORARE
- NEL
- INDICE
- INFILE
- INTERNO
- DENTRO FUORI
- INSENSIBILE
- INSERIRE
- INT
- INT1
- INT2
- INT3
- INT4
- INT8
- NUMERO INTERO
- INTERVALLO
- IN
- IO_AFTER_GTIDS
- IO_BEFORE_GTIDS
- È
- ITERATE
- ADERIRE
- CHIAVE
- CHIAVI
- UCCIDERE
- LEADER
- PARTIRE
- SINISTRA
- PIACE
- LIMITE
- LINEARE
- LINEE
- CARICARE
- ORA LOCALE
- LOCALTIMESTAMP
- SERRATURA
- LUNGO

- LONGBLOB
- LONGTEXT
- CICLO CONTINUO
- BASSA PRIORITÀ
- MASTER_BIND
- MASTER_SSL_VERIFY_SERVER_CERT
- INCONTRO
- MAXVALUE
- MEDIUMBLOB
- MEDIUMINT
- MEDIUMTEXT
- MIDDLEINT
- MINUTE_MICROSECOND
- MINUTE_SECOND
- MOD
- MODIFIES
- NATURALE
- NON
- NO_WRITE_TO_BINLOG
- NULLO
- NUMERICO
- SOPRA
- OTTIMIZZARE
- OPTIMIZER_COSTS
- OPZIONE
- FACOLTATIVAMENTE
- O
- ORDINE
- SU
- ESTERNO
- OUTFILE
- PARTIZIONE
- PRECISIONE
- PRIMARIO
- PROCEDURA
- EPURAZIONE
- GAMMA
- LEGGERE
- LEGGE
- LEGGERE SCRIVERE
- VERO
- RIFERIMENTI
- REGEXP
- PUBBLICAZIONE
- RINOMINARE
- RIPETERE

- SOSTITUIRE
- RICHIEDERE
- RESIGNAL
- LIMITARE
- RITORNO
- REVOCARE
- DESTRA
- RLIKE
- SCHEMA
- SCHEMI
- SECOND_MICROSECOND
- SELEZIONARE
- SENSIBILE
- SEPARATORE
- IMPOSTATO
- MOSTRARE
- SEGNALE
- SMALLINT
- SPAZIALE
- SPECIFICO
- SQL
- SQLEXCEPTION
- SQLSTATE
- SQLWARNING
- SQL_BIG_RESULT
- SQL_CALC_FOUND_ROWS
- SQL_SMALL_RESULT
- SSL
- DI PARTENZA
- IMMAGAZZINATO
- STRAIGHT_JOIN
- TAVOLO
- TERMINATED
- POI
- TINYBLOB
- TINYINT
- TINYTEXT
- A
- TRAILING
- TRIGGER
- VERO
- DISFARE
- UNIONE
- UNICO
- SBLOCCARE
- UNSIGNED

- AGGIORNARE
- USO
- USO
- UTILIZZO
- UTC_DATE
- UTC_TIME
- UTC_TIMESTAMP
- VALORI
- VARBINARY
- VARCHAR
- VARCHARACTER
- VARIANDO
- VIRTUALE
- QUANDO
- DOVE
- MENTRE
- CON
- SCRIVI
- XOR
- ANNO MESE
- ZEROFILL
- GENERATO
- OPTIMIZER_COSTS
- IMMAGAZZINATO
- VIRTUALE

Examples

Errori dovuti a parole riservate

Quando provi a selezionare da una tabella chiamata `order` come questo

```
select * from order
```

l'errore aumenta:

> Codice di errore: 1064. Si è verificato un errore nella sintassi SQL; controlla il manuale che corrisponde alla versione del tuo server MySQL per la sintassi corretta da usare vicino a 'ordine' alla riga 1

Le parole chiave riservate in MySQL devono essere sottoposte a escape con backtick (`)

```
select * from `order`
```

distinguere tra una parola chiave e un nome di tabella o colonna.

Vedi anche: Errore di sintassi dovuto all'utilizzo di una parola riservata come nome di tabella o

colonna in MySQL .

Leggi Parole riservate online: https://riptutorial.com/it/mysql/topic/1398/parole-riservate

Capitolo 46: partizionamento

Osservazioni

- **RANGE partizionamento** . Questo tipo di partizionamento assegna le righe alle partizioni in base ai valori delle colonne che rientrano in un determinato intervallo.
- **Partizione LIST** . Simile al partizionamento per RANGE, tranne per il fatto che la partizione è selezionata in base a colonne corrispondenti a uno di un set di valori discreti.
- **Partizionamento HASH** . Con questo tipo di partizionamento, viene selezionata una partizione in base al valore restituito da un'espressione definita dall'utente che opera sui valori delle colonne nelle righe da inserire nella tabella. La funzione può essere costituita da qualsiasi espressione valida in MySQL che restituisca un valore intero non negativo. È disponibile anche un'estensione per questo tipo, `LINEAR HASH` .
- **Partizionamento KEY** . Questo tipo di partizionamento è simile al partizionamento di HASH, tranne per il fatto che vengono fornite solo una o più colonne da valutare e il server MySQL fornisce la propria funzione di hashing. Queste colonne possono contenere valori diversi dai numeri interi, poiché la funzione di hashing fornita da MySQL garantisce un risultato intero indipendentemente dal tipo di dati della colonna. È disponibile anche un'estensione per questo tipo, `LINEAR KEY` .

Examples

RANGE Partizionamento

Una tabella partizionata per intervallo viene partizionata in modo tale che ciascuna partizione contenga righe per le quali il valore dell'espressione partizionamento si trova all'interno di un determinato intervallo. Gli intervalli devono essere contigui ma non sovrapposti e vengono definiti utilizzando l'operatore `VALUES LESS THAN` Per i prossimi esempi, si supponga di creare una tabella come la seguente per contenere record personali per una catena di 20 negozi video, numerati da 1 a 20:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

Questa tabella può essere suddivisa in vari modi, a seconda delle esigenze. Un modo sarebbe utilizzare la colonna `store_id` . Ad esempio, potresti decidere di suddividere la tabella in 4 modi aggiungendo una clausola `PARTITION BY RANGE` come mostrato qui:

```
ALTER TABLE employees PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

> `MAXVALUE` rappresenta un valore intero che è sempre maggiore del valore intero più grande possibile (in linguaggio matematico, funge da limite minimo).

basato sul documento ufficiale MySQL .

LISTA Partizionamento

Il partizionamento delle liste è simile al partizionamento della gamma in molti modi. Come nel partizionamento di RANGE, ogni partizione deve essere definita in modo esplicito. La principale differenza tra i due tipi di partizionamento è che, nel partizionamento delle liste, ogni partizione viene definita e selezionata in base all'appartenenza di un valore di colonna in uno di un insieme di liste valori, piuttosto che in uno di un insieme di intervalli contigui di valori. Questo viene fatto usando `PARTITION BY LIST(expr)` dove `expr` è un valore di colonna o un'espressione basata su un valore di colonna e restituendo un valore intero, e quindi definendo ogni partizione per mezzo di `VALUES IN (value_list)` , dove `value_list` è una elenco di numeri interi separati da virgola.

Per gli esempi che seguono, assumiamo che la definizione di base della tabella da partizionare sia fornita `CREATE TABLE` mostrata qui:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
);
```

Supponiamo che ci siano 20 negozi video distribuiti tra 4 franchising come mostrato nella seguente tabella.

Regione	Numeri ID negozio
Nord	3, 5, 6, 9, 17
est	1, 2, 10, 11, 19, 20
ovest	4, 12, 13, 14, 18
Centrale	7, 8, 15, 16

Per suddividere questa tabella in modo tale che le righe per i negozi appartenenti alla stessa

regione siano memorizzate nella stessa partizione

```
ALTER TABLE employees PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

basato sul documento ufficiale MySQL .

Partizionamento HASH

Il partizionamento di HASH viene utilizzato principalmente per garantire una distribuzione uniforme dei dati tra un numero predeterminato di partizioni. Con intervallo o elenco di partizioni, è necessario specificare esplicitamente in quale partizione deve essere memorizzato un determinato valore di colonna o un insieme di valori di colonna; con il partizionamento hash, MySQL si occupa di questo per te, e devi solo specificare un valore o un'espressione di colonna in base al valore di una colonna da sottoporre a hash e il numero di partizioni in cui deve essere divisa la tabella partizionata.

La seguente istruzione crea una tabella che utilizza l'hashing sulla colonna store_id ed è divisa in 4 partizioni:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

> Se non si include una clausola `PARTITIONS` , il numero di partizioni è impostato su 1.

basato sul documento ufficiale MySQL .

Leggi partizionamento online: https://riptutorial.com/it/mysql/topic/5128/partizionamento

Capitolo 47: Personalizza PS1

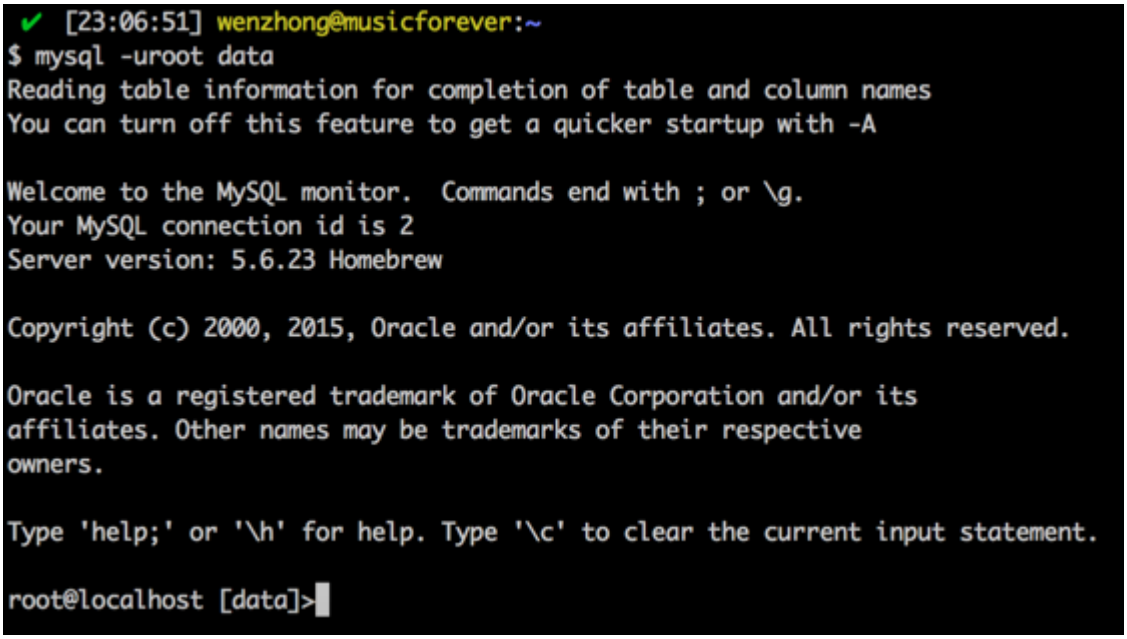
Examples

Personalizza MySQL PS1 con il database corrente

In .bashrc o .bash_profile, aggiungendo:

```
export MYSQL_PS1="\u@\h [\d]>"
```

fai in modo che il client MySQL PROMPT mostri l'utente corrente @ host [database].



PS1 personalizzato tramite file di configurazione MySQL

In `mysqld.cnf` o equivalente:

```
[mysql]
prompt = '\u@\h [\d]> '
```

Questo raggiunge un effetto simile, senza dover gestire `.bashrc` .

Leggi Personalizza PS1 online: https://riptutorial.com/it/mysql/topic/5795/personalizza-ps1

Capitolo 48: PREPARAZIONE delle dichiarazioni

Sintassi

- PREPARAZIONE stmt_name FROM preparable_stmt
- ESEGUI stmt_name [USING @var_name [, @var_name] ...]
- {DEALLOCATE | DROP} PREPARA stmt_name

Examples

PREPARARE, ESEGUIRE e DISALLOCARE le dichiarazioni PREPARA

PREPARE prepara una dichiarazione per l'esecuzione

EXECUTE esegue una dichiarazione preparata

DEALLOCATE PREPARE rilascia una dichiarazione preparata

```
SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
PREPARE stmt2 FROM @s;
SET @a = 6;
SET @b = 8;
EXECUTE stmt2 USING @a, @b;
```

Risultato:

```
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
```

Finalmente,

```
DEALLOCATE PREPARE stmt2;
```

Gli appunti:

- Devi usare @variables, non DECLAREd variabili per `FROM @s`
- Un uso primario di Prepara, ecc. È di "costruire" una query per situazioni in cui l'associazione non funzionerà, come l'inserimento del nome della tabella.

Costruisci ed esegui

(Questa è una richiesta per un buon esempio che mostra come *costruire* un `SELECT` usando `CONCAT` ,

quindi prepararlo + eseguirlo. Per favore enfatizza l'uso di @variables rispetto alle variabili DECLAREd: fa una grande differenza, ed è qualcosa che i novizi (includi me stesso) inciampare.)

Modificare la tabella con Aggiungi colonna

```
SET v_column_definition := CONCAT(
  v_column_name
  ,' ',v_column_type
  ,' ',v_column_options
);

SET @stmt := CONCAT('ALTER TABLE ADD COLUMN ', v_column_definition);

PREPARE stmt FROM @stmt;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

Leggi PREPARAZIONE delle dichiarazioni online:
https://riptutorial.com/it/mysql/topic/2603/preparazione-delle-dichiarazioni

Capitolo 49: Query pivot

Osservazioni

La creazione di query pivot in MySQL si basa sulla funzione `GROUP_CONCAT()` . Se il risultato dell'espressione che crea le colonne della query pivot dovrebbe essere grande, il valore della variabile `group_concat_max_len` deve essere aumentato:

```
set session group_concat_max_len = 1024 * 1024; -- This should be enough for most cases
```

Examples

Creazione di una query pivot

MySQL non fornisce un modo integrato per creare query pivot. Tuttavia, questi possono essere creati utilizzando istruzioni preparate.

Assumi la tabella `tbl_values` :

Id	Nome	Gruppo	Valore
1	Pete	UN	10
2	Pete	B	20
3	John	UN	10

Richiesta: creare una query che mostri la somma del `Value` per ciascun `Name` ; il `Group` deve essere intestazione di colonna e `Name` deve essere l'intestazione di riga.

```
-- 1. Create an expression that builds the columns
set @sql = (
    select group_concat(distinct
        concat(
            "sum(case when `Group`='", Group, "' then `Value` end) as `", `Group`, "`"
        )
    )
    from tbl_values
);

-- 2. Complete the SQL instruction
set @sql = concat("select Name, ", @sql, " from tbl_values group by `Name`");

-- 3. Create a prepared statement
prepare stmt from @sql;

-- 4. Execute the prepared statement
execute stmt;
```

Risultato:

Nome	UN	B
John	10	NULLO
Pete	10	20

Importante: deallocare l'istruzione preparata quando non è più necessaria:

```
deallocate prepare stmt;
```

Esempio su SQL Fiddle

Leggi Query pivot online: https://riptutorial.com/it/mysql/topic/3074/query-pivot

Capitolo 50: Raggruppa per

Sintassi

1. SELEZIONA espressione1, espressione2, ... espressione_n,
2. aggregate_function (espressione)
3. DA tabelle
4. [WHERE condizioni]
5. GROUP BY espressione1, espressione2, ... espressione_n;

Parametri

Parametro	DETTAGLI
espressione1, espressione2, ... espressione_n	Le espressioni che non sono incapsulate in una funzione di aggregazione e devono essere incluse nella clausola GROUP BY.
aggregate_function	Una funzione come SUM, COUNT, MIN, MAX o AVG.
tavoli	tavoli da cui desideri recuperare i record. Ci deve essere almeno una tabella elencata nella clausola FROM.
Dove condizioni	Opzionale. Le condizioni che devono essere soddisfatte per i record da selezionare.

Osservazioni

La clausola GROUP BY di MySQL viene utilizzata in un'istruzione SELECT per raccogliere dati su più record e raggruppare i risultati di una o più colonne.

Il suo comportamento è governato in parte dal valore della variabile `ONLY_FULL_GROUP_BY` . Quando è abilitato, le istruzioni `SELECT` che raggruppano per qualsiasi colonna non presente nell'output restituiscono un errore. (Questa è l'impostazione predefinita 5.7.5 .) Entrambe le impostazioni e non l'impostazione di questa variabile possono causare problemi agli utenti ingenui o agli utenti abituati ad altri DBMS.

Examples

GRUPPO UTILIZZANDO la funzione SOMMA

```
SELECT product, SUM(quantity) AS "Total quantity"
FROM order_details
GROUP BY product;
```

Raggruppa usando la funzione MIN

Assumere una tabella di dipendenti in cui ogni riga è un dipendente che ha un `name` , un `department` e uno `salary` .

```
SELECT department, MIN(salary) AS "Lowest salary"
FROM employees
GROUP BY department;
```

Questo ti direbbe quale dipartimento contiene il dipendente con lo stipendio più basso e quale è lo stipendio. Trovare il `name` del dipendente con lo stipendio più basso in ogni dipartimento è un problema diverso, oltre lo scopo di questo esempio. Vedi "groupwise max".

GRUPPO UTILIZZANDO COUNT Funzione

```
SELECT department, COUNT(*) AS "Man_Power"
FROM employees
GROUP BY department;
```

GROUP BY utilizzando HAVING

```
SELECT department, COUNT(*) AS "Man_Power"
FROM employees
GROUP BY department
HAVING COUNT(*) >= 10;
```

L'utilizzo di `GROUP BY ... HAVING` per filtrare i record aggregati è analogo all'uso di `SELECT ... WHERE` per filtrare i singoli record.

Potresti anche dire `HAVING Man_Power >= 10` dal momento che `HAVING` comprende "alias".

Gruppo utilizzando Group Concat

Group Concat viene utilizzato in MySQL per ottenere valori concatenati di espressioni con più di un risultato per colonna. Significato, ci sono molte righe da selezionare indietro per una colonna come `Name(1):Score(*)`

Nome	Punto
Adamo	A +
Adamo	UN-
Adamo	B
Adamo	C +
Conto	D-

Nome	Punto
John	UN-

```
SELECT Name, GROUP_CONCAT(Score ORDER BY Score desc SEPERATOR ' ') AS Grades
FROM   Grade
GROUP BY Name
```

risultati:

```
+------+------------+
| Name | Grades     |
+------+------------+
| Adam | C+ B A- A+ |
| Bill | D-         |
| John | A-         |
+------+------------+
```

GROUP BY con funzioni AGGREGATE

Tabella ORDINI

```
+---------+------------+----------+-------+--------+
| orderid | customerid | customer | total | items  |
+---------+------------+----------+-------+--------+
|       1 |          1 | Bob      |  1300 |     10 |
|       2 |          3 | Fred     |   500 |      2 |
|       3 |          5 | Tess     |  2500 |      8 |
|       4 |          1 | Bob      |   300 |      6 |
|       5 |          2 | Carly    |   800 |      3 |
|       6 |          2 | Carly    |  1000 |     12 |
|       7 |          3 | Fred     |   100 |      1 |
|       8 |          5 | Tess     | 11500 |     50 |
|       9 |          4 | Jenny    |   200 |      2 |
|      10 |          1 | Bob      |   500 |     15 |
+---------+------------+----------+-------+--------+
```

- **CONTARE**

Restituisce il **numero di righe** che soddisfano un criterio specifico nella clausola `WHERE` .

Es .: numero di ordini per ogni cliente.

```
SELECT customer, COUNT(*) as orders
FROM orders
GROUP BY customer
ORDER BY customer
```

Risultato:

```
+----------+--------+
| customer | orders |
+----------+--------+
| Bob      |      3 |
```

```
| Carly    |      2 |
| Fred     |      2 |
| Jenny    |      1 |
| Tess     |      2 |
+----------+--------+
```

- **SOMMA**

Restituisce la **somma** della colonna selezionata.

Es .: somma del totale e articoli per ciascun cliente.

```
SELECT customer, SUM(total) as sum_total, SUM(items) as sum_items
FROM orders
GROUP BY customer
ORDER BY customer
```

Risultato:

```
+----------+-----------+-----------+
| customer | sum_total | sum_items |
+----------+-----------+-----------+
| Bob      |      2100 |        31 |
| Carly    |      1800 |        15 |
| Fred     |       600 |         3 |
| Jenny    |       200 |         2 |
| Tess     |     14000 |        58 |
+----------+-----------+-----------+
```

- **AVG**

Restituisce il valore **medio** di una colonna di valore numerico.

Ad esempio: valore medio dell'ordine per ciascun cliente.

```
SELECT customer, AVG(total) as avg_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Risultato:

```
+----------+-----------+
| customer | avg_total |
+----------+-----------+
| Bob      |       700 |
| Carly    |       900 |
| Fred     |       300 |
| Jenny    |       200 |
| Tess     |      7000 |
+----------+-----------+
```

- **MAX**

Restituisce il valore **più** alto di una determinata colonna o espressione.

Es .: Totale ordine più alto per ogni cliente.

```
SELECT customer, MAX(total) as max_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Risultato:

```
+----------+-----------+
| customer | max_total |
+----------+-----------+
| Bob      |      1300 |
| Carly    |      1000 |
| Fred     |       500 |
| Jenny    |       200 |
| Tess     |     11500 |
+----------+-----------+
```

- **MIN**

Restituisce il valore **più** basso di una determinata colonna o espressione.

Ad esempio: il totale dell'ordine più basso per ogni cliente.

```
SELECT customer, MIN(total) as min_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Risultato:

```
+----------+-----------+
| customer | min_total |
+----------+-----------+
| Bob      |       300 |
| Carly    |       800 |
| Fred     |       100 |
| Jenny    |       200 |
| Tess     |      2500 |
+----------+-----------+
```

Leggi Raggruppa per online: https://riptutorial.com/it/mysql/topic/3523/raggruppa-per

Capitolo 51: Recupera dalla password di root persa

Examples

Imposta la password di root, abilita l'utente root per il socket e l'accesso http

Risolve il problema di: accesso negato per utente root usando la password SÌ Stop mySQL:

```
sudo systemctl stop mysql
```

Riavvia mySQL, saltando le tabelle di sovvenzione:

```
sudo mysqld_safe --skip-grant-tables
```

Accesso:

```
mysql -u root
```

Nella shell SQL, controlla se gli utenti esistono:

```
select User, password,plugin FROM mysql.user ;
```

Aggiorna gli utenti (il plugin null abilita per tutti i plugin):

```
update mysql.user set password=PASSWORD('mypassword'), plugin = NULL WHERE User = 'root';
exit;
```

In Unix, la shell blocca mySQL senza tabelle di concessione, quindi si riavvia con le tabelle di concessione:

```
sudo service mysql stop
sudo service mysql start
```

Leggi Recupera dalla password di root persa online:
https://riptutorial.com/it/mysql/topic/9973/recupera-dalla-password-di-root-persa

Capitolo 52: replicazione

Osservazioni

La replica viene utilizzata per copiare i dati [di backup] da un server di database MySQL a uno o più server di database MySQL.

Master - Il server di database MySQL, che serve i dati da copiare

Slave - Il server di database MySQL, copia i dati che sono serviti da Master

Con MySQL, la replica è asincrona per impostazione predefinita. Ciò significa che gli slave non devono essere collegati in modo permanente per ricevere gli aggiornamenti dal master. Ad esempio, se il tuo slave è spento o non connesso al master e stai commutando slave su ON o si connette con Master in un secondo momento, allora si sincronizzerà automaticamente con il master.

A seconda della configurazione, è possibile replicare tutti i database, i database selezionati o anche le tabelle selezionate all'interno di un database.

Formati di replica

Esistono due tipi principali di formati di replica

Statement Based Replication (SBR) - che replica intere istruzioni SQL. In questo, il master scrive istruzioni SQL nel registro binario. La replica del master allo slave funziona eseguendo le istruzioni SQL sullo slave.

Row Based Replication (RBR) - che replica solo le righe modificate. In questo, il master scrive gli eventi nel log binario che indicano come vengono modificate le singole righe della tabella. La replica del master allo slave funziona copiando gli eventi che rappresentano le modifiche alle righe della tabella nello slave.

È anche possibile utilizzare una terza varietà, *Mixed Based Replication (MBR)* . In questo, viene utilizzata la registrazione basata su istruzioni e basata su riga. Il registro verrà creato in base a quale è più appropriato per la modifica.

Il formato basato su istruzioni era il predefinito nelle versioni di MySQL precedenti alla 5.7.7. In MySQL 5.7.7 e versioni successive, il formato basato su file è l'impostazione predefinita.

Examples

Master - Slave Replication Setup

Considerare 2 server MySQL per l'installazione della replica, uno è un master e l'altro è uno slave.

Configureremo il Master che dovrebbe tenere un registro di ogni azione eseguita su di esso.

Configureremo il server Slave che dovrebbe guardare il log sul Master e ogni volta che si verificano cambiamenti nel log sul Master, dovrebbe fare la stessa cosa.

Configurazione principale

Prima di tutto, dobbiamo creare un utente sul Master. Questo utente verrà utilizzato da Slave per creare una connessione con il master.

```
CREATE USER 'user_name'@'%' IDENTIFIED BY 'user_password';
GRANT REPLICATION SLAVE ON *.* TO 'user_name'@'%';
FLUSH PRIVILEGES;
```

Cambia `user_name` e `user_password` base al tuo nome utente e password.

Ora il file `my.inf` (my.cnf in Linux) dovrebbe essere modificato. Includere le seguenti righe nella sezione [mysqld].

```
server-id = 1
log-bin = mysql-bin.log
binlog-do-db = your_database
```

La prima riga viene utilizzata per assegnare un ID a questo server MySQL.

La seconda riga indica a MySQL di iniziare a scrivere un log nel file di log specificato. In Linux questo può essere configurato come `log-bin = /home/mysql/logs/mysql-bin.log` . Se si sta iniziando la replica in un server MySQL in cui è già stata utilizzata la replica, assicurarsi che questa directory sia vuota di tutti i registri di replica.

La terza riga viene utilizzata per configurare il database per il quale scriveremo il log. Dovresti sostituire `your_database` con il nome del tuo database.

Assicurati che `skip-networking` non sia stato abilitato e riavvia il server MySQL (Master)

Configurazione slave

`my.inf` file `my.inf` deve essere modificato in slave. Includere le seguenti righe nella sezione [mysqld].

```
server-id = 2
master-host = master_ip_address
master-connect-retry = 60

master-user = user_name
master-password = user_password
replicate-do-db = your_database

relay-log = slave-relay.log
relay-log-index = slave-relay-log.index
```

La prima riga viene utilizzata per assegnare un ID a questo server MySQL. Questo ID dovrebbe essere unico.

La seconda riga è l'indirizzo IP del server master. Cambia questo in base al tuo IP del sistema master

La terza riga viene utilizzata per impostare un limite di tentativi in secondi.

Le prossime due righe indicano lo username e la password allo Slave, usando il quale connette il Master.

La riga successiva imposta il database che deve replicare.

Le ultime due righe utilizzate per assegnare i nomi dei file `relay-log` e `relay-log-index` .

Assicurati che `skip-networking` non sia stato abilitato e riavvia il server MySQL (Slave)

Copia i dati nello slave

Se i dati vengono costantemente aggiunti al Master, dovremo impedire a tutti gli accessi al database sul Master in modo che nulla possa essere aggiunto. Questo può essere ottenuto eseguendo la seguente dichiarazione in Master.

```
FLUSH TABLES WITH READ LOCK;
```

Se nessun dato viene aggiunto al server, è possibile saltare il passaggio precedente.

Stiamo andando a prendere il backup dei dati del Master usando `mysqldump`

```
mysqldump your_database –u root –p > D://Backup/backup.sql;
```

Cambia `your_database` e la directory di backup in base alla configurazione. Ora avrai un file chiamato `backup.sql` nella posizione indicata.

Se il tuo database non esiste nel tuo Slave, crealo eseguendo quanto segue

```
CREATE DATABASE `your_database`;
```

Ora dobbiamo importare il backup nel server MySQL slave.

```
mysql –u root –p your_database  <D://Backup/backup.sql
--->Change `your_database` and backup directory according to your setup
```

Avvia la replica

Per avviare la replica, è necessario trovare il nome del file di registro e la posizione del registro nel master. Quindi, esegui quanto segue in Master

```
SHOW MASTER STATUS;
```

Questo ti darà un'uscita come sotto

```
+---------------------+----------+------------------------------+------------------+
| File                | Position | Binlog_Do_DB                 | Binlog_Ignore_DB |
+---------------------+----------+------------------------------+------------------+
| mysql-bin.000001    | 130      | your_database                |                  |
+---------------------+----------+------------------------------+------------------+
```

Quindi eseguire quanto segue in Slave

```
SLAVE STOP;
CHANGE MASTER TO MASTER_HOST='master_ip_address', MASTER_USER='user_name',
   MASTER_PASSWORD='user_password', MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=130;
SLAVE START;
```

Per prima cosa fermiamo lo schiavo. Quindi diciamo esattamente dove cercare nel file di log principale. Per il nome `MASTER_LOG_FILE` e `MASTER_LOG_POS` , utilizzare i valori ottenuti eseguendo il comando `SHOW MASTER STATUS` sul master.

Dovresti modificare l'IP del Master in `MASTER_HOST` e modificare l'utente e la password di conseguenza.

Lo schiavo ora aspetterà. Lo stato dello slave può essere visualizzato eseguendo quanto segue

```
SHOW SLAVE STATUS;
```

Se in precedenza sono state eseguite le `FLUSH TABLES WITH READ LOCK` in Master, rilasciare le tabelle dal blocco esegui quanto segue

```
UNLOCK TABLES;
```

Ora il Master tiene un log per ogni azione eseguita su di esso e il server Slave guarda il log sul Master. Ogni volta che si verificano cambiamenti nel log del Master, lo slave lo replica.

Errori di replica

Ogni volta che si verifica un errore durante l'esecuzione di una query sullo slave, MySQL interrompe automaticamente la replica per identificare il problema e risolverlo. Questo principalmente perché un evento ha causato la mancata individuazione di una chiave duplicata o di una riga e non può essere aggiornato o eliminato. È possibile saltare tali errori, anche se ciò non è raccomandato

Per saltare solo una query che sta appeso allo slave, utilizzare la seguente sintassi

```
SET GLOBAL sql_slave_skip_counter = N;
```

Questa istruzione salta i successivi N eventi dal master. Questa istruzione è valida solo quando i thread slave non sono in esecuzione. Altrimenti, produce un errore.

```
STOP SLAVE;
SET GLOBAL sql_slave_skip_counter=1;
```

```
START SLAVE;
```

In alcuni casi va bene. Ma se l'istruzione è parte di una transazione multi-statement, diventa più complessa, perché saltando l'istruzione di produzione dell'errore si salterà l'intera transazione.

Se vuoi saltare più query che producono lo stesso codice di errore e se sei sicuro che saltare quegli errori non porterà lo slave incoerente e vuoi saltarli tutti, dovrai aggiungere una riga per saltare quel codice di errore nel tuo `my.cnf` .

Ad esempio potresti voler saltare tutti gli errori duplicati che potresti ottenere

```
1062 | Error 'Duplicate entry 'xyz' for key 1' on query
```

Quindi aggiungi il seguente al tuo `my.cnf`

```
slave-skip-errors = 1062
```

Puoi saltare anche altri tipi di errori o tutti i codici di errore, ma assicurati che saltare questi errori non porti lo slave incoerente. Quanto segue sono la sintassi e gli esempi

```
slave-skip-errors=[err_code1,err_code2,...|all]

slave-skip-errors=1062,1053
slave-skip-errors=all
slave-skip-errors=ddl_exist_errors
```

Leggi replicazione online: https://riptutorial.com/it/mysql/topic/7218/replicazione

Capitolo 53: Ricerca full-text

introduzione

MySQL offre la ricerca FULLTEXT. Cerca nelle tabelle con colonne contenenti testo le corrispondenze migliori per parole e frasi.

Osservazioni

`FULLTEXT` ricerca `FULLTEXT` funziona stranamente su tabelle contenenti piccoli numeri di righe. Quindi, quando lo stai sperimentando, potresti trovare utile ottenere un tavolo di medie dimensioni online. Ecco una tabella di articoli di libri , con titoli e autori. Puoi scaricarlo, decomprimerlo e caricarlo in MySQL.

`FULLTEXT` ricerca `FULLTEXT` è destinata all'uso con l'assistenza umana. È progettato per fornire più corrispondenze di una normale `WHERE column LIKE 'text%'` un'operazione di filtraggio `WHERE column LIKE 'text%'` .

`FULLTEXT` ricerca `FULLTEXT` è disponibile per le tabelle `MyISAM` . È anche disponibile per le tabelle `InnoDB` in MySQL versione 5.6.4 o successiva.

Examples

Semplice ricerca FULLTEXT

```
    SET @searchTerm= 'Database Programming';
 SELECT MATCH (Title) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) Score,
        ISBN, Author, Title
   FROM book
  WHERE MATCH (Title) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE)
  ORDER BY MATCH (Title) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) DESC;
```

Data una tabella denominata `book` con colonne denominate `ISBN` , "Titolo" e "Autore", trova i libri corrispondenti ai termini `'Database Programming'` . Mostra prima le migliori partite.

Affinché ciò funzioni, è necessario che sia disponibile un indice di testo completo nella colonna `Title` :

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_index (Title);
```

Semplice ricerca BOOLEAN

```
    SET @searchTerm= 'Database Programming -Java';
 SELECT MATCH (Title) AGAINST (@searchTerm IN BOOLEAN MODE) Score,
        ISBN, Author, Title
   FROM book
```

```
  WHERE MATCH (Title) AGAINST (@searchTerm IN BOOLEAN MODE)
  ORDER BY MATCH (Title) AGAINST (@searchTerm IN BOOLEAN MODE) DESC;
```

Data una tabella denominata `book` con colonne denominate `ISBN` , `Title` e `Author` , cerca i libri con le parole `'Database'` e `'Programming'` nel titolo, ma non con la parola `'Java'` .

Affinché ciò funzioni, è necessario che sia disponibile un indice di testo completo nella colonna Titolo:

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_index (Title);
```

Ricerca FULLTEXT multi-colonna

```
    SET @searchTerm= 'Date Database Programming';
 SELECT MATCH (Title, Author) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) Score,
        ISBN, Author, Title
   FROM book
  WHERE MATCH (Title, Author) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE)
  ORDER BY MATCH (Title, Author) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) DESC;
```

Data una tabella denominata libro con colonne denominate `ISBN` , `Title` e `Author` , vengono trovati libri corrispondenti ai termini "Programmazione database data". Mostra prima le migliori partite. Le migliori corrispondenze includono libri scritti dal Prof. CJ Date.

(Ma una delle migliori corrispondenze è anche *The Date Doctor's Guide to Dating: come ottenere da First Date a Perfect Mate* . Questo mostra una limitazione della ricerca FULLTEXT: non pretende di capire cose come parti del discorso o il significato delle parole indicizzate.)

Perché ciò avvenga, deve essere disponibile un indice di testo completo nelle colonne Titolo e Autore:

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_author_index (Title, Author);
```

Leggi Ricerca full-text online: https://riptutorial.com/it/mysql/topic/8759/ricerca-full-text

Capitolo 54: Ripristina e reimposta la password di root predefinita per MySQL 5.7+

introduzione

Dopo MySQL 5.7, quando installiamo MySQL a volte non è necessario creare un account di root o fornire una password di root. Per impostazione predefinita, quando si avvia il server, la password predefinita viene archiviata nel file `mysqld.log` . Dobbiamo accedere al sistema usando quella password e dobbiamo cambiarlo.

Osservazioni

Il ripristino e il ripristino della password di root predefinita con questo metodo è applicabile solo per MySQL 5.7+

Examples

Cosa succede quando l'avvio iniziale del server

Dato che la directory dei dati del server è vuota:

- Il server è inizializzato.
- Il certificato SSL e i file chiave vengono generati nella directory dei dati.
- Il plug-in validate_password è installato e abilitato.
- L'account superuser 'root' @ 'localhost' viene creato. La password per il superutente è impostata e memorizzata nel file di log degli errori.

Come cambiare la password di root usando la password predefinita

Per rivelare la password di "root" predefinita:

```
shell> sudo grep 'temporary password' /var/log/mysqld.log
```

Modifica la password di root il prima possibile accedendo con la password temporanea generata e imposta una password personalizzata per l'account superuser:

```
shell> mysql -uroot -p

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass5!';
```

Nota: il plugin validate_password di MySQL è installato per impostazione predefinita. Ciò richiede che le password contengano almeno una lettera maiuscola, una lettera minuscola, una cifra e un carattere speciale e che la lunghezza totale della password sia di almeno 8 caratteri.

reimpostare la password di root quando "/ var / run / mysqld" per il file socket UNIX non esiste "

se dimentico la password, ricevo l'errore.

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

Inserire la password:

ERRORE 1045 (28000): accesso negato per utente 'root' @ 'localhost' (utilizzando la password: YES)

Ho provato a risolvere il problema conoscendo prima lo stato:

```
$ systemctl status mysql.service
```

mysql.service - MySQL Community Server Loaded: caricato (/lib/systemd/system/mysql.service; enabled; impostazione del fornitore: it Attivo: attivo (in esecuzione) da Thu 2017-06-08 14:31:33 IST; 38s fa

Poi ho usato il codice `mysqld_safe --skip-grant-tables &` ho ricevuto l'errore:

> mysqld_safe La directory '/ var / run / mysqld' per il file socket UNIX non esiste.

```
$ systemctl stop  mysql.service
$ ps -eaf|grep mysql
$ mysqld_safe --skip-grant-tables &
```

> Ho risolto:

```
$ mkdir -p /var/run/mysqld
$ chown mysql:mysql /var/run/mysqld
```

Ora uso lo stesso codice `mysqld_safe --skip-grant-tables &` e get

> mysqld_safe Avvio del demone mysqld con i database da / var / lib / mysql

Se uso `$ mysql -u root` :

Versione server: 5.7.18-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle e / o le sue affiliate. Tutti i diritti riservati.

Oracle è un marchio registrato di Oracle Corporation e / o delle sue affiliate. Gli altri nomi possono essere marchi dei rispettivi proprietari.

Digita "aiuto"; o "\ h" per aiuto. Digita "\ c" per cancellare l'istruzione di input corrente.

mysql>

Ora è il momento di cambiare password:

```
mysql> use mysql
mysql> describe user;
```

Informazioni sulla tabella di lettura per il completamento dei nomi di tabelle e colonne È possibile disattivare questa funzionalità per ottenere un avvio più rapido con -A

Database cambiato

```
mysql> FLUSH PRIVILEGES;
mysql> SET PASSWORD FOR root@'localhost' = PASSWORD('newpwd');
```

oppure Se si dispone di un account root mysql che può connettersi ovunque, si dovrebbe anche fare:

```
UPDATE mysql.user SET Password=PASSWORD('newpwd') WHERE User='root';
```

Metodo alternativo:

```
   USE mysql
   UPDATE user SET Password = PASSWORD('newpwd')
   WHERE Host = 'localhost' AND User = 'root';
```

E se hai un account root che può accedere da ovunque:

```
 USE mysql
 UPDATE user SET Password = PASSWORD('newpwd')
 WHERE Host = '%' AND User = 'root';`enter code here
```

ora è necessario `quit` da mysql e fermarsi / iniziare

```
FLUSH PRIVILEGES;
sudo /etc/init.d/mysql stop
sudo /etc/init.d/mysql start
```

ora di nuovo `mysql -u root -p 'e usa la nuova password per ottenere

mysql>

Leggi Ripristina e reimposta la password di root predefinita per MySQL 5.7+ online: https://riptutorial.com/it/mysql/topic/9563/ripristina-e-reimposta-la-password-di-root-predefinita-per-mysql-5-7plus

Capitolo 55: Routine memorizzate (procedure e funzioni)

Parametri

Parametro	Dettagli
RITORNA	Specifica il tipo di dati che può essere restituito da una funzione.
RITORNO	La variabile effettiva o il valore che segue la sintassi `RETURN` è ciò che viene restituito da dove è stata chiamata la funzione.

Osservazioni

Una routine memorizzata è una procedura o una funzione.

Viene richiamata una procedura utilizzando un'istruzione CALL e possono solo passare i valori utilizzando le variabili di output.

Una funzione può essere richiamata all'interno di un'istruzione come qualsiasi altra funzione e può restituire un valore scalare.

Examples

Crea una funzione

La seguente (banale) funzione di esempio restituisce semplicemente il valore `INT` costante `12` .

```
DELIMITER ||
CREATE FUNCTION functionname()
RETURNS INT
BEGIN
    RETURN 12;
END;
||
DELIMITER ;
```

La prima riga definisce a cosa deve essere modificato il carattere delimitatore (`DELIMITER ||`), questo è necessario per essere impostato prima che venga creata una funzione, altrimenti se lo si lascia al suo valore predefinito `;` poi il primo `;` quello che si trova nella funzione corpo sarà preso come la fine `CREATE` , che di solito non è ciò che si desidera.

Dopo aver eseguito `CREATE FUNCTION` è necessario impostare il delimitatore sul valore predefinito di `;` come si vede dopo il codice funzione nell'esempio precedente (`DELIMITER ;`).

L'esecuzione di questa funzione è la seguente:

```
SELECT functionname();
+----------------+
| functionname() |
+----------------+
|             12 |
+----------------+
```

Un esempio leggermente più complesso (ma ancora banale) prende un parametro e aggiunge una costante ad esso:

```
DELIMITER $$
CREATE FUNCTION add_2 ( my_arg INT )
  RETURNS INT
BEGIN
  RETURN (my_arg + 2);
END;
$$
DELIMITER ;

SELECT add_2(12);
+-----------+
| add_2(12) |
+-----------+
|        14 |
+-----------+
```

Notare l'uso di un argomento diverso per la direttiva `DELIMITER` . Puoi effettivamente usare qualsiasi sequenza di caratteri che non appare nel corpo `CREATE` , ma la solita pratica è usare un carattere non alfanumerico raddoppiato come `\\` , `||` o `$$` .

È buona norma modificare sempre il parametro prima e dopo una funzione, procedura o creazione o aggiornamento del trigger poiché alcune GUI non richiedono la modifica del delimitatore mentre le query in esecuzione tramite la riga di comando richiedono sempre il delimitatore da impostare.

Creare una procedura con una preparazione costruita

```
DROP PROCEDURE if exists displayNext100WithName;
DELIMITER $$
CREATE PROCEDURE displayNext100WithName
(    nStart int,
    tblName varchar(100)
)
BEGIN
    DECLARE thesql varchar(500); -- holds the constructed sql string to execute

    -- expands the sizing of the output buffer to accomodate the output (Max value is at least
4GB)
    SET session group_concat_max_len = 4096; -- prevents group_concat from barfing with error
1160 or whatever it is

    SET @thesql=CONCAT("select group_concat(qid order by qid SEPARATOR '%3B') as nums ","from
(    select qid from ");
    SET @thesql=CONCAT(@thesql,tblName," where qid>? order by qid limit 100 )xDerived");
```

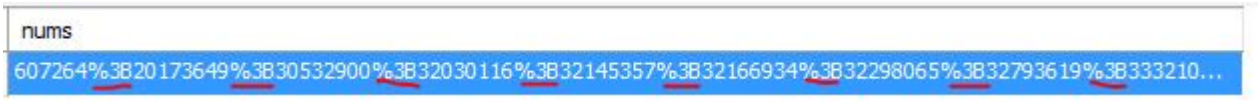
```
    PREPARE stmt1 FROM @thesql; -- create a statement object from the construct sql string to
execute
    SET @p1 = nStart; -- transfers parameter passed into a User Variable compatible with the
below EXECUTE
    EXECUTE stmt1 USING @p1;

    DEALLOCATE PREPARE stmt1; -- deallocate the statement object when finished
END$$
DELIMITER ;
```

La creazione della stored procedure mostra il wrapping con DELIMITER necessario in molti strumenti client.

Esempio di chiamata:

```
call displayNext100WithName(1,"questions_mysql");
```

Esempio di output con separatore `%3B` (punto e virgola):



Procedura memorizzata con i parametri IN, OUT, INOUT

```
DELIMITER $$

DROP PROCEDURE IF EXISTS sp_nested_loop$$
CREATE PROCEDURE sp_nested_loop(IN i INT, IN j INT, OUT x INT, OUT y INT, INOUT z INT)
BEGIN
    DECLARE a INTEGER DEFAULT 0;
    DECLARE b INTEGER DEFAULT 0;
    DECLARE c INTEGER DEFAULT 0;
    WHILE a < i DO
        WHILE b < j DO
            SET c = c + 1;
            SET b = b + 1;
        END WHILE;
        SET a = a + 1;
        SET b = 0;
    END WHILE;
    SET x = a, y = c;
    SET z = x + y + z;
END $$
DELIMITER ;
```

Invoca (CALL) la stored procedure:

```
SET @z = 30;
call sp_nested_loop(10, 20, @x, @y, @z);
SELECT @x, @y, @z;
```

Risultato:

```
+------+------+------+
```

```
|  @x  |  @y  |  @z  |
+------+------+------+
|  10  |  200 |  240 |
+------+------+------+
```

Un parametro `IN` passa un valore in una procedura. La procedura potrebbe modificare il valore, ma la modifica non è visibile al chiamante al ritorno della procedura.

Un parametro `OUT` passa un valore dalla procedura al chiamante. Il suo valore iniziale è NULL all'interno della procedura e il suo valore è visibile al chiamante quando viene restituita la procedura.

Un parametro `INOUT` viene inizializzato dal chiamante, può essere modificato dalla procedura e qualsiasi modifica apportata dalla procedura è visibile al chiamante al ritorno della procedura.

Rif: http://dev.mysql.com/doc/refman/5.7/en/create-procedure.html

Cursori

I cursori ti consentono di itterare i risultati della query uno per riga. `DECLARE` comando `DECLARE` viene utilizzato per avviare il cursore e associarlo a una query SQL specifica:

```
DECLARE student CURSOR FOR SELECT name FROM studend;
```

Diciamo che vendiamo prodotti di alcuni tipi. Vogliamo contare quanti prodotti di ciascun tipo esistono.

I nostri dati:

```
CREATE TABLE product
(
  id   INT(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  type VARCHAR(50)      NOT NULL,
  name VARCHAR(255)     NOT NULL

);
CREATE TABLE product_type
(
  name VARCHAR(50) NOT NULL PRIMARY KEY
);
CREATE TABLE product_type_count
(
  type  VARCHAR(50)      NOT NULL PRIMARY KEY,
  count INT(10) UNSIGNED NOT NULL DEFAULT 0
);

INSERT INTO product_type (name) VALUES
  ('dress'),
  ('food');

INSERT INTO product (type, name) VALUES
  ('dress', 'T-shirt'),
  ('dress', 'Trousers'),
  ('food', 'Apple'),
```

```
  ('food', 'Tomatoes'),
  ('food', 'Meat');
```

Possiamo raggiungere l'obiettivo usando la stored procedure usando il cursore:

```
DELIMITER //
DROP PROCEDURE IF EXISTS product_count;
CREATE PROCEDURE product_count()
  BEGIN
    DECLARE p_type VARCHAR(255);
    DECLARE p_count INT(10) UNSIGNED;
    DECLARE done INT DEFAULT 0;
    DECLARE product CURSOR FOR
      SELECT
        type,
        COUNT(*)
      FROM product
      GROUP BY type;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

    TRUNCATE product_type;

    OPEN product;

    REPEAT
      FETCH product
      INTO p_type, p_count;
      IF NOT done
      THEN
        INSERT INTO product_type_count
        SET
          type  = p_type,
          count = p_count;
      END IF;
    UNTIL done
    END REPEAT;

    CLOSE product;
  END //
DELIMITER ;
```

Quando puoi chiamare la procedura con:

```
CALL product_count();
```

Il risultato sarebbe nella tabella `product_type_count` :

```
type   | count
---------------
dress  |   2
food   |   3
```

Mentre quello è un buon esempio di `CURSOR` , notate come l'intero corpo della procedura può essere sostituito semplicemente

```
INSERT INTO product_type_count
```

```
        (type, count)
    SELECT type, COUNT(*)
        FROM product
        GROUP BY type;
```

Questo funzionerà molto più velocemente.

Multiple ResultSets

A differenza di un'istruzione `SELECT` , una `Stored Procedure` restituisce più set di risultati. Richiede un codice diverso da utilizzare per raccogliere i risultati di una `CALL` in Perl, PHP, ecc.

(Hai bisogno di un codice specifico qui o altrove!)

Crea una funzione

```
DELIMITER $$
CREATE
    DEFINER=`db_username`@`hostname_or_IP`
    FUNCTION `function_name`(optional_param data_type(length_if_applicable))
    RETURNS data_type
BEGIN
    /*
    SQL Statements goes here
    */
END$$
DELIMITER ;
```

Il tipo_di dati RETURNS è qualsiasi tipo di dati MySQL.

Leggi Routine memorizzate (procedure e funzioni) online:
https://riptutorial.com/it/mysql/topic/1351/routine-memorizzate--procedure-e-funzioni-

Capitolo 56: SELEZIONARE

introduzione

`SELECT` è usato per recuperare le righe selezionate da una o più tabelle.

Sintassi

- SELECT DISTINCT [espressioni] FROM TableName [WHERE conditions]; /// Selezione semplice
- SELECT DISTINCT (a), b ... è uguale a SELECT DISTINCT a, b ...
- SELEZIONA [TUTTI | DISTINCT | DISTINCTROW] [HIGH_PRIORITY] [STRAIGHT_JOIN] [SQL_SMALL_RESULT | SQL_BIG_RESULT] [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] espressioni FROM tabelle [WHERE condizioni] [GROUP BY espressioni] [condizione HAVING] [espressione ORDER BY [ASC | DESC]] [LIMIT [offset_value] number_rows | LIMIT number_rows OFFSET offset_value] [PROCEDURE procedure_name] [INTO [OUTFILE 'file_name' opzioni | DUMPFILE 'nome_file' | @ variable1, @ variable2, ... @variable_n] [FOR UPDATE | LOCK IN SHARE MODE]; /// Full Select Sintassi

Osservazioni

Per ulteriori informazioni sull'istruzione `SELECT` di MySQL, consultare MySQL Docs .

Examples

SELEZIONA per nome colonna

```
CREATE TABLE stack(
    id INT,
    username VARCHAR(30) NOT NULL,
    password VARCHAR(30) NOT NULL
);

INSERT INTO stack (`id`, `username`, `password`) VALUES (1, 'Foo', 'hiddenGem');
INSERT INTO stack (`id`, `username`, `password`) VALUES (2, 'Baa', 'verySecret');
```

domanda

```
SELECT id FROM stack;
```

Risultato

```
+------+
| id   |
+------+
|    1 |
|    2 |
+------+
```

SELEZIONA tutte le colonne (*)

domanda

```
SELECT * FROM stack;
```

Risultato

```
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | admin    |
|    2 | stack    | stack    |
+------+----------+----------+
2 rows in set (0.00 sec)
```

Puoi selezionare tutte le colonne da una tabella in un join facendo:

```
SELECT stack.* FROM stack JOIN Overflow ON stack.id = Overflow.id;
```

Best Practice Non usare `*` meno che non si stia eseguendo il debug o il recupero delle righe in array associativi, altrimenti le modifiche dello schema (ADD / DROP / riarrangia le colonne) possono causare errori di applicazione. Inoltre, se si fornisce l'elenco di colonne necessario nel set di risultati, il pianificatore di query di MySQL spesso può ottimizzare la query.

Professionisti:

1. Quando aggiungi / rimuovi colonne, non devi apportare modifiche dove hai usato `SELECT *`
2. È più breve da scrivere
3. Vedi anche le risposte, quindi `SELECT *` -usage può essere mai giustificato?

Contro:

1. Stai restituendo più dati del necessario. Supponiamo che tu aggiunga una colonna VARBINARY che contiene 200k per riga. Hai solo bisogno di questi dati in un unico posto per un singolo record - utilizzando `SELECT *` puoi finire con la restituzione di 2 MB per 10 righe che non ti servono
2. Esplicito su quali dati vengono utilizzati
3. Specificare colonne significa ottenere un errore quando una colonna viene rimossa
4. Il processore di query deve fare ancora un po 'di lavoro - capire quali colonne esistono sul tavolo (grazie @vinodadhikary)
5. Puoi trovare dove una colonna è usata più facilmente
6. Ottieni tutte le colonne in join se usi SELECT *

7. Non è possibile utilizzare in modo sicuro la referenziazione ordinale (sebbene l'uso di riferimenti ordinali per le colonne sia di per sé una cattiva pratica)
8. Nelle query complesse con campi `TEXT` , la query può essere rallentata dall'elaborazione della tabella temporanea meno ottimale

SELEZIONA con DOVE

domanda

```
SELECT * FROM stack WHERE username = "admin" AND password = "admin";
```

Risultato

```
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | admin    |
+------+----------+----------+
1 row in set (0.00 sec)
```

Interrogazione con SELECT nidificato nella clausola WHERE

La clausola `WHERE` può contenere qualsiasi istruzione `SELECT` valida per scrivere query più complesse. Questa è una query 'annidata'

domanda

Le query annidate vengono solitamente utilizzate per restituire valori atomici singoli da query per confronti.

```
SELECT title FROM books WHERE author_id = (SELECT id FROM authors WHERE last_name = 'Bar' AND
first_name = 'Foo');
```

Seleziona tutti i nomi utente senza indirizzo email

```
SELECT * FROM stack WHERE username IN (SELECT username FROM signups WHERE email IS NULL);
```

Dichiarazione di non responsabilità: prendere in considerazione l'utilizzo di join per migliorare le prestazioni quando si confronta un intero set di risultati.

SELEZIONA con LIKE (%)

```
CREATE TABLE stack
(  id int AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(100) NOT NULL
```

```
);

INSERT stack(username) VALUES
('admin'),('k admin'),('adm'),('a adm b'),('b XadmY c'), ('adm now'), ('not here');
```

"adm" ovunque:

```
SELECT * FROM stack WHERE username LIKE "%adm%";
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  2 | k admin   |
|  3 | adm       |
|  4 | a adm b   |
|  5 | b XadmY c |
|  6 | adm now   |
+----+-----------+
```

Inizia con "adm":

```
SELECT * FROM stack WHERE username LIKE "adm%";
+----+----------+
| id | username |
+----+----------+
|  1 | admin    |
|  3 | adm      |
|  6 | adm now  |
+----+----------+
```

Termina con "adm":

```
SELECT * FROM stack WHERE username LIKE "%adm";
+----+----------+
| id | username |
+----+----------+
|  3 | adm      |
+----+----------+
```

Proprio come il carattere `%` in una clausola `LIKE` corrisponde a qualsiasi numero di caratteri, il carattere _ corrisponde a un solo carattere. Per esempio,

```
SELECT * FROM stack WHERE username LIKE "adm_n";
+----+----------+
| id | username |
+----+----------+
|  1 | admin    |
+----+----------+
```

Note sulle prestazioni Se c'è un indice sul `username` , allora

- `LIKE 'adm'` lo stesso effetto di `= 'adm'
- `LIKE 'adm%` è un "range", simile a `BETWEEN..AND..` Può fare buon uso di un indice sulla colonna.
- `LIKE '%adm'` (o qualsiasi variante con un carattere jolly *iniziale)* non possono utilizzare un

indice. Quindi sarà lento. Sui tavoli con molte file, è probabile che sia così lento che è inutile.
- `RLIKE` (`REGEXP`) tende ad essere più lento di `LIKE` , ma ha più capacità.
- Mentre MySQL offre l'indicizzazione `FULLTEXT` su molti tipi di tabelle e colonne, quegli indici `FULLTEXT` *non* vengono utilizzati per soddisfare le query utilizzando `LIKE` .

SELEZIONA con Alias (AS)

Gli alias SQL vengono utilizzati per rinominare temporaneamente una tabella o una colonna. Sono generalmente utilizzati per migliorare la leggibilità.

domanda

```
SELECT username AS val FROM stack;
SELECT username val FROM stack;
```

(Nota: `AS` è sintatticamente opzionale).

Risultato

```
+-------+
| val   |
+-------+
| admin |
| stack |
+-------+
2 rows in set (0.00 sec)
```

SELEZIONA con una clausola LIMIT

Query:

```
SELECT *
  FROM Customers
 ORDER BY CustomerID
 LIMIT 3;
```

Risultato:

Identificativo del cliente	Nome del cliente	Nome del contatto	Indirizzo	Città	Codice postale	Nazione
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlino	12209	Germania
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	Messico DF	05021	Messico
3	Antonio	Antonio	Mataderos	Messico	05023	Messico

	Moreno Taquería	Moreno	2312	DF		

Best Practice Utilizzare sempre `ORDER BY` quando si utilizza `LIMIT` ; altrimenti le righe che otterrai saranno imprevedibili.

Query:

```
SELECT *
  FROM Customers
 ORDER BY CustomerID
 LIMIT 2,1;
```

Spiegazione:

Quando una clausola `LIMIT` contiene due numeri, viene interpretata come `LIMIT offset,count` . Quindi, in questo esempio la query salta due record e ne restituisce uno.

Risultato:

Identificativo del cliente	Nome del cliente	Nome del contatto	Indirizzo	Città	Codice postale	Nazione
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	Messico DF	05023	Messico

Nota:

I valori nelle clausole `LIMIT` devono essere costanti; potrebbero non essere valori di colonna.

SELEZIONA con DISTINCT

La clausola `DISTINCT` dopo `SELECT` elimina le righe duplicate dal set di risultati.

```
CREATE TABLE `car`
(   `car_id` INT UNSIGNED NOT NULL PRIMARY KEY,
    `name` VARCHAR(20),
    `price` DECIMAL(8,2)
);

INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (1, 'Audi A1', '20000');
INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (2, 'Audi A1', '15000');
INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (3, 'Audi A2', '40000');
INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (4, 'Audi A2', '40000');

SELECT DISTINCT `name`, `price` FROM CAR;
+---------+----------+
| name    | price    |
+---------+----------+
| Audi A1 | 20000.00 |
| Audi A1 | 15000.00 |
```

```
| Audi A2 | 40000.00 |
+---------+----------+
```

`DISTINCT` funziona su tutte le colonne per fornire i risultati, non singole colonne. Quest'ultimo è spesso un equivoco di nuovi sviluppatori SQL. In breve, è la distinzione a livello di riga del set di risultati che conta, non la distinzione a livello di colonna. Per visualizzarlo, guarda "Audi A1" nel set di risultati sopra.

Per le versioni successive di MySQL, `DISTINCT` ha implicazioni con il suo uso insieme a `ORDER BY` . L'impostazione di `ONLY_FULL_GROUP_BY` entra in gioco come mostrato nella seguente pagina del manuale MySQL intitolata Gestione MySQL di GROUP BY .

SELEZIONA con LIKE (_)

Un carattere _ in un modello di clausola `LIKE` corrisponde a un singolo carattere.

domanda

```
SELECT username FROM users WHERE users LIKE 'admin_';
```

Risultato

```
+----------+
| username |
+----------+
| admin1   |
| admin2   |
| admin-   |
| adminA   |
+----------+
```

SELEZIONA con CASE o SE

domanda

```
SELECT st.name,
       st.percentage,
       CASE WHEN st.percentage >= 35 THEN 'Pass' ELSE 'Fail' END AS `Remark`
FROM student AS st ;
```

Risultato

```
+-------------------------------+
|   name   | percentage | Remark |
+-------------------------------+
|   Isha   |     67     |  Pass  |
|   Rucha  |     28     |  Fail  |
|   Het    |     35     |  Pass  |
|   Ansh   |     92     |  Pass  |
+-------------------------------+
```

O con IF

```
SELECT st.name,
       st.percentage,
       IF(st.percentage >= 35, 'Pass', 'Fail') AS `Remark`
FROM student AS st ;
```

NB

```
IF(st.percentage >= 35, 'Pass', 'Fail')
```

> Ciò significa: IF st.percentage> = 35 è **TRUE,** quindi restituire `'Pass'` ELSE return `'Fail'`

SELEZIONA CON BETWEEN

È possibile utilizzare la clausola BETWEEN per sostituire una combinazione di condizioni "maggiore di uguale AND inferiore di uguale".

Dati

```
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  2 | root      |
|  3 | toor      |
|  4 | mysql     |
|  5 | thanks    |
|  6 | java      |
+----+-----------+
```

Interrogare con gli operatori

```
SELECT * FROM stack WHERE id >= 2 and id <= 5;
```

Query simile con BETWEEN

```
SELECT * FROM stack WHERE id BETWEEN 2 and 5;
```

Risultato

```
+----+-----------+
| id | username  |
+----+-----------+
|  2 | root      |
|  3 | toor      |
|  4 | mysql     |
|  5 | thanks    |
+----+-----------+
4 rows in set (0.00 sec)
```

Nota

> **TRA utilizza `>=` e `<=` , non `>` e `<` .**

Usando NON TRA

Se vuoi usare il negativo puoi usare `NOT` . Per esempio :

```
SELECT * FROM stack WHERE id NOT BETWEEN 2 and 5;
```

Risultato

```
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  6 | java      |
+----+-----------+
2 rows in set (0.00 sec)
```

Nota

> **NON TRA usi `>` e `<` e non `>=` e `<=`** Cioè, `WHERE id NOT BETWEEN 2 and 5` è uguale a `WHERE (id < 2 OR id > 5)` .

Se hai un indice su una colonna che usi in una ricerca `BETWEEN` , MySQL può usare quell'indice per una scansione di intervallo.

SELEZIONA con l'intervallo di date

```
SELECT ... WHERE dt >= '2017-02-01'
             AND dt  < '2017-02-01' + INTERVAL 1 MONTH
```

Certo, questo potrebbe essere fatto con `BETWEEN` e l'inclusione di `23:59:59` . Ma il modello ha questi vantaggi:

- Non hai pre-calcolare la data di fine (che spesso è una lunghezza esatta dall'inizio)
- Non si includono entrambi gli endpoint (come `BETWEEN`), né si digita '23: 59: 59 'per evitarlo.
- Funziona per `DATE` , `TIMESTAMP` , `DATETIME` e persino `DATETIME(6)` incluso nel microsecondo.
- Si occupa dei giorni bisestili, della fine dell'anno, ecc.
- È indice-friendly (quindi è `BETWEEN`).

Leggi SELEZIONARE online: https://riptutorial.com/it/mysql/topic/3307/selezionare

Capitolo 57: Set di caratteri e regole di confronto

Examples

Dichiarazione

```
CREATE TABLE foo ( ...
    name CHARACTER SET utf8mb4
    ... );
```

Connessione

È fondamentale utilizzare i set di caratteri per dire al server MySQL quali sono i byte del client. Ecco un modo:

```
SET NAMES utf8mb4;
```

Ogni lingua (PHP, Python, Java, ...) ha il suo modo di preferire solitamente `SET NAMES` .

Ad esempio: `SET NAMES utf8mb4` , insieme a una colonna dichiarata `CHARACTER SET latin1` - questo convertirà da latin1 a utf8mb4 quando `INSERTing` e converti indietro quando `SELECTing` .

Quale SET DI CARATTERE e COLLEZIONE?

Ci sono dozzine di set di caratteri con centinaia di regole di confronto. (Una determinata fascicolazione appartiene a un solo set di caratteri.) Vedere l'output di `SHOW COLLATION;` .

Di solito ci sono solo 4 `CHARACTER SETs` che contano:

```
ascii -- basic 7-bit codes.
latin1 -- ascii, plus most characters needed for Western European languages.
utf8 -- the 1-, 2-, and 3-byte subset of utf8.  This excludes Emoji and some of Chinese.
utf8mb4 -- the full set of UTF8 characters, covering all current languages.
```

Tutti includono caratteri inglesi, codificati in modo identico. utf8 è un sottoinsieme di utf8mb4.

La migliore pratica...

- Usa utf8mb4 per qualsiasi colonna `TEXT` o `VARCHAR` che può contenere una varietà di lingue.
- Usa ascii (latin1 è ok) per stringhe esadecimali (UUID, MD5, ecc.) E codici semplici (country_code, codice_postale, ecc.).

utf8mb4 non esisteva fino alla versione 5.5.3, quindi utf8 era il migliore disponibile prima.

Fuori da MySQL , "UTF8" significa le stesse cose di utf8mb4 di MySQL, non di utf8 di MySQL.

Le regole di confronto iniziano con il nome charset e di solito terminano con `_ci` per "case e accento insensibile" o " `_bin` for" per confrontare semplicemente i bit.

L'ultima collazione `utf8mb4_unicode_520_ci` è `utf8mb4_unicode_520_ci` , basata su Unicode 5.20. Se stai lavorando con una sola lingua, potresti, ad esempio, `utf8mb4_polish_ci` , che riorganizzerà leggermente le lettere, in base alle convenzioni polacche.

Impostazione dei set di caratteri su tabelle e campi

È possibile impostare un set di caratteri sia per tabella, sia per singolo campo utilizzando le istruzioni `CHARACTER SET` e `CHARSET` :

```
CREATE TABLE Address (
    `AddressID`   INTEGER NOT NULL PRIMARY KEY,
    `Street`      VARCHAR(80) CHARACTER SET ASCII,
    `City`        VARCHAR(80),
    `Country`     VARCHAR(80) DEFAULT "United States",
    `Active`      BOOLEAN DEFAULT 1,
) Engine=InnoDB default charset=UTF8;
```

`City` e `Country` utilizzeranno `UTF8` , poiché lo impostiamo come set di caratteri predefinito per la tabella. `Street` d'altra parte, userà `ASCII` , come gli abbiamo detto specificamente di farlo.

L'impostazione del giusto set di caratteri dipende in gran parte dal set di dati, ma può anche migliorare notevolmente la portabilità tra i sistemi che lavorano con i dati.

Leggi Set di caratteri e regole di confronto online: https://riptutorial.com/it/mysql/topic/4569/set-di-caratteri-e-regole-di-confronto

Capitolo 58: Si unisce

Sintassi

- `INNER` e `OUTER` sono ignorati.
- `FULL` non è implementato in MySQL.
- "commajoin" (`FROM a,b WHERE ax=by`) è corrugato; usa `FROM a JOIN b ON ax=by` invece.
- FROM a JOIN b ON ax = include le righe che corrispondono in entrambe le tabelle.
- DA UN GIUNTO SINISTRO b ON ax = include tutte le righe da `a` , più i dati corrispondenti da `b` , o `NULLs` se non esiste una riga corrispondente.

Examples

Unire esempi

Query per creare una tabella su db

```
CREATE TABLE `user` (
`id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
`name` varchar(30) NOT NULL,
`course` smallint(5) unsigned DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `course` (
`id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
`name` varchar(50) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

Poiché utilizziamo le tabelle InnoDB e sappiamo che user.course e course.id sono correlati, possiamo specificare una relazione di chiave esterna:

```
ALTER TABLE `user`
ADD CONSTRAINT `FK_course`
FOREIGN KEY (`course`) REFERENCES `course` (`id`)
ON UPDATE CASCADE;
```

Join Query (Inner Join)

```
SELECT user.name, course.name
FROM `user`
INNER JOIN `course` on user.course = course.id;
```

Iscriviti con sottoquery (tabella "Derivata")

```
SELECT x, ...
    FROM ( SELECT y, ... FROM ... ) AS a
    JOIN tbl  ON tbl.x = a.y
    WHERE ...
```

Questo valuterà la sottoquery in una tabella temporanea, quindi `JOIN` a `tbl` .

Prima di 5.6, non poteva esserci un indice sulla tabella temporanea. Quindi, questo era potenzialmente molto inefficiente:

```
SELECT ...
    FROM ( SELECT y, ... FROM ... ) AS a
    JOIN ( SELECT x, ... FROM ... ) AS b  ON b.x = a.y
    WHERE ...
```

Con 5.6, l'ottimizzatore calcola l'indice migliore e lo crea al volo. (Questo ha un sovraccarico, quindi non è ancora 'perfetto'.)

Un altro paradigma comune è avere una sottoquery per inizializzare qualcosa:

```
SELECT
        @n := @n + 1,
        ...
    FROM ( SELECT @n := 0 ) AS initialize
    JOIN the_real_table
    ORDER BY ...
```

(Nota: questo è tecnicamente un `CROSS JOIN` (prodotto cartesiano), come indicato dalla mancanza di `ON` . Tuttavia è efficiente perché la sottoquery restituisce solo una riga che deve essere abbinata alle n righe in `the_real_table` .)

Recupera i clienti con ordini: variazioni su un tema

Ciò otterrà tutti gli ordini per tutti i clienti:

```
SELECT c.CustomerName, o.OrderID
    FROM Customers AS c
    INNER JOIN Orders AS o
        ON c.CustomerID = o.CustomerID
    ORDER BY c.CustomerName, o.OrderID;
```

Questo conterà il numero di ordini per ogni cliente:

```
SELECT c.CustomerName, COUNT(*) AS 'Order Count'
    FROM Customers AS c
    INNER JOIN Orders AS o
        ON c.CustomerID = o.CustomerID
    GROUP BY c.CustomerID;
    ORDER BY c.CustomerName;
```

Inoltre, conta, ma probabilmente più veloce:

```
SELECT  c.CustomerName,
        ( SELECT COUNT(*) FROM Orders WHERE CustomerID = c.CustomerID ) AS 'Order Count'
    FROM Customers AS c
    ORDER BY c.CustomerName;
```

Elencare solo il cliente con gli ordini.

```
SELECT  c.CustomerName,
    FROM Customers AS c
    WHERE EXISTS ( SELECT * FROM Orders WHERE CustomerID = c.CustomerID )
    ORDER BY c.CustomerName;
```

Full Outer Join

MySQL non supporta `FULL OUTER JOIN` , ma ci sono modi per emularne uno.

Impostazione dei dati

```
-- ----------------------------
-- Table structure for `owners`
-- ----------------------------
DROP TABLE IF EXISTS `owners`;
CREATE TABLE `owners` (
`owner_id` int(11) NOT NULL AUTO_INCREMENT,
`owner` varchar(30) DEFAULT NULL,
PRIMARY KEY (`owner_id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Records of owners
-- ----------------------------
INSERT INTO `owners` VALUES ('1', 'Ben');
INSERT INTO `owners` VALUES ('2', 'Jim');
INSERT INTO `owners` VALUES ('3', 'Harry');
INSERT INTO `owners` VALUES ('6', 'John');
INSERT INTO `owners` VALUES ('9', 'Ellie');
-- ----------------------------
-- Table structure for `tools`
-- ----------------------------
DROP TABLE IF EXISTS `tools`;
CREATE TABLE `tools` (
`tool_id` int(11) NOT NULL AUTO_INCREMENT,
`tool` varchar(30) DEFAULT NULL,
`owner_id` int(11) DEFAULT NULL,
PRIMARY KEY (`tool_id`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=latin1;
-- ----------------------------
-- Records of tools
-- ----------------------------
INSERT INTO `tools` VALUES ('1', 'Hammer', '9');
INSERT INTO `tools` VALUES ('2', 'Pliers', '1');
INSERT INTO `tools` VALUES ('3', 'Knife', '1');
INSERT INTO `tools` VALUES ('4', 'Chisel', '2');
INSERT INTO `tools` VALUES ('5', 'Hacksaw', '1');
INSERT INTO `tools` VALUES ('6', 'Level', null);
INSERT INTO `tools` VALUES ('7', 'Wrench', null);
INSERT INTO `tools` VALUES ('8', 'Tape Measure', '9');
```

```
INSERT INTO `tools` VALUES ('9', 'Screwdriver', null);
INSERT INTO `tools` VALUES ('10', 'Clamp', null);
```

Cosa vogliamo vedere?

Vogliamo ottenere una lista, in cui vediamo chi possiede quali strumenti e quali strumenti potrebbero non avere un proprietario.

Le domande

Per fare ciò, possiamo combinare due query usando `UNION` . In questa prima query stiamo unendo gli strumenti sui proprietari usando un `LEFT JOIN` . Ciò aggiungerà tutti i nostri proprietari al nostro gruppo di risultati, non importa se possiedono effettivamente strumenti.

Nella seconda query usiamo un `RIGHT JOIN` per unire gli strumenti ai proprietari. In questo modo riusciamo a ottenere tutti gli strumenti nel nostro set di risultati, se non sono di proprietà di nessuno, la colonna del proprietario conterrà semplicemente `NULL` . Aggiungendo un `WHERE` -clause che sta filtrando da `owners.owner_id IS NULL` stiamo definendo il risultato come quei set di dati, che non sono ancora stati restituiti dalla prima query, poiché stiamo cercando solo i dati nella giusta tabella unita.

Poiché utilizziamo `UNION ALL` il set di risultati della seconda query verrà collegato al primo set di risultati delle query.

```
SELECT `owners`.`owner`, tools.tool
FROM `owners`
LEFT JOIN `tools` ON `owners`.`owner_id` = `tools`.`owner_id`
UNION ALL
SELECT `owners`.`owner`, tools.tool
FROM `owners`
RIGHT JOIN `tools` ON `owners`.`owner_id` = `tools`.`owner_id`
WHERE `owners`.`owner_id` IS NULL;

+-------+--------------+
| owner | tool         |
+-------+--------------+
| Ben   | Pliers       |
| Ben   | Knife        |
| Ben   | Hacksaw      |
| Jim   | Chisel       |
| Harry | NULL         |
| John  | NULL         |
| Ellie | Hammer       |
| Ellie | Tape Measure |
| NULL  | Level        |
| NULL  | Wrench       |
| NULL  | Screwdriver  |
| NULL  | Clamp        |
+-------+--------------+
12 rows in set (0.00 sec)
```

Inner-join per 3 tavoli

supponiamo di avere tre tabelle che possono essere utilizzate per un semplice sito Web con tag.

- La tabella del pugno è per i post.
- Secondo per i tag
- Terzo per tag e posta

pugno tavolo "videogame"

id	titolo	reg_date	Soddisfare
1	BioShock Infinite	2016/08/08	....

tabella "tags"

id	nome
1	Yennefer
2	Elisabetta

tabella "tags_meta"

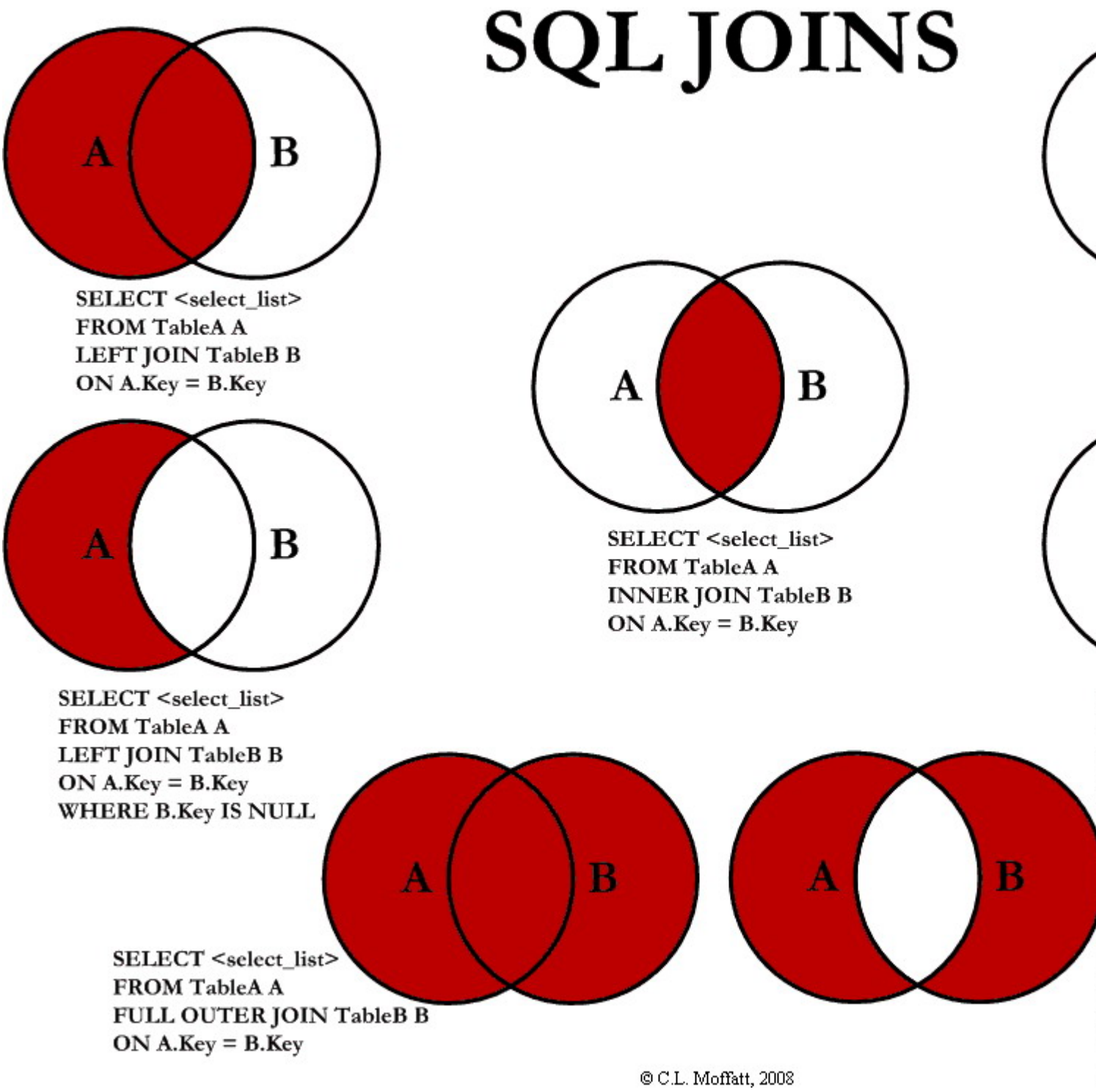
post_id	tag_id
1	2

```
SELECT videogame.id,
    videogame.title,
    videogame.reg_date,
    tags.name,
    tags_meta.post_id
FROM tags_meta
INNER JOIN videogame ON videogame.id = tags_meta.post_id
INNER JOIN tags ON tags.id = tags_meta.tag_id
WHERE tags.name = "elizabeth"
ORDER BY videogame.reg_date
```

questo codice può restituire tutti i post relativi a quel tag "#elizabeth"

Join visualizzati

Se sei una persona con un orientamento visivo, questo diagramma di Venn può aiutarti a capire i diversi tipi di `JOIN` che esistono all'interno di MySQL.



Leggi Si unisce online: https://riptutorial.com/it/mysql/topic/2736/si-unisce

Capitolo 59: Sicurezza tramite GRANT

Examples

La migliore pratica

Limitare root (e qualsiasi altro utente con privilegi SUPER) a

```
GRANT ... TO root@localhost ...
```

Questo impedisce l'accesso da altri server. Dovresti distribuire SUPER a pochissime persone e dovrebbero essere consapevoli delle loro responsabilità. L'applicazione non dovrebbe avere SUPER.

Limita gli accessi delle applicazioni a un database che utilizza:

```
GRANT ... ON dbname.* ...
```

In questo modo, qualcuno che esegue l'hacking nel codice dell'applicazione non può superare dbname. Questo può essere ulteriormente perfezionato tramite uno di questi:

```
GRANT SELECT ON dname.* ...    -- "read only"
GRANT ... ON dname.tblname ... -- "just one table"
```

Il readonly può anche avere bisogno di cose 'sicure' come

```
GRANT SELECT, CREATE TEMPORARY TABLE ON dname.* ...    -- "read only"
```

Come dici tu, non esiste una sicurezza assoluta. Il mio punto qui è che puoi fare alcune cose per rallentare gli hacker. (Lo stesso vale per le persone oneste che fanno finta.)

In rari casi, potrebbe essere necessario che l'applicazione esegua qualcosa disponibile solo per il root. questo può essere fatto tramite una "stored procedure" che ha `SECURITY DEFINER` (e root lo definisce). Ciò esporrà solo ciò che fa l'SP, che potrebbe, ad esempio, essere una determinata azione su un particolare tavolo.

Host (dell'utente @ host)

L'host può essere un nome host o un indirizzo IP. Inoltre, può comportare wild card.

```
GRANT SELECT ON db.* TO sam@'my.domain.com' IDENTIFIED BY 'foo';
```

Esempi: Nota: questi di solito devono essere citati

```
localhost -- the same machine as mysqld
```

```
'my.domain.com' -- a specific domain; this involves a lookup
'11.22.33.44' -- a specific IP address
'192.168.1.%' -- wild card for trailing part of IP address.  (192.168.% and 10.% and 11.% are
"internal" ip addresses.)
```

L'utilizzo di `localhost` si basa sulla sicurezza del server. Per la procedura consigliata, la `root` dovrebbe essere consentita solo tramite il localhost. In alcuni casi, questi significano la stessa cosa: `0.0.0.1` e `::1` .

Leggi Sicurezza tramite GRANT online: https://riptutorial.com/it/mysql/topic/5131/sicurezza-tramite-grant

Capitolo 60: Suggerimenti per le prestazioni Mysql

Examples

Seleziona l'ottimizzazione della dichiarazione

Di seguito sono riportati alcuni suggerimenti da ricordare mentre stiamo scrivendo una query di selezione in MySQL che può aiutarci e ridurre i tempi di richiesta: -

1. Ogni volta che utilizziamo la posizione in una tabella di grandi dimensioni, dovremmo assicurarci che la colonna in cui la clausola è indicizzata o meno. Es .: - Selezionare * dal dipendente dove user_id> 2000. user_id se indicizzato quindi velocizzerà la valutazione della query atlot. Gli indici sono anche molto importanti durante i join e le chiavi esterne.

2. Quando hai bisogno della sezione più piccola di contenuti piuttosto che recuperare interi dati dalla tabella, prova a usare il limite. Piuttosto che scrivere Ex: - Seleziona * dal dipendente. Se hai bisogno solo dei primi 20 dipendenti da lakh, usa solo il limite Ex: - Seleziona * dal dipendente LIMIT 20.

3. È inoltre possibile ottimizzare la query fornendo il nome della colonna che si desidera nel set di risultati. Piuttosto che scrivere Ex: - Seleziona * dal dipendente. Basta menzionare il nome della colonna da cui hai bisogno di dati se la tabella ha un sacco di colonne e vuoi avere i dati per pochi di essi. Es .: - Seleziona id, nome dal dipendente.

4. Colonna dell'indice se si sta utilizzando per verificare la clausola NULL in where. Se hai qualche istruzione come SELECT * FROM tbl_name WHERE key_col IS NULL; quindi se key_col è indicizzato allora la query sarà valutata più velocemente.

Ottimizzazione del layout di archiviazione per le tabelle InnoDB

1. In InnoDB, con un lungo PRIMARY KEY (una singola colonna con un valore lungo o più colonne che formano un valore composito lungo) spreca molto spazio su disco. Il valore della chiave primaria per una riga è duplicato in tutti i record dell'indice secondario che puntano alla stessa riga. Crea una colonna AUTO_INCREMENT come chiave primaria se la tua chiave primaria è lunga.
2. Utilizzare il tipo di dati VARCHAR anziché CHAR per memorizzare stringhe di lunghezza variabile o per colonne con molti valori NULL. Una colonna CHAR (N) accetta sempre N caratteri per memorizzare i dati, anche se la stringa è più corta o il suo valore è NULL. Le tabelle più piccole si adattano meglio al pool buffer e riducono l'I / O del disco.

   Quando si utilizza il formato di riga COMPACT (il formato InnoDB predefinito) e i set di caratteri a lunghezza variabile, come utf8 o sjis, le colonne CHAR (N) occupano una quantità variabile di spazio, ma comunque almeno N byte.

3. Per le tabelle che sono grandi o contengono molti testi ripetitivi o dati numerici, prendere in considerazione l'utilizzo del formato di riga COMPRESSA. È richiesto un numero minore di I / O su disco per portare i dati nel pool buffer o per eseguire scansioni complete della tabella. Prima di prendere una decisione definitiva, misurare la quantità di compressione che è possibile ottenere utilizzando il formato di riga COMPRESSA e COMPATTA. *Avvertenza: i* benchmark raramente mostrano una compressione migliore di 2: 1 e c'è molto overhead nel buffer_pool per COMPRESSED.
4. Una volta che i tuoi dati raggiungono una dimensione stabile, o una tabella crescente è aumentata di decine o qualche centinaia di megabyte, considera l'utilizzo dell'istruzione OPTIMIZE TABLE per riorganizzare la tabella e compattare qualsiasi spazio sprecato. Le tabelle riorganizzate richiedono meno I / O su disco per eseguire scansioni complete della tabella. Questa è una tecnica semplice che può migliorare le prestazioni quando altre tecniche come il miglioramento dell'uso dell'indice o del codice dell'applicazione di ottimizzazione non sono pratici. *Avvertenza* : indipendentemente dalla dimensione della tabella, la tabella OPTIMIZE deve essere eseguita solo raramente. Questo perché è costoso e raramente migliora il tavolo tanto da valerne la pena. InnoDB è abbastanza bravo a mantenere le sue B + Trees libere da molto spazio sprecato.

   OTTIMIZZA TABELLA copia la parte di dati della tabella e ricostruisce gli indici. I vantaggi derivano dal miglioramento della compressione dei dati all'interno degli indici e dalla riduzione della frammentazione all'interno degli spazi tabella e sul disco. I vantaggi variano in base ai dati di ogni tabella. Potresti scoprire che ci sono guadagni significativi per alcuni e non per altri, o che i guadagni diminuiscono nel tempo fino a quando non ottimizzi la tabella. Questa operazione può essere lenta se la tabella è grande o se gli indici da ricostruire non si adattano al pool di buffer. La prima esecuzione dopo l'aggiunta di molti dati a una tabella è spesso molto più lenta delle esecuzioni successive.

Costruire un indice composito

In molte situazioni, un indice composito si comporta meglio di un indice con una singola colonna. Per costruire un indice composito ottimale, popolarlo con colonne in questo ordine.

- `=` colonna (i) prima dalla clausola `WHERE` . (ad es. `INDEX(a,b,...)` per `WHERE a=12 AND b='xyz' ...`)
- Colonna (e) `IN` ; l'ottimizzatore potrebbe essere in grado di saltare attraverso l'indice.
- Un "range" (es. `x BETWEEN 3 AND 9` , `name LIKE 'J%'`) Non utilizzerà nulla oltre la colonna del primo intervallo.
- Tutte le colonne in `GROUP BY` , nell'ordine
- Tutte le colonne in `ORDER BY` , nell'ordine. Funziona solo se tutti sono `ASC` o tutti sono `DESC` o si utilizza 8.0.

Note ed eccezioni:

- Non duplicare nessuna colonna.
- Salta su tutti i casi che non si applicano.
- Se non si utilizzano tutte le colonne di `WHERE` , non è necessario andare a `GROUP BY` , ecc.

- Ci sono casi in cui è utile indicizzare solo le colonne `ORDER BY` , ignorando `WHERE` .
- Non "nascondere" una colonna in una funzione (es. `DATE(x) = ...` non può usare `x` nell'indice.)
- L'indicizzazione 'Prefix' (ad esempio, `text_col(99)`) è improbabile che sia utile; può ferire

Maggiori dettagli e suggerimenti .

Leggi Suggerimenti per le prestazioni Mysql online:
https://riptutorial.com/it/mysql/topic/5752/suggerimenti-per-le-prestazioni-mysql

Capitolo 61: Tabella di mappatura molti-a-molti

Osservazioni

- Mancanza di un id `AUTO_INCREMENT` per questa tabella: il PK dato è il PK "naturale"; non c'è una buona ragione per un surrogato.
- `MEDIUMINT` - Questo è un promemoria per ricordare che tutti gli `INTs` devono essere ridotti al minimo (più piccoli ⇒ più veloci). Ovviamente la dichiarazione qui deve corrispondere alla definizione nella tabella a cui è collegato.
- `UNSIGNED` - Quasi tutti gli INT possono essere dichiarati non negativi
- `NOT NULL` - Beh, è vero, non è vero?
- `InnoDB` - Più efficiente di MyISAM a causa del modo in cui la `PRIMARY KEY` è in cluster con i dati in InnoDB.
- `INDEX(y_id, x_id)` - Il `PRIMARY KEY` rende efficiente andare in una direzione; rende l'altra direzione efficiente. Non c'è bisogno di dire `UNIQUE` ; questo sarebbe un ulteriore sforzo per `INSERTs` .
- Nell'indice secondario, dire solo `INDEX(y_id)` funzionerebbe perché implicherebbe includere `x_id` . Ma preferirei rendere più ovvio che spero in un indice di "copertura".

Si *consiglia* di aggiungere ulteriori colonne alla tabella; questo è raro. Le colonne aggiuntive potrebbero fornire informazioni sulla *relazione* rappresentata dalla tabella.

Si *consiglia* di aggiungere `FOREIGN KEY` vincoli.

Examples

Schema tipico

```
CREATE TABLE XtoY (
    # No surrogate id for this table
    x_id MEDIUMINT UNSIGNED NOT NULL,   -- For JOINing to one table
    y_id MEDIUMINT UNSIGNED NOT NULL,   -- For JOINing to the other table
    # Include other fields specific to the 'relation'
    PRIMARY KEY(x_id, y_id),            -- When starting with X
    INDEX      (y_id, x_id)             -- When starting with Y
) ENGINE=InnoDB;
```

(Vedi Note, sotto, per la logica).

Leggi Tabella di mappatura molti-a-molti online: https://riptutorial.com/it/mysql/topic/4857/tabella-di-mappatura-molti-a-molti

Capitolo 62: Tabella non pivot dinamica con istruzione preparata

Examples

Annulla la rotazione di un insieme dinamico di colonne in base alle condizioni

L'esempio seguente è una base molto utile quando si tenta di convertire i dati della transazione in dati non imperniati per ragioni di BI / reporting, in cui le dimensioni che non devono essere pivot possono avere un insieme dinamico di colonne.

Per il nostro esempio, supponiamo che la tabella dei dati grezzi contenga dati di valutazione dei dipendenti sotto forma di domande contrassegnate.

La tabella dei dati non elaborati è la seguente:

```
create table rawdata

(
 PersonId VARCHAR(255)
,Question1Id INT(11)
,Question2Id INT(11)
,Question3Id INT(11)
)
```

La tabella rawdata è una tabella temporanea come parte della procedura ETL e può avere un numero variabile di domande. L'obiettivo è quello di utilizzare la stessa procedura non pivot per un numero arbitrario di domande, vale a dire colonne che non saranno imperniate.

Di seguito è riportato un esempio di tabella rawdata:

PersonId	Question1Id	Question2Id	Question3Id
Giannaros	1	3	1
Patra	2	4	3

Il noto modo statico per sbloccare i dati, in MYSQL, è utilizzando UNION ALL:

```
create table unpivoteddata

(

 PersonId VARCHAR(255)
,QuestionId VARCHAR(255)
,QuestionValue INT(11)

);

INSERT INTO unpivoteddata SELECT PersonId, 'Question1Id' col, Question1Id
FROM rawdata
```

```
UNION ALL
SELECT PersonId, 'Question2Id' col, Question2Id
FROM rawdata
UNION ALL
SELECT PersonId, 'Question3Id' col, Question3Id
FROM rawdata;
```

Nel nostro caso vogliamo definire un modo per annullare l'annullamento di un numero arbitrario di colonne QuestionId. Per questo abbiamo bisogno di eseguire un'istruzione preparata che sia una selezione dinamica delle colonne desiderate. Per poter scegliere quali colonne debbano essere non pivot, useremo un'istruzione GROUP_CONCAT e sceglieremo le colonne per le quali il tipo di dati è impostato su "int". Nel GROUP_CONCAT includiamo anche tutti gli elementi aggiuntivi della nostra istruzione SELECT da eseguire.

```
set @temp2 = null;

SELECT GROUP_CONCAT(' SELECT ', 'PersonId',',','''',COLUMN_NAME,'''', ' col
',',',COLUMN_NAME,' FROM rawdata' separator ' UNION ALL' ) FROM     INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'rawdata' AND DATA_TYPE = 'Int' INTO     @temp2;

select @temp2;
```

In un'altra occasione avremmo potuto scegliere colonne che il nome della colonna corrisponda a un modello, ad esempio invece di

```
DATA_TYPE = 'Int'
```

uso

```
COLUMN_NAME LIKE 'Question%'
```

o qualcosa di adatto che può essere controllato attraverso la fase ETL.

La dichiarazione preparata è finalizzata come segue:

```
set @temp3 = null;

select concat('INSERT INTO unpivoteddata',@temp2) INTO @temp3;

select @temp3;

prepare stmt FROM @temp3;
execute stmt;
deallocate prepare stmt;
```

La tabella unpivoteddata è la seguente:

```
SELECT * FROM unpivoteddata
```

PersonId	QuestionId	QuestionValue
Giannaros	Question1Id	1
Patra	Question1Id	2
Giannaros	Question2Id	3
Patra	Question2Id	4
Giannaros	Question3Id	1
Patra	Question3Id	3

Selezionare le colonne in base a una condizione e quindi creare una dichiarazione preparata è un modo efficace per disattivare dinamicamente i dati.

Leggi Tabella non pivot dinamica con istruzione preparata online:
https://riptutorial.com/it/mysql/topic/6491/tabella-non-pivot-dinamica-con-istruzione-preparata

Capitolo 63: Tabelle temporanee

Examples

Crea una tabella temporanea

Le tabelle temporanee potrebbero essere molto utili per mantenere i dati temporanei. L'opzione tabelle temporanee è disponibile in MySQL versione 3.23 e successive.

La tabella temporanea verrà automaticamente distrutta al termine della sessione o alla chiusura della connessione. L'utente può anche abbandonare la tabella temporanea.

Lo stesso nome di tabella temporanea può essere utilizzato in molte connessioni contemporaneamente, poiché la tabella temporanea è disponibile e accessibile solo dal client che crea quella tabella.

La tabella temporanea può essere creata nei seguenti tipi

```
--->Basic temporary table creation
CREATE TEMPORARY TABLE tempTable1(
       id INT NOT NULL AUTO_INCREMENT,
       title VARCHAR(100) NOT NULL,
       PRIMARY KEY ( id )
    );

--->Temporary table creation from select query
CREATE TEMPORARY TABLE tempTable1
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

Puoi aggiungere indici mentre costruisci la tabella:

```
CREATE TEMPORARY TABLE tempTable1
        ( PRIMARY KEY(ColumnName2) )
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

`IF NOT EXISTS` parola chiave `IF NOT EXISTS` può essere utilizzata come indicato di seguito per evitare errori di *"tabella già esistente"* . Ma in quel caso la tabella non verrà creata, se il nome della tabella che stai usando esiste già nella tua sessione corrente.

```
CREATE TEMPORARY TABLE IF NOT EXISTS tempTable1
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

Elimina tabella temporanea

Drop Temporary Table viene utilizzato per eliminare la tabella temporanea che si è creata nella sessione corrente.

```
DROP TEMPORARY TABLE tempTable1
```

```
DROP TEMPORARY TABLE IF EXISTS tempTable1
```

Utilizzare `IF EXISTS` per evitare che si verifichi un errore per le tabelle che potrebbero non esistere

Leggi Tabelle temporanee online: https://riptutorial.com/it/mysql/topic/5757/tabelle-temporanee

Capitolo 64: Tempo con precisione al di sotto del secondo

Osservazioni

È necessario essere in MySQL versione 5.6.4 o successive per dichiarare le colonne con i tipi di dati del tempo frazionario.

Ad esempio, `DATETIME(3)` ti darà una risoluzione in millisecondi nei tuoi timestamp e `TIMESTAMP(6)` ti darà una risoluzione in microsecondi su un timestamp in stile * nix.

Leggi questo: http://dev.mysql.com/doc/refman/5.7/en/fractional-seconds.html

`NOW(3)` ti darà il tempo presente dal sistema operativo del tuo server MySQL con precisione millisecondo.

(Si noti che l'aritmetica interna frazionaria di MySQL, come * 0.001, viene sempre gestita come IEEE754 in virgola mobile a doppia precisione, quindi è improbabile che si perda precisione prima che il Sole diventi una stella nana bianca.)

Examples

Ottieni l'ora corrente con precisione millisecondo

```
SELECT NOW(3)
```

fa il trucco

Ottieni l'ora corrente in un modulo simile a un timestamp in Javascript.

I timestamp di Javascript si basano sul tipo di dati `time_t` UNIX venerabile e mostrano il numero di millisecondi dal `1970-01-01 00:00:00` UTC.

Questa espressione ottiene l'ora corrente come numero intero di timestamp JavaScript. (Lo fa correttamente indipendentemente dall'attuale impostazione time_zone.)

```
ROUND(UNIX_TIMESTAMP(NOW(3)) * 1000.0, 0)
```

Se si dispone di valori `TIMESTAMP` memorizzati in una colonna, è possibile recuperarli come timestamp JavaScript intero utilizzando la funzione UNIX_TIMESTAMP ().

```
 SELECT ROUND(UNIX_TIMESTAMP(column) * 1000.0, 0)
```

Se la colonna contiene colonne `DATETIME` e le si recupera come timestamp JavaScript, tali

timestamp verranno compensati dall'offset fuso orario del fuso orario in cui sono memorizzati.

Crea una tabella con colonne per archiviare una volta al secondo.

```
CREATE TABLE times (
     dt DATETIME(3),
     ts TIMESTAMP(3)
 );
```

crea una tabella con campi data / ora con precisione millisecondo.

```
 INSERT INTO times VALUES (NOW(3), NOW(3));
```

inserisce una riga contenente valori `NOW()` con precisione millisecondo nella tabella.

```
 INSERT INTO times VALUES ('2015-01-01 16:34:00.123','2015-01-01 16:34:00.128');
```

inserisce valori di precisione millisecondi specifici.

Si noti che è necessario utilizzare `NOW(3)` anziché `NOW()` se si utilizza tale funzione per inserire valori temporali di alta precisione.

Convertire un valore di data / ora di precisione millisecondo in testo.

`%f` è l'identificatore di formato con precisione frazionaria per la funzione DATE_FORMAT () .

```
SELECT DATE_FORMAT(NOW(3), '%Y-%m-%d %H:%i:%s.%f')
```

visualizza un valore come `2016-11-19 09:52:53.248000` con microsecondi frazionari. Perché abbiamo usato `NOW(3)` , le ultime tre cifre nella frazione sono 0.

Memorizza un timestamp Javascript in una colonna TIMESTAMP

Se si dispone di un valore di timestamp JavaScript, ad esempio `1478960868932` , è possibile convertirlo in un valore temporale frazionario MySQL come questo:

```
FROM_UNIXTIME(1478960868932 * 0.001)
```

È semplice usare quel tipo di espressione per memorizzare il timestamp di Javascript in una tabella MySQL. Fai questo:

```
INSERT INTO table (col) VALUES (FROM_UNIXTIME(1478960868932 * 0.001))
```

(Ovviamente, ti consigliamo di inserire altre colonne.)

Leggi Tempo con precisione al di sotto del secondo online:
https://riptutorial.com/it/mysql/topic/7850/tempo-con-precisione-al-di-sotto-del-secondo

Capitolo 65: Tipi di dati

Examples

Casting implicito / automatico

```
select '123' * 2;
```

Per rendere la **moltiplicazione** con `2` MySQL converte automaticamente la stringa `123` in un numero.

Valore di ritorno:

> 246

La conversione in un numero inizia da sinistra a destra. Se la conversione non è possibile, il risultato è `0`

```
select '123ABC' * 2
```

Valore di ritorno:

> 246

```
select 'ABC123' * 2
```

Valore di ritorno:

> 0

VARCHAR (255) - o no

Max len suggerito

Innanzitutto, menzionerò alcune stringhe comuni che sono sempre esadecimali o altrimenti limitate ad ASCII. Per questi, è necessario specificare `CHARACTER SET ascii` (`latin1` è ok) in modo che non sprechi spazio:

```
UUID CHAR(36) CHARACTER SET ascii -- or pack into BINARY(16)
country_code CHAR(2) CHARACTER SET ascii
ip_address CHAR(39) CHARACTER SET ascii -- or pack into BINARY(16)
phone VARCHAR(20) CHARACTER SET ascii -- probably enough to handle extension
postal_code VARCHAR(20) CHARACTER SET ascii -- (not 'zip_code') (don't know the max

city VARCHAR(100) -- This Russian town needs 91:
    Poselok Uchebnogo Khozyaystva Srednego Professionalno-Tekhnicheskoye Uchilishche Nomer
Odin
country VARCHAR(50) -- probably enough
```

```
name VARCHAR(64) -- probably adequate; more than some government agencies allow
```

Perché non semplicemente 255? Ci sono due ragioni per evitare la pratica comune di usare (255) per tutto.

- Quando un `SELECT` complesso deve creare una tabella temporanea (per una sottoquery, `UNION` , `GROUP BY` , ecc.), La scelta migliore è usare il motore `MEMORY` , che mette i dati nella RAM. Ma i `VARCHARs` vengono trasformati in `CHAR` in questo processo. Ciò rende `VARCHAR(255) CHARACTER SET utf8mb4` 1020 byte. Ciò può portare alla necessità di versare su disco, che è più lento.
- In alcune situazioni, InnoDB esaminerà la dimensione potenziale delle colonne in una tabella e deciderà che sarà troppo grande, interrompendo una `CREATE TABLE` .

VARCHAR contro **TEXT**

Suggerimenti di utilizzo per `*TEXT` , `CHAR` e `VARCHAR` , oltre a alcune best practice:

- Non usare mai `TINYTEXT` .
- Quasi mai usare `CHAR` - è una lunghezza fissa; ogni carattere è la lunghezza massima del `CHARACTER SET` (es. 4 byte / carattere per utf8mb4).
- Con `CHAR` , usa `CHARACTER SET ascii` meno che tu non sappia diversamente.
- `VARCHAR(n)` troncerà su n *caratteri* ; `TEXT` troncerà a un numero di *byte* . (Ma vuoi il troncamento?)
- `*TEXT` *può* rallentare `SELECTs` complessi a causa di come vengono gestite le tabelle temporanee.

INT come AUTO_INCREMENT

È possibile utilizzare qualsiasi dimensione di `INT` per `AUTO_INCREMENT` . `UNSIGNED` è sempre appropriato.

Tieni presente che determinate operazioni "bruciano" ID `AUTO_INCREMENT` . Ciò potrebbe portare a un vuoto inatteso. Esempi: `INSERT IGNORE` e `REPLACE` . *Possono* preallocare un ID *prima di* rendersi conto che non sarà necessario. Questo è un comportamento previsto e di progettazione nel motore InnoDB e non dovrebbe scoraggiare il loro utilizzo.

Altri

C'è già una voce separata per "FLOAT, DOUBLE e DECIMAL" e "ENUM". È probabile che una singola pagina sui tipi di dati sia ingombrante - suggerisco "Tipi di campo" (o dovrebbe essere chiamato "Tipi di dati"?) Essere una panoramica, quindi divisa in queste pagine di argomenti:

- INT
- GALLEGGIANTE, DOPPIO e DECIMALE
- Stringhe (CHAR, TEXT, ecc.)
- BINARIO e BLOB
- DATETIME, TIMESTAMP e amici
- ENUM e SET
- Dati spaziali

- Tipo JSON (MySQL 5.7.8+)
- Come rappresentare Soldi e altri "tipi" comuni che hanno bisogno di abbinarli ai tipi di dati esistenti

Laddove appropriato, ogni pagina di argomento dovrebbe includere, oltre alla sintassi e agli esempi:

- Considerazioni sulla modifica
- Dimensione (byte)
- Contrasto con motori non MySQL (bassa priorità)
- Considerazioni sull'utilizzo del tipo di dati in una CHIAVE PRIMARIA o in una chiave secondaria
- altre migliori pratiche
- altri problemi di prestazioni

(Presumo che questo "esempio" si auto-distrugga quando i miei suggerimenti sono stati soddisfatti o posto il veto.)

Introduzione (numerico)

MySQL offre un numero di tipi numerici diversi. Questi possono essere suddivisi in

Gruppo	tipi
Tipi interi	`INTEGER` , `INT` , `SMALLINT` , `TINYINT` , `MEDIUMINT` , `BIGINT`
Tipi di punti fissi	`DECIMAL` , `NUMERIC`
Tipi di virgola mobile	`FLOAT` , `DOUBLE`
Tipo di valore bit	`BIT`

Tipi interi

Il valore senza segno minimo è sempre 0.

genere	Conservazione (Byte)	Valore minimo (Firmato)	Valore massimo (Firmato)	Valore massi (Non firmato)
`TINYINT`	1	-2^7 -128	2^7-1 127	2^8-1 255
`SMALLINT`	2	-2^{15} 32.768	$2^{15}-1$ 32.767	$2^{16}-1$ 65.535
`MEDIUMINT`	3	-2^{23} -8.388.608	$2^{23}-1$ 8.388.607	$2^{24}-1$ 16.777.215

genere	Conservazione (Byte)	Valore minimo (Firmato)	Valore massimo (Firmato)	Valore massi (Non firmato)
`INT`	4	-2 ^{31} -2.147.483.648	2 ^{31} -1 2,147,483,647	2 ^{32} -1 4,294,967,295
`BIGINT`	8	-2 ^{63} - 9.223.372.036.854.775.808	2 ^{63} -1 9.223.372.036.854.775.807	2 ^{64} -1 18.446.744.07

Tipi di punti fissi

I tipi `DECIMAL` e `NUMERIC` di MySQL memorizzano valori di dati numerici esatti. Si consiglia di utilizzare questi tipi per preservare la precisione esatta, ad esempio per i soldi.

Decimale

Questi valori sono memorizzati in formato binario. In una dichiarazione di colonna, è necessario specificare la precisione e la scala

Precisione rappresenta il numero di cifre significative memorizzate per i valori.

La scala rappresenta il numero di cifre memorizzate *dopo* il decimale

```
salary DECIMAL(5,2)
```

5 rappresenta la `precision` e 2 rappresenta la `scale` . Per questo esempio, l'intervallo di valori che è possibile memorizzare in questa colonna è `-999.99 to 999.99`

Se il parametro di scala è omesso, il suo valore predefinito è 0

Questo tipo di dati può contenere fino a 65 cifre.

Il numero di byte presi da `DECIMAL(M,N)` è *approssimativamente* `M/2` .

Tipi di virgola mobile

`FLOAT` e `DOUBLE` rappresentano tipi di dati *approssimativi* .

genere	Conservazione	Precisione	Gamma
GALLEGGIANTE	4 byte	23 bit significativi / ~ 7 cifre decimali	10 ^ + / - 38
DOPPIO	8 byte	53 bit significativi / ~ 16 cifre decimali	10 ^ + / - 308

`REAL` è sinonimo di `FLOAT` . `DOUBLE PRECISION` è sinonimo di `DOUBLE` .

Sebbene MySQL permetta anche il qualificatore (M, D), *non* usarlo. (M, D) significa che i valori possono essere memorizzati con un massimo di M cifre totali, dove D può essere dopo il decimale. *I numeri saranno arrotondati due volte o troncati; questo causerà più problemi che benefici.*

Poiché i valori in virgola mobile sono approssimativi e non vengono memorizzati come valori esatti, i tentativi di trattarli come esatti nei confronti possono portare a problemi. Nota in particolare che un valore `FLOAT` equivale raramente a un valore `DOUBLE` .

Tipo di valore bit

Il tipo `BIT` è utile per la memorizzazione dei valori dei campi di bit. `BIT(M)` consente la memorizzazione di valori fino a M-bit in cui M è nell'intervallo da `1 to 64`

È inoltre possibile specificare valori con notazione del `bit value` .

```
b'111'      -> 7
b'10000000' -> 128
```

A volte è utile usare 'shift' per costruire un valore a singolo bit, ad esempio `(1 << 7)` per 128.

La dimensione combinata massima di tutte le colonne BIT in una tabella `NDB` è 4096.

CHAR (n)

`CHAR(n)` è una stringa di una lunghezza *fissa* di `n` *caratteri* . Se è `CHARACTER SET utf8mb4` , significa che occupa esattamente `4*n` *byte* , indipendentemente dal testo in esso contenuto.

La maggior parte dei casi d'uso per `CHAR(n)` coinvolge stringhe che contengono caratteri inglesi, quindi dovrebbe essere `CHARACTER SET ascii` . (`latin1` andrà altrettanto bene.)

```
country_code CHAR(2) CHARACTER SET ascii,
postal_code  CHAR(6) CHARACTER SET ascii,
uuid    CHAR(39) CHARACTER SET ascii,  -- more discussion elsewhere
```

DATA, DATETIME, TIMESTAMP, ANNO e ORA

Il datatype `DATE` comprende la data ma nessun componente orario. Il suo formato è `'YYYY-MM-DD'` con un intervallo di '1000-01-01' a '9999-12-31'.

Il tipo `DATETIME` include l'ora con il formato 'AAAA-MM-GG HH: MM: SS'. Ha un intervallo da '1000-01-01 00:00:00' a '9999-12-31 23:59:59'.

Il tipo `TIMESTAMP` è un tipo intero che comprende la data e l'ora con un intervallo effettivo da '1970-01-01 00:00:01' UTC a '2038-01-19 03:14:07' UTC.

Il tipo `YEAR` rappresenta un anno e contiene un intervallo dal 1901 al 2155.

Il tipo `TIME` rappresenta un orario con un formato di 'HH: MM: SS' e contiene un intervallo da '-838:

59: 59' a '838: 59: 59'.

Requisiti di archiviazione:

```
|-----------|--------------------|----------------------------------------|
| Data Type | Before MySQL 5.6.4 | as of MySQL 5.6.4                      |
|-----------|--------------------|----------------------------------------|
| YEAR      |      1 byte        |  1 byte                                |
| DATE      |      3 bytes       |  3 bytes                               |
| TIME      |      3 bytes       |  3 bytes + fractional seconds storage  |
| DATETIME  |      8 bytes       |  5 bytes + fractional seconds storage  |
| TIMESTAMP |      4 bytes       |  4 bytes + fractional seconds storage  |
|-----------|--------------------|----------------------------------------|
```

Fractional Seconds (dalla versione 5.6.4):

```
|-----------------------------|------------------|
| Fractional Seconds Precision | Storage Required |
|-----------------------------|------------------|
|              0              |      0 bytes     |
|              1,2            |      1 byte      |
|              3,4            |      2 byte      |
|              5,6            |      3 byte      |
|-----------------------------|------------------|
```

Consultare le pagine del manuale MySQL DATA, DATETIME e TIMESTAMP , Requisiti di archiviazione dei tipi di dati e Secondi frazionari nei valori temporali .

Leggi Tipi di dati online: https://riptutorial.com/it/mysql/topic/4137/tipi-di-dati

Capitolo 66: Transazione

Examples

Avvia transazione

Una transazione è un gruppo sequenziale di istruzioni SQL come selezionare, inserire, aggiornare o eliminare, che viene eseguito come una singola unità di lavoro.

In altre parole, una transazione non sarà mai completa a meno che ogni singola operazione all'interno del gruppo abbia esito positivo. Se una qualsiasi operazione all'interno della transazione fallisce, l'intera transazione fallirà.

La transazione bancaria sarà il miglior esempio per spiegare questo. Considera un trasferimento tra due account. Per ottenere ciò è necessario scrivere istruzioni SQL che eseguono le seguenti operazioni

1. Verifica la disponibilità dell'importo richiesto nel primo account
2. Detrarre l'importo richiesto dal primo account
3. Depositalo in secondo conto

Se qualcuno di questi processi fallisce, l'intero dovrebbe essere ripristinato allo stato precedente.

ACID: Proprietà delle transazioni

Le transazioni hanno le seguenti quattro proprietà standard

- **Atomicità:** assicura che tutte le operazioni all'interno dell'unità di lavoro siano state completate con successo; in caso contrario, la transazione viene interrotta nel punto di errore e le operazioni precedenti vengono riportate al loro stato precedente.
- **Coerenza:** assicura che il database modifichi correttamente gli stati in seguito a una transazione confermata correttamente.
- **Isolamento:** consente alle transazioni di operare in modo indipendente e trasparente l'una dall'altra.
- **Durata:** assicura che il risultato o l'effetto di una transazione confermata persista in caso di un errore del sistema.

Le transazioni iniziano con l'istruzione `START TRANSACTION` o `BEGIN WORK` e terminano con un'istruzione `COMMIT` o `ROLLBACK` . I comandi SQL tra le istruzioni di inizio e di fine costituiscono la maggior parte della transazione.

```
START TRANSACTION;
SET @transAmt = '500';
SELECT @availableAmt:=ledgerAmt FROM accTable WHERE customerId=1 FOR UPDATE;
UPDATE accTable SET ledgerAmt=ledgerAmt-@transAmt WHERE customerId=1;
UPDATE accTable SET ledgerAmt=ledgerAmt+@transAmt WHERE customerId=2;
COMMIT;
```

Con `START TRANSACTION` , l'autocommit rimane disabilitato fino a quando non si termina la transazione con `COMMIT` o `ROLLBACK` . La modalità autocommit ritorna al suo stato precedente.

`FOR UPDATE` indica (e blocca) la / e riga / e per la durata della transazione.

Mentre la transazione rimane non impegnata, questa transazione non sarà disponibile per altri utenti.

Procedure generali coinvolte nella transazione

- Iniziare la transazione emettendo il comando SQL `BEGIN WORK` o `START TRANSACTION` .
- Esegui tutte le tue istruzioni SQL.
- Controlla se tutto è eseguito in base alle tue esigenze.
- In caso affermativo, emettere il comando `COMMIT` , altrimenti inviare un comando `ROLLBACK` per riportare tutto allo stato precedente.
- Verificare gli errori anche dopo `COMMIT` se si utilizza, o potrebbe eventualmente utilizzare, Galera / PXC.

COMMIT, ROLLBACK e AUTOCOMMIT

AUTOCOMMIT

MySQL impegna automaticamente le dichiarazioni che non fanno parte di una transazione. I risultati di qualsiasi `INSERT UPDATE` , `DELETE` o `INSERT` non preceduta da un `BEGIN` o `START TRANSACTION` saranno immediatamente visibili a tutte le connessioni.

La variabile `AUTOCOMMIT` è impostata su *true* per impostazione predefinita. Questo può essere cambiato nel modo seguente,

```
--->To make autcommit false
SET AUTOCOMMIT=false;
--or
SET AUTOCOMMIT=0;

--->To make autcommit true
SET AUTOCOMMIT=true;
--or
SET AUTOCOMMIT=1;
```

Per visualizzare lo stato `AUTOCOMMIT`

```
SELECT @@autocommit;
```

COMMETTERE

Se `AUTOCOMMIT` impostato su false e la transazione non è stata confermata, le modifiche saranno visibili solo per la connessione corrente.

Dopo l'istruzione `COMMIT` le modifiche alla tabella, il risultato sarà visibile per tutte le connessioni.

Consideriamo due connessioni per spiegare questo

Connessione 1

```
--->Before making autocommit false one row added in a new table
mysql> INSERT INTO testTable VALUES (1);

--->Making autocommit = false
mysql> SET autocommit=0;

mysql> INSERT INTO testTable VALUES (2), (3);
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+
```

Connessione 2

```
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
+-----+
---> Row inserted before autocommit=false only visible here
```

Connessione 1

```
mysql> COMMIT;
--->Now COMMIT is executed in connection 1
mysql> SELECT * FROM testTable;
    +-----+
    | tId |
    +-----+
    |   1 |
    |   2 |
    |   3 |
    +-----+
```

Connessione 2

```
mysql> SELECT * FROM testTable;
    +-----+
    | tId |
    +-----+
    |   1 |
    |   2 |
    |   3 |
    +-----+
--->Now all the three rows are visible here
```

ROLLBACK

Se qualcosa è andato storto nell'esecuzione della query, `ROLLBACK` utilizzato per annullare le

modifiche. Vedi la spiegazione qui sotto

```
--->Before making autocommit false one row added in a new table
mysql> INSERT INTO testTable VALUES (1);

--->Making autocommit = false
mysql> SET autocommit=0;

mysql> INSERT INTO testTable VALUES (2), (3);
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+
```

Ora stiamo eseguendo `ROLLBACK`

```
--->Rollback executed now
mysql> ROLLBACk;

mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
+-----+
--->Rollback removed all rows which all are not committed
```

Una volta eseguito `COMMIT` , quindi `ROLLBACK` non causerà nulla

```
mysql> INSERT INTO testTable VALUES (2), (3);
mysql> SELECT * FROM testTable;
mysql> COMMIT;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+

--->Rollback executed now
mysql> ROLLBACk;

mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+
--->Rollback not removed any rows
```

Se `AUTOCOMMIT` è impostato su *true* , `COMMIT` e `ROLLBACK` sono inutili

Transazione utilizzando il driver JDBC

La transazione che utilizza il driver JDBC viene utilizzata per controllare come e quando una transazione deve eseguire il commit e il rollback. La connessione al server MySQL viene creata utilizzando il driver JDBC

Il driver JDBC per MySQL può essere scaricato qui

Iniziamo con una connessione al database usando il driver JDBC

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection(DB_CONNECTION_URL,DB_USER,USER_PASSWORD);
--->Example for connection url "jdbc:mysql://localhost:3306/testDB");
```

Set di caratteri : indica quale set di caratteri verrà utilizzato dal client per inviare istruzioni SQL al server. Specifica inoltre il set di caratteri che il server deve utilizzare per inviare i risultati al client.

Questo dovrebbe essere menzionato durante la creazione di una connessione al server. Quindi la stringa di connessione dovrebbe essere come,

```
jdbc:mysql://localhost:3306/testDB?useUnicode=true&amp;characterEncoding=utf8
```

Vedi questo per maggiori dettagli su Set di caratteri e regole di confronto

Quando si apre la connessione, la modalità `AUTOCOMMIT` è impostata su *true* per impostazione predefinita, che deve essere modificata *false* per avviare la transazione.

```
con.setAutoCommit(false);
```

Dovresti sempre chiamare il metodo `setAutoCommit()` subito dopo aver aperto una connessione.

Altrimenti usa `START TRANSACTION` o `BEGIN WORK` per iniziare una nuova transazione. Utilizzando `START TRANSACTION` o `BEGIN WORK` , non è necessario modificare `AUTOCOMMIT` *false* . Questo sarà automaticamente disabilitato.

Ora puoi iniziare la transazione. Vedere un esempio di transazione JDBC completo di seguito.

```
package jdbcTest;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;


public class accTrans {

    public static void doTransfer(double transAmount,int customerIdFrom,int customerIdTo) {

        Connection con = null;
```

```
        PreparedStatement pstmt = null;
        ResultSet rs = null;

        try {
            String DB_CONNECTION_URL =
"jdbc:mysql://localhost:3306/testDB?useUnicode=true&amp;characterEncoding=utf8";

            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(DB_CONNECTION_URL,DB_USER,USER_PASSWORD);

            --->set auto commit to false
            con.setAutoCommit(false);
            ---> or use con.START TRANSACTION / con.BEGIN WORK

            --->Start SQL Statements for transaction
            --->Checking availability of amount
            double availableAmt    = 0;
            pstmt = con.prepareStatement("SELECT ledgerAmt FROM accTable WHERE customerId=?
FOR UPDATE");
            pstmt.setInt(1, customerIdFrom);
            rs = pstmt.executeQuery();
            if(rs.next())
                availableAmt    = rs.getDouble(1);

            if(availableAmt >= transAmount)
            {
                ---> Do Transfer
                ---> taking amount from cutomerIdFrom
                pstmt = con.prepareStatement("UPDATE accTable SET ledgerAmt=ledgerAmt-? WHERE
customerId=?");
                pstmt.setDouble(1, transAmount);
                pstmt.setInt(2, customerIdFrom);
                pstmt.executeUpdate();

                ---> depositing amount in cutomerIdTo
                pstmt = con.prepareStatement("UPDATE accTable SET ledgerAmt=ledgerAmt+? WHERE
customerId=?");
                pstmt.setDouble(1, transAmount);
                pstmt.setInt(2, customerIdTo);
                pstmt.executeUpdate();

                con.commit();
            }
            --->If you performed any insert,update or delete operations before
            ----> this availability check, then include this else part
            /*else { --->Rollback the transaction if availability is less than required
                con.rollback();
            }*/

        } catch (SQLException ex) {
            ---> Rollback the transaction in case of any error
            con.rollback();
        } finally {
            try {
                if(rs != null)  rs.close();
                if(pstmt != null) pstmt.close();
                if(con != null) con.close();
            }
        }
    }
```

```
    public static void main(String[] args) {
        doTransfer(500, 1020, 1021);
        -->doTransfer(transAmount, customerIdFrom, customerIdTo);
    }
}
```

La transazione JDBC garantisce che tutte le istruzioni SQL all'interno di un blocco di transazione siano eseguite correttamente, se una delle istruzioni SQL nel blocco di transazione non è riuscita, interrompe e ripristina tutto all'interno del blocco di transazione.

Leggi Transazione online: https://riptutorial.com/it/mysql/topic/5771/transazione

Capitolo 67: TRIGGER

Sintassi

- CREATE [DEFINER = {utente | CURRENT_USER}] TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR EACH ROW [trigger_order] trigger_body
- trigger_time: {PRIMA | DOPO }
- trigger_event: {INSERIRE | AGGIORNAMENTO | ELIMINA }
- trigger_order: {SEGUE | PRECEDES} other_trigger_name

Osservazioni

Due punti devono attirare la tua attenzione se già usi i trigger su altri DB:

PER OGNI FILA

`FOR EACH ROW` è una parte obbligatoria della sintassi

Non puoi fare un trigger di *istruzioni* (una volta per query) come Oracle. È più un problema legato alle prestazioni che una vera caratteristica mancante

CREARE O SOSTITUIRE IL TRIGGER

`CREATE OR REPLACE` non è supportato da MySQL

MySQL non consente questa sintassi, devi invece usare quanto segue:

```
DELIMITER $$

DROP TRIGGER IF EXISTS myTrigger;
$$
CREATE TRIGGER myTrigger
-- ...

$$
DELIMITER ;
```

Fai attenzione, questa **non** è **una transazione atomica** :

- perderai il vecchio trigger se `CREATE` fallisce
- su un carico pesante, possono verificarsi altre operazioni tra il `DROP` e il `CREATE` , utilizzare un `LOCK TABLES myTable WRITE;` innanzitutto per evitare l'inconsistenza dei dati e `UNLOCK TABLES;` dopo il `CREATE` per rilasciare la tabella

Examples

Trigger di base

Crea tabella

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)
```

Crea trigger

```
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

L'istruzione CREATE TRIGGER crea un trigger denominato ins_sum associato alla tabella dell'account. Include anche clausole che specificano il tempo di azione del trigger, l'evento di attivazione e cosa fare quando il trigger si attiva

Inserisci valore

Per utilizzare il trigger, impostare la variabile accumulator (@sum) su zero, eseguire un'istruzione INSERT e quindi vedere quale valore ha in seguito la variabile:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
```

In questo caso, il valore di @sum dopo l'esecuzione dell'istruzione INSERT è 14.98 + 1937.50 - 100 o 1852.48.

Drop Trigger

```
mysql> DROP TRIGGER test.ins_sum;
```

Se si rilascia una tabella, vengono scartati anche eventuali trigger per la tabella.

Tipi di trigger

sincronizzazione

Esistono due modificatori del tempo di azione del trigger:

- `BEFORE` trigger si attiva prima di eseguire la richiesta,
- `AFTER` attivare il fuoco dopo il cambiamento.

Evento scatenante

Esistono tre eventi ai quali è possibile associare trigger:

- `INSERT`
- `UPDATE`
- `DELETE`

Prima di inserire l'esempio di trigger

```
DELIMITER $$

CREATE TRIGGER insert_date
    BEFORE INSERT ON stack
    FOR EACH ROW
BEGIN
        -- set the insert_date field in the request before the insert
    SET NEW.insert_date = NOW();
END;

$$
DELIMITER ;
```

Prima dell'esempio di trigger di aggiornamento

```
DELIMITER $$

CREATE TRIGGER update_date
    BEFORE UPDATE ON stack
    FOR EACH ROW
BEGIN
        -- set the update_date field in the request before the update
    SET NEW.update_date = NOW();
END;

$$
DELIMITER ;
```

Dopo l'esempio di trigger Elimina

```
DELIMITER $$
```

```
CREATE TRIGGER deletion_date
    AFTER DELETE ON stack
    FOR EACH ROW
BEGIN
        -- add a log entry after a successful delete
    INSERT INTO log_action(stack_id, deleted_date) VALUES(OLD.id, NOW());
END;

$$
DELIMITER ;
```

Leggi TRIGGER online: https://riptutorial.com/it/mysql/topic/3069/trigger

Capitolo 68: UNIONE

Sintassi

- UNION DISTINCT: deduplica dopo aver combinato i SELECT
- UNION ALL - non dedup (più veloce)
- UNION - il valore predefinito è DISTINCT
- SELEZIONA ... UNION SELECT ... - è OK, ma ambiguo su `ORDER BY`
- (SELEZIONA ...) UNIONE (SELEZIONA ...) ORDINA PER ... - risolve l'ambiguità

Osservazioni

UNION non usa più CPU.

UNION sempre * comporta una tabella temporanea per raccogliere i risultati. * A partire da 5.7.3 / MariaDB 10.1, alcune forme di UNION forniscono i risultati senza utilizzare una tabella tmp (quindi, più veloce).

Examples

Combinare le istruzioni SELECT con UNION

È possibile combinare i risultati di due query identicamente strutturate con la parola chiave `UNION` .

Ad esempio, se si desidera un elenco di tutte le informazioni di contatto da due tabelle separate, `authors` e `editors` , ad esempio, è possibile utilizzare la parola chiave `UNION` modo:

```
select name, email, phone_number
from authors

union

select name, email, phone_number
from editors
```

L'utilizzo `union` da sola eliminerà i duplicati. Se dovessi conservare i duplicati nella tua query, potresti utilizzare la parola chiave `ALL` modo: `UNION ALL` .

ORDINATO DA

Se è necessario ordinare i risultati di UNION, utilizzare questo modello:

```
( SELECT ... )
UNION
( SELECT ... )
ORDER BY
```

Senza le parentesi, l'ORDER BY finale apparterrebbe all'ultima SELECT.

Paginazione tramite OFFSET

Quando si aggiunge un LIMIT a UNION, questo è il modello da utilizzare:

```
( SELECT ... ORDER BY x  LIMIT 10 )
UNION
( SELECT ... ORDER BY x  LIMIT 10 )
ORDER BY x  LIMIT 10
```

Dal momento che non è possibile prevedere quale / i SELEZIONA / i arriverà il "10", sarà necessario ottenere 10 da ciascuno, quindi ridurre ulteriormente l'elenco, ripetendo sia `ORDER BY` che `LIMIT` .

Per la quarta pagina di 10 articoli, questo modello è necessario:

```
( SELECT ... ORDER BY x  LIMIT 40 )
UNION
( SELECT ... ORDER BY x  LIMIT 40 )
ORDER BY x  LIMIT 30, 10
```

Vale a dire, raccogliere il valore di 4 pagine in ciascuna `SELECT` , quindi eseguire lo `OFFSET UNION` .

Combinare i dati con colonne diverse

```
SELECT name, caption as title, year, pages FROM books
UNION
SELECT name, title, year, 0 as pages FROM movies
```

Quando si combinano 2 set di record con colonne diverse, emulare quelli mancanti con i valori predefiniti.

UNIONE TUTTI e UNIONE

SELEZIONA 1,22,44 UNIONE SELEZIONA 2,33,55

1	22	44
1	22	44
2	33	55

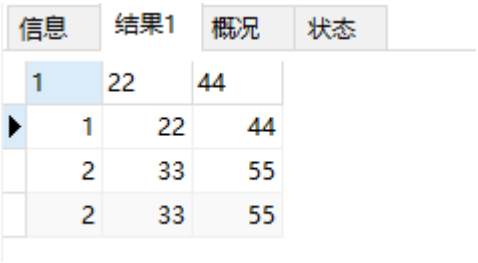
SELECT 1,22,44 UNION SELECT 2,33,55 UNION SELECT 2,33,55

Il risultato è lo stesso di sopra.

usa UNION ALL

quando

SELECT 1,22,44 UNION SELECT 2,33,55 UNION ALL SELECT 2,33,55


信息 结果1 概况 状态

1	22	44
1	22	44
2	33	55
2	33	55

Combinazione e fusione dei dati su diverse tabelle MySQL con le stesse colonne in righe univoche e query in esecuzione

Questo ***UNION ALL*** combina i dati di più tabelle e funge da alias di nome tabella da utilizzare per le query:

```
SELECT YEAR(date_time_column), MONTH(date_time_column), MIN(DATE(date_time_column)),
MAX(DATE(date_time_column)), COUNT(DISTINCT (ip)), COUNT(ip), (COUNT(ip) / COUNT(DISTINCT
(ip))) AS Ratio
FROM (
    (SELECT date_time_column, ip FROM server_log_1 WHERE state = 'action' AND log_id = 150)
UNION ALL
    (SELECT date_time_column, ip FROM server_log_2 WHERE state = 'action' AND log_id = 150)
UNION ALL
    (SELECT date_time_column, ip FROM server_log_3 WHERE state = 'action' AND log_id = 150)
UNION ALL
    (SELECT date_time_column, ip FROM server_log WHERE state = 'action' AND log_id = 150)
) AS table_all
GROUP BY YEAR(date_time_column), MONTH(date_time_column);
```

Leggi UNIONE online: https://riptutorial.com/it/mysql/topic/3847/unione

Capitolo 69: Unioni MySQL

Sintassi

- SELECT nome_colonna FROM tabella1 UNION SELECT nome_colonna FROM tabella2;
- SELECT nome_colonna FROM tabella1 UNION ALL SELECT nome_colonna / i FROM tabella2;
- SELECT nome_colonna FROM tabella1 WHERE col_name = "XYZ" UNION ALL SELECT nome_colonna (s) FROM table2 WHERE col_name = "XYZ";

Osservazioni

`UNION DISTINCT` è uguale a `UNION` ; è più lento di `UNION ALL` causa di un passaggio di deduplicazione. Una buona pratica è sempre precisare `DISTINCT` o `ALL` , segnalando così che hai pensato a cosa fare.

Examples

Operatore dell'Unione

L'operatore UNION viene utilizzato per combinare il set di risultati (*solo valori distinti*) di due o più istruzioni SELECT.

Query: (per selezionare tutte le città diverse (*solo valori distinti*) dalle tabelle "Clienti" e "Fornitori")

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Risultato:

```
Number of Records: 10

City
------
Aachen
Albuquerque
Anchorage
Annecy
Barcelona
Barquisimeto
Bend
Bergamo
Berlin
Bern
```

Unione TUTTI

UNION ALL per selezionare tutte le città (anche i valori duplicati) dalle tabelle "Clienti" e "Fornitori".

Query:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

Risultato:

```
Number of Records: 12

City
-------
Aachen
Albuquerque
Anchorage
Ann Arbor
Annecy
Barcelona
Barquisimeto
Bend
Bergamo
Berlin
Berlin
Bern
```

UNIONE TUTTO CON DOVE

UNION ALL per selezionare tutti (valori duplicati anche) città tedesche dalle tabelle "Clienti" e "Fornitori". Qui `Country="Germany"` deve essere specificato nella clausola where.

Query:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

Risultato:

```
Number of Records: 14
```

Città	Nazione
Aachen	Germania

Berlino	Germania
Berlino	Germania
Brandenburg	Germania
Cunewalde	Germania
Cuxhaven	Germania
Francoforte	Germania
Francoforte aM	Germania
Köln	Germania
Lipsia	Germania
Mannheim	Germania
München	Germania
Münster	Germania
Stoccarda	Germania

Leggi Unioni MySQL online: https://riptutorial.com/it/mysql/topic/5376/unioni-mysql

Capitolo 70: UNISCI: iscriviti a 3 tavoli con lo stesso nome di id.

Examples

Unisci 3 tabelle su una colonna con lo stesso nome

```
CREATE TABLE Table1 (
    id INT UNSIGNED NOT NULL,
    created_on DATE NOT NULL,
    PRIMARY KEY (id)
)
CREATE TABLE Table2 (
    id INT UNSIGNED NOT NULL,
    personName VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
)
CREATE TABLE Table3 (
    id INT UNSIGNED NOT NULL,
    accountName VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
)
```

dopo aver creato le tabelle, è possibile eseguire una query di selezione per ottenere gli id di tutte e tre le tabelle uguali

```
SELECT
    t1.id AS table1Id,
    t2.id AS table2Id,
    t3.id AS table3Id
FROM Table1 t1
LEFT JOIN Table2 t2 ON t2.id = t1.id
LEFT JOIN Table3 t3 ON t3.id = t1.id
```

Leggi UNISCI: iscriviti a 3 tavoli con lo stesso nome di id. online: https://riptutorial.com/it/mysql/topic/9921/unisci--iscriviti-a-3-tavoli-con-lo-stesso-nome-di-id-

Capitolo 71: Uno a molti

introduzione

L'idea di uno a molti (1: M) riguarda l'unione di righe tra di loro, in particolare i casi in cui una singola riga in una tabella corrisponde a più righe in un'altra.

1: M è unidirezionale, ovvero ogni volta che si esegue una query su una relazione 1: M, è possibile utilizzare la riga 'uno' per selezionare 'molte' righe in un'altra tabella, ma non è possibile utilizzare una singola riga 'molte' per seleziona più di una singola riga "uno".

Osservazioni

Per la maggior parte dei casi, lavorare con una relazione 1: M richiede che comprendiamo *le chiavi primarie* e *le chiavi esterne* .

Una chiave primaria è una colonna in una tabella in cui qualsiasi singola riga di quella colonna rappresenta una singola entità oppure, selezionando un valore in una colonna chiave primaria, si ottiene esattamente una riga. Utilizzando gli esempi precedenti, un EMP_ID rappresenta un singolo dipendente. Se esegui una query per ogni singolo EMP_ID, vedrai una singola riga che rappresenta il dipendente corrispondente.

Una chiave esterna è una colonna in una tabella che corrisponde alla chiave primaria di un'altra tabella diversa. Dal nostro esempio precedente, MGR_ID nella tabella EMPLOYEES è una chiave esterna. Generalmente per unire due tabelle, ti unirai a loro in base alla chiave primaria di una tabella e alla chiave esterna in un'altra.

Examples

Esempio di tabelle aziendali

Considera un'azienda in cui ogni dipendente che è un manager, gestisce uno o più dipendenti e ogni dipendente ha solo un manager.

Ciò si traduce in due tabelle:

DIPENDENTI

EMP_ID	NOME DI BATTESIMO	COGNOME	MGR_ID
E01	Johnny	Appleseed	M02
E02	Erin	Macklemore	M01
E03	Colby	lavoro d'ufficio	M03

EMP_ID	NOME DI BATTESIMO	COGNOME	MGR_ID
E04	Ron	Sonswan	M01

DIRIGENTI

MGR_ID	NOME DI BATTESIMO	COGNOME
M01	Forte	McQueen
M02	Prepotente	Pantaloni
M03	barile	Jones

Ottieni i dipendenti gestiti da un singolo manager

```
SELECT e.emp_id , e.first_name , e.last_name FROM employees e INNER JOIN managers m ON m.mgr_id
= e.mgr_id WHERE m.mgr_id = 'M01' ;
```

Risultati in:

EMP_ID	NOME DI BATTESIMO	COGNOME
E02	Erin	Macklemore
E04	Ron	Sonswan

In definitiva, per ogni manager su cui eseguiamo la query, vedremo restituire 1 o più dipendenti.

Ottieni il manager per un singolo dipendente

Consulta le tabelle di esempio sopra quando guardi questo esempio.

```
SELECT m.mgr_id , m.first_name , m.last_name FROM managers m INNER JOIN employees e ON e.mgr_id
= m.mgr_id WHERE e.emp_id = 'E03' ;
```

MGR_ID	NOME DI BATTESIMO	COGNOME
M03	barile	Jones

Poiché questo è l'inverso dell'esempio sopra riportato, sappiamo che per ogni dipendente per il quale richiediamo, vedremo sempre un manager corrispondente.

Leggi Uno a molti online: https://riptutorial.com/it/mysql/topic/9600/uno-a-molti

Capitolo 72: Uso delle variabili

Examples

Impostazione delle variabili

Ecco alcuni modi per impostare le variabili:

1. È possibile impostare una variabile su uno specifico, una stringa, un numero, una data usando SET

   EX: SET @var_string = 'my_var'; SET @var_num = '2' SET @var_date = '2015-07-20';

2. puoi impostare una variabile come risultato di un'istruzione select usando: =

   EX: Seleziona @var: = '123'; (Nota: devi usare: = quando assegni una variabile che non usa la sintassi SET, perché in altre istruzioni, (seleziona, aggiorna ...) il "=" è usato per confrontare, quindi quando aggiungi un due punti prima del " = ", stai dicendo" Questo non è un confronto, questo è un SET ").

3. È possibile impostare una variabile come risultato di un'istruzione select utilizzando INTO

   (Questo è stato particolarmente utile quando avevo bisogno di scegliere dinamicamente da quale partizione eseguire le query)

EX: SET @start_date = '2015-07-20'; SET @end_date = '2016-01-31';

```
#this gets the year month value to use as the partition names
SET @start_yearmonth = (SELECT EXTRACT(YEAR_MONTH FROM @start_date));
SET @end_yearmonth = (SELECT EXTRACT(YEAR_MONTH FROM @end_date));

#put the partitions into a variable
SELECT GROUP_CONCAT(partition_name)
FROM information_schema.partitions p
WHERE table_name = 'partitioned_table'
AND SUBSTRING_INDEX(partition_name,'P',-1) BETWEEN @start_yearmonth AND @end_yearmonth
INTO @partitions;

#put the query in a variable. You need to do this, because mysql did not recognize my variable
as a variable in that position. You need to concat the value of the variable together with the
rest of the query and then execute it as a stmt.
SET @query =
CONCAT('CREATE TABLE part_of_partitioned_table (PRIMARY KEY(id))
SELECT partitioned_table.*
FROM partitioned_table PARTITION(', @partitions,')
JOIN users u USING(user_id)
WHERE date(partitioned_table.date) BETWEEN ', @start_date,' AND ', @end_date);

#prepare the statement from @query
PREPARE stmt FROM @query;
#drop table
DROP TABLE IF EXISTS tech.part_of_partitioned_table;
```

```
#create table using statement
EXECUTE stmt;
```

Numero e gruppo di righe Usando le variabili nell'istruzione Select

Diciamo che abbiamo un tavolo `team_person` come di seguito:

```
+======+===========+
| team |    person |
+======+===========+
|   A  |      John |
+------+-----------+
|   B  |     Smith |
+------+-----------+
|   A  |    Walter |
+------+-----------+
|   A  |     Louis |
+------+-----------+
|   C  | Elizabeth |
+------+-----------+
|   B  |     Wayne |
+------+-----------+


CREATE TABLE team_person AS SELECT 'A' team, 'John' person
UNION ALL  SELECT 'B' team,  'Smith' person
UNION ALL  SELECT 'A' team,  'Walter' person
UNION ALL  SELECT 'A' team,  'Louis' person
UNION ALL  SELECT 'C' team,  'Elizabeth' person
UNION ALL  SELECT 'B' team,  'Wayne' person;
```

Per selezionare la tabella `team_person` con la colonna `row_number` , anche

```
SELECT @row_no := @row_no+1 AS row_number, team, person
FROM team_person, (SELECT @row_no := 0) t;
```

O

```
SET @row_no := 0;
SELECT  @row_no := @row_no + 1 AS row_number, team, person
FROM team_person;
```

produrrà il risultato qui sotto:

```
+============+======+===========+
| row_number | team |    person |
+============+======+===========+
|          1 |   A  |      John |
+------------+------+-----------+
|          2 |   B  |     Smith |
+------------+------+-----------+
|          3 |   A  |    Walter |
+------------+------+-----------+
|          4 |   A  |     Louis |
+------------+------+-----------+
```

```
|          5 |   C  | Elizabeth |
+------------+------+-----------+
|          6 |   B  |     Wayne |
+------------+------+-----------+
```

Infine, se vogliamo ottenere il `row_number` gruppo per colonna `team`

```
SELECT @row_no := IF(@prev_val = t.team, @row_no + 1, 1) AS row_number
   ,@prev_val := t.team AS team
   ,t.person
FROM team_person t,
  (SELECT @row_no := 0) x,
  (SELECT @prev_val := '') y
ORDER BY t.team ASC,t.person DESC;

+============+======+===========+
| row_number | team |    person |
+============+======+===========+
|          1 |   A  |    Walter |
+------------+------+-----------+
|          2 |   A  |     Louis |
+------------+------+-----------+
|          3 |   A  |      John |
+------------+------+-----------+
|          1 |   B  |     Wayne |
+------------+------+-----------+
|          2 |   B  |     Smith |
+------------+------+-----------+
|          1 |   C  | Elizabeth |
+------------+------+-----------+
```

Leggi Uso delle variabili online: https://riptutorial.com/it/mysql/topic/5013/uso-delle-variabili

Capitolo 73: VISTA

Sintassi

- CREATE VIEW view_name AS SELECT nome_colonna FROM nome_tabella WHERE condition; /// Semplifica la sintassi della vista
- CREARE [O SOSTITUIRE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] [DEFINER = {utente | CURRENT_USER}] [SQL SECURITY {DEFINER | INVOKER}] VIEW view_name [(column_list)] AS select_statement [WITH [CASCADED | LOCAL] CHECK OPTION]; /// Full Create view syntax
- DROP VIEW [IF EXISTS] [db_name.] View_name; /// Drop view sintassi

Parametri

parametri	Dettagli
view_name	Nome della vista
SELECT statement	Dichiarazioni SQL da impacchettare nelle viste. Può essere un'istruzione SELECT per recuperare i dati da una o più tabelle.

Osservazioni

Le viste sono tabelle virtuali e non contengono i dati restituiti. Possono salvarti dalla scrittura di query complesse ancora e ancora.

- **Prima che una vista venga resa** specifica, la sua specificazione consiste interamente in un'istruzione `SELECT` . L'istruzione `SELECT` non può contenere una sottoquery nella clausola FROM.
- **Una volta che una vista è fatto** è usato in gran parte proprio come un tavolo e può essere `SELECT` cata dal proprio come un tavolo.

Devi creare viste, quando vuoi limitare alcune colonne della tua tabella, dall'altro utente.

- Ad esempio: nella propria organizzazione, si desidera che i gestori visualizzino poche informazioni da una tabella denominata "Vendite", ma non si desidera che i tecnici del software possano visualizzare tutti i campi della tabella "Vendite". Qui puoi creare due diverse viste per i tuoi manager e i tuoi ingegneri del software.

Prestazioni `VIEWs` sono zucchero sintattico. Tuttavia, la performance può essere o meno peggiore della query equivalente con la selezione della vista inserita. Lo Strumento di ottimizzazione tenta di eseguire questo "inserimento" per te, ma non sempre ha esito positivo. MySQL 5.7.6 fornisce ulteriori miglioramenti nello Strumento di ottimizzazione. Ma, a prescindere, l'utilizzo di una `VIEW`

non genererà una query *più veloce* .

Examples

Crea una vista

privilegi

L'istruzione CREATE VIEW richiede il privilegio CREATE VIEW per la vista e alcuni privilegi per ogni colonna selezionata dall'istruzione SELECT. Per le colonne utilizzate altrove nell'istruzione SELECT, è necessario avere il privilegio SELECT. Se è presente la clausola OR REPLACE, è necessario disporre anche del privilegio DROP per la vista. CREATE VIEW potrebbe anche richiedere il privilegio SUPER, a seconda del valore DEFINER, come descritto più avanti in questa sezione.

Quando si fa riferimento a una vista, si verifica il controllo dei privilegi.

Una vista appartiene a un database. Per impostazione predefinita, viene creata una nuova vista nel database predefinito. Per creare esplicitamente la vista in un determinato database, utilizzare un nome completo

Per esempio:

db_name.view_name

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Nota: all'interno di un database, le tabelle di base e le viste condividono lo stesso spazio dei nomi, pertanto una tabella di base e una vista non possono avere lo stesso nome.

A VISTA può:

- essere creato da molti tipi di istruzioni SELECT
- fare riferimento alle tabelle di base o ad altre viste
- usa join, UNION e sottoquery
- SELECT non deve nemmeno fare riferimento a nessuna tabella

Un altro esempio

L'esempio seguente definisce una vista che seleziona due colonne da un'altra tabella e un'espressione calcolata da tali colonne:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
```

```
+------+-------+-------+
```

restrizioni

- Prima di MySQL 5.7.7, l'istruzione SELECT non può contenere una sottoquery nella clausola FROM.
- L'istruzione SELECT non può fare riferimento a variabili di sistema o variabili definite dall'utente.
- All'interno di un programma memorizzato, l'istruzione SELECT non può fare riferimento a parametri del programma o variabili locali.
- L'istruzione SELECT non può fare riferimento ai parametri dell'istruzione preparati.
- Qualsiasi tabella o vista cui fa riferimento la definizione deve esistere. Dopo aver creato la vista, è possibile rilasciare una tabella o visualizzarla
  la definizione si riferisce a. In questo caso, l'uso della vista provoca un errore. Per verificare la definizione di una vista per problemi di questo tipo, utilizzare l'istruzione CHECK TABLE.
- La definizione non può fare riferimento a una tabella TEMPORARY e non è possibile crea una vista TEMPORANEA.
- Non è possibile associare un trigger a una vista.
- Gli alias per i nomi delle colonne nell'istruzione SELECT vengono confrontati con la lunghezza massima della colonna di 64 caratteri (non l'alias massimo lunghezza di 256 caratteri).
- A `VIEW` può o non può ottimizzare così come l'equivalente `SELECT` . È improbabile che l'ottimizzazione sia migliore.

Una vista da due tavoli

Una vista è molto utile quando può essere utilizzata per inserire dati da più di una tabella.

```
CREATE VIEW myview AS
SELECT a.*, b.extra_data FROM main_table a
LEFT OUTER JOIN other_table b
ON a.id = b.id
```

Nelle viste mysql non si materializzano. Se ora esegui la query semplice `SELECT * FROM myview` , mysql eseguirà effettivamente LEFT JOIN dietro la scena.

Una vista creata una volta può essere unita ad altre viste o tabelle

Aggiornamento di una tabella tramite VISTA

A `VIEW` funziona molto come un tavolo. Sebbene tu possa `UPDATE` una tabella, potresti o non riuscire ad aggiornare una vista in quella tabella. In generale, se la `SELECT` nella vista è abbastanza complessa da richiedere una tabella temporanea, `UPDATE` non è consentito.

Cose come `GROUP BY` , `UNION` , `HAVING` , `DISTINCT` e alcune subquery impediscono l'aggiornamento della vista. Dettagli nel *manuale di riferimento* .

DROPPARE UNA VISTA

- Creare e rilasciare una vista nel database corrente.

```
CREATE VIEW few_rows_from_t1 AS SELECT * FROM t1 LIMIT 10;
DROP VIEW few_rows_from_t1;
```

- Creare e rilasciare una vista che fa riferimento a una tabella in un altro database.

```
CREATE VIEW table_from_other_db AS SELECT x FROM db1.foo WHERE x IS NOT NULL;
DROP VIEW table_from_other_db;
```

Leggi VISTA online: https://riptutorial.com/it/mysql/topic/1489/vista

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con MySQL	A. Raza, Aman Dhanda, Andy, Athafoud, CodeWarrior, Community, Confiqure, Dipen Shah, e4c5, Epodax, Giacomo Garabello, greatwolf, inetphantom, JayRizzo, juergen d, Lahiru Ashan, Lambda Ninja, Magisch, Marek Skiba, Md. Nahiduzzaman Rose, moopet, msohng, Noah van der Aa, O. Jones, OverCoder, Panda, Parth Patel, rap-2-h, rhavendc, Romain Vincent, YCF_L
2	AGGIORNARE	4thfloorstudios, Chris, Drew, Khurram, Ponnarasu, Rick James, Sevle
3	ALTER TABLE	e4c5, JohnLBevan, kolunar, LiuYan , Matas Vaitkevicius, mayojava, Rick James, Steve Chambers, Thuta Aung, WAF, YCF_L
4	Amministratore MySQL	Florian Genser, Matas Vaitkevicius, RationalDev, Rick James
5	apici inversi	Drew, SuperDJ
6	Aritmetica	Barranka, Dinidu, Drew, JonMark Perry, O. Jones, RamenChef, Richard Hamilton, Rick James
7	Backup con mysqldump	agold, Asaph, Barranka, Batsu, KalenGi, Mark Amery, Matthew, mnoronha, Ponnarasu, RamenChef, Rick James , still_learning, strangeqargo, Sumit Gupta, Timothy, WAF
8	Cambia la password	e4c5, Hardik Kanjariya ツ, Rick James, Viktor, ydaetskcoR
9	CARICARE DATI INFIL	aries12, Asaph, bhrached, CGritton, e4c5, RamenChef, Rick James, WAF
10	Client MySQL	Batsu, Nathaniel Ford, Rick James
11	Clustering	Drew, Rick James
12	Codici di errore	Drew, e4c5, juergen d, Lucas Paolillo, O. Jones, Ponnarasu, Rick James, WAF, Wojciech Kazior
13	Commento Mysql	Franck Dernoncourt, Rick James, WAF, YCF_L
14	Configurazione della	4444, a coder, Eugene

	connessione SSL	
15	Configurazione e messa a punto	ChintaMoney, CodeWarrior, Epodax, Eugene, jan_kiran, Rick James
16	Connessione con UTF-8 utilizzando vari linguaggi di programmazione.	Epodax, Rick James
17	Conversione da MyISAM a InnoDB	Ponnarasu, Rick James, yukoff
18	Crea nuovo utente	Aminadav, Batsu, Hardik Kanjariya ツ, josinalvo, Rick James, WAF
19	Creazione della tabella	4444, Alex Shesterov, alex9311, andygeers, Aryo, Asaph, Barranka, Benvorth, Brad Larson, CPHPython, Darwin von Corax, Dinidu, Drew, fedorqui, HCarrasko, Jean Vitor, John M, Matt, Misa Lazovic, Panda, Parth Patel, Paulo Freitas, Přemysl Šťastný, Rick, Rick James, Ronnie Wang , Saroj Sasmal, Sebastian Brosch, skytreader, Stefan Rogin, Strawberry, Timothy, ultrajohn, user6655061, vijaykumar, Vini.g.fer, Vladimir Kovpak, WAF, YCF_L, Yury Fedorov
20	Creazione di database	Daniel Käfer, Drew, Ponnarasu, R.K123, Rick James, still_learning
21	Drop Table	Noah van der Aa, Parth Patel, Ponnarasu, R.K123, Rick James, trf, Tushar patel, YCF_L
22	ELIMINA	Batsu, Drew, e4c5, ForguesR, gabe3886, Khurram, Parth Patel, Ponnarasu, Rick James, strangeqargo, WAF, whrrgarbl, ypercube, Илья Плотников
23	ENUM	Philipp, Rick James
24	Errore 1055: ONLY_FULL_GROUP_BY: qualcosa non è nella clausola GROUP BY ...	Damian Yerrick, O. Jones
25	Espressioni regolari	user2314737, YCF_L
26	Estrai valori dal tipo JSON	MohaMad
27	eventi	Drew, rene
28	Gestione dei fusi orari	O. Jones

29	Gestione di dati sparsi o mancanti	Batsu, Nate Vaughan
30	Indici e chiavi	Alex Recarey, Barranka, Ben Visness, Drew, kolunar, Rick James, Sanjeev kumar
31	Informazioni sul server	FMashiro
32	INSERIRE	0x49D1, AbcAeffchen, Abubakkar, Aukhan, CGritton, Dinidu, Dreamer, Drew, e4c5, fnkr, gabe3886, Horen, Hugo Buff, Ian Kenney, Johan, Magisch, NEER, Parth Patel, Philipp, Rick James, Riho, strangeqargo, Thuta Aung, zeppelin
33	Installa il contenitore Mysql con Docker-Compose	Marc Alff, molavec
34	JSON	A. Raza, Ben, Drew, e4c5, Manatax, Mark Amery, MohaMad, phatfingers, Rick James, sunkuet02
35	Limite e offset	Alvaro Flaño Larrondo, Ani Menon, animuson, ChaoticTwist, Chris Rasys, CPHPython, Ian Gregory, Matt S, Rick James, Sumit Gupta, WAF
36	Log files	Drew, Rick James
37	Motore MyISAM	Rick James
38	MySQL LOCK TABLE	Ponnarasu, Rick James, vijeeshin
39	mysqlimport	Batsu
40	NULLO	Rick James, Sumit Gupta
41	Operazioni di data e ora	Abhishek Aggrawal, Drew, Matt S, O. Jones, Rick James, Sumit Gupta
42	Operazioni di stringa	Abubakkar, Batsu, juergen d, kolunar, Rick James, uruloke, WAF
43	ORDINATO DA	Florian Genser, Rick James
44	Ottimizzazione delle prestazioni	e4c5, RamenChef, Rick James
45	Parole riservate	juergen d, user2314737
46	partizionamento	Majid, Rick James
47	Personalizza PS1	Eugene, Wenzhong

48	PREPARAZIONE delle dichiarazioni	kolunar, Rick James, winter
49	Query pivot	Barranka
50	Raggruppa per	Adam, Filipe Martins, Lijo, Rick James, Thuta Aung, WAF, whrrgarbl
51	Recupera dalla password di root persa	BacLuc, Jen R
52	replicazione	Ponnarasu
53	Ricerca full-text	O. Jones
54	Ripristina e reimposta la password di root predefinita per MySQL 5.7+	Lahiru, ParthaSen
55	Routine memorizzate (procedure e funzioni)	Abhishek Aggrawal, Abubakkar, Darwin von Corax, Dinidu , Drew, e4c5, juergen d, kolunar, llanato, Rick James, userlond
56	SELEZIONARE	Ani Menon, Asjad Athick, Benvorth, Bhavin Solanki, Chip, Drew, greatwolf, Inzimam Tariq IT, julienc, KartikKannapur , Kruti Patel, Matthis Kohli, O. Jones, Ponnarasu, Rick James, SeeuD1, ThisIsImpossible, timmyRS, YCF_L, ypercube
57	Set di caratteri e regole di confronto	frlan, Rick, Rick James
58	Si unisce	Artisan72, Batsu, Benvorth, Bikash P, Drew, Matt, Philipp, Rick, Rick James, user3617558
59	Sicurezza tramite GRANT	Rick James
60	Suggerimenti per le prestazioni Mysql	arushi, RamenChef, Rick James, Rodrigo Darti da Costa
61	Tabella di mappatura molti-a-molti	Rick James
62	Tabella non pivot dinamica con istruzione preparata	rpd
63	Tabelle temporanee	Ponnarasu, Rick James
64	Tempo con precisione al di	O. Jones

	sotto del secondo	
65	Tipi di dati	Batsu, dakab, Drew, Dylan Vander Berg, e4c5, juergen d, MohaMad, Richard Hamilton, Rick James
66	Transazione	Ponnarasu, Rick James
67	TRIGGER	Blag, e4c5, Matas Vaitkevicius, ratchet, WAF, YCF_L
68	UNIONE	Mattew Whitt, Rick James, Riho, Tarik, wangengzheng
69	Unioni MySQL	Ani Menon, Rick James
70	UNISCI: iscriviti a 3 tavoli con lo stesso nome di id.	FMashiro
71	Uno a molti	falsefive
72	Uso delle variabili	kolunar, user6655061
73	VISTA	Abhishek Aggrawal, Divya, e4c5, Marina K., Nikita Kurtin, Ponnarasu, R.K123, ratchet, Rick James, WAF, Yury Fedorov, Илья Плотников