Kostenloses eBook

LERNEN MySQL

Free unaffiliated eBook created from
Stack Overflow contributors.

#mysql

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit MySQL

Bemerkungen

MySQL ist ein Open Source-Relational Database Management System (RDBMS), das von der Oracle Corporation entwickelt und unterstützt wird.

MySQL wird unterstützt von einer großen Anzahl von Plattformen, einschließlich Linux - Varianten, OS X und Windows. Es verfügt auch über APIs für eine Vielzahl von Sprachen, darunter C, C ++, Java, Lua, .Net, Perl, PHP, Python und Ruby.

MariaDB ist eine Abspaltung von MySQL mit einem etwas anderen Funktionssatz . Es ist für die meisten Anwendungen vollständig mit MySQL kompatibel.

Versionen

Ausführung	Veröffentlichungsdatum
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

Ausführung	Veröffentlichungsdatum
5,7	2015-10-01

Examples

Fertig machen

Erstellen einer Datenbank in MySQL

```
CREATE DATABASE mydb;
```

Rückgabewert:

> Abfrage OK, 1 Zeile betroffen (0,05 Sek.)

Verwenden der erstellten Datenbank `mydb`

```
USE mydb;
```

Rückgabewert:

> Datenbank geändert

Erstellen einer Tabelle 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` erstellt eine neue Tabelle namens `mytable` .

`id int unsigned NOT NULL auto_increment` erstellt die `id` Spalte. Dieser `id int unsigned NOT NULL auto_increment` weist jedem Datensatz in der Tabelle eine eindeutige numerische ID zu (dh, in diesem Fall dürfen keine zwei Zeilen dieselbe `id` haben). MySQL weist automatisch eine neue, eindeutiger Wert für das `id` Feld des Datensatzes (beginnend mit 1).

Rückgabewert:

> Abfrage OK, 0 Zeilen betroffen (0,10 Sek.)

Eine Zeile in eine MySQL-Tabelle einfügen

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

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

Beispielrückgabewert:

> Abfrage OK, 1 Zeile betroffen (0,06 Sek.)

Die `varchar strings` können auch mit einfachen Anführungszeichen eingefügt werden:

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

Aktualisieren einer Zeile in einer MySQL-Tabelle

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

Beispielrückgabewert:

> Abfrage OK, 1 Zeile betroffen (0,06 Sek.)

Der `int` Wert kann ohne Anführungszeichen in eine Abfrage eingefügt werden. Strings und Daten müssen in einfachen Anführungszeichen eingeschlossen werden `'` oder doppelte Anführungszeichen `"` .

Eine Zeile in einer MySQL-Tabelle löschen

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

Beispielrückgabewert:

> Abfrage OK, 1 Zeile betroffen (0,06 Sek.)

Dadurch wird die Zeile mit der `id` 8 gelöscht.

Auswählen von Zeilen basierend auf den Bedingungen in MySQL

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

Rückgabewert:

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

> 1 Reihe im Satz (0,00 Sek.)

Liste der vorhandenen Datenbanken anzeigen

```
SHOW databases;
```

Rückgabewert:

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

2 Reihen im Satz (0,00 Sek.)

Sie können sich "information_schema" als eine "Master-Datenbank" vorstellen, die Zugriff auf Datenbank-Metadaten bietet.

Tabellen in einer vorhandenen Datenbank anzeigen

```
SHOW tables;
```

Rückgabewert:

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

1 Reihe im Satz (0,00 Sek.)

Alle Felder einer Tabelle anzeigen

```
DESCRIBE databaseName.tableName;
```

oder, wenn Sie bereits eine Datenbank verwenden:

```
DESCRIBE tableName;
```

Rückgabewert:

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

`Extra` kann zum Beispiel `auto_increment` enthalten.

`Key` bezieht sich auf den Schlüsseltyp, der das Feld beeinflussen kann. Primary (PRI), Unique (UNI) ...

    n Reihe im Satz (0,00 s)

Dabei ist n die Anzahl der Felder in der Tabelle.

Benutzer erstellen

Zuerst müssen Sie einen Benutzer erstellen und dem Benutzer dann Berechtigungen für bestimmte Datenbanken / Tabellen erteilen. Beim Erstellen des Benutzers müssen Sie außerdem angeben, von wo aus sich dieser Benutzer verbinden kann.

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

Erstellt einen Benutzer, der nur eine Verbindung auf dem lokalen Computer herstellen kann, auf dem die Datenbank gehostet wird.

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

Erstellt einen Benutzer, der von überall her eine Verbindung herstellen kann (außer auf dem lokalen Computer).

Beispielrückgabewert:

    Abfrage OK, 0 Zeilen betroffen (0,00 Sek.)

Privilegien hinzufügen

Gewähren Sie dem Benutzer allgemeine, grundlegende Berechtigungen für alle Tabellen der angegebenen Datenbank:

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

Gewähren Sie dem Benutzer alle Berechtigungen für alle Tabellen in allen Datenbanken (Achtung).

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

Wie oben gezeigt, bezieht sich `*.*` alle Datenbanken und Tabellen, `databaseName.*` alle Tabellen der jeweiligen Datenbank. Es ist auch möglich, Datenbank und Tabelle wie `databaseName.tableName` anzugeben.

`WITH GRANT OPTION` sollte ausgelassen werden, wenn der Benutzer anderen Benutzern keine Berechtigungen gewähren muss.

Privilegien können **entweder sein**

```
ALL
```

oder eine Kombination der folgenden, jeweils durch Komma getrennt (nicht erschöpfende Liste).

```
SELECT
INSERT
UPDATE
DELETE
CREATE
DROP
```

Hinweis

Im Allgemeinen sollten Sie versuchen, die Verwendung von Spalten- oder Tabellennamen, die Leerzeichen enthalten, oder die Verwendung von reservierten Wörtern in SQL zu vermeiden. Beispielsweise sollten Sie Namen wie `table` oder `first name` am besten vermeiden.

Wenn Sie solche Namen verwenden müssen, setzen Sie sie zwischen die Trennzeichen "back-tick `` . Zum Beispiel:

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

Eine Abfrage mit den Back-Tick-Trennzeichen in dieser Tabelle könnte lauten:

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

Informationsschema Beispiele

Prozessliste

Dadurch werden alle aktiven und schlafenden Abfragen in dieser Reihenfolge angezeigt.

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

Dies ist ein wenig detaillierter für Zeitfenster, da es standardmäßig in Sekunden ist

```
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;
```

Gespeicherte Prozedur suchen

Durchsuchen Sie alle `Stored Procedures` einfach nach Wörtern und Platzhaltern.

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

Erste Schritte mit MySQL online lesen: https://riptutorial.com/de/mysql/topic/302/erste-schritte-mit-mysql

Kapitel 2: AKTUALISIEREN

Syntax

- UPDATE [LOW_PRIORITY] [IGNORE] Tabellenname SET Spalte1 = Ausdruck1, Spalte2 = Ausdruck2, ... [WHERE-Bedingungen]; // Einfaches Update einer einzelnen Tabelle
- UPDATE [LOW_PRIORITY] [IGNORE] tableName SET Spalte1 = Ausdruck1, Spalte2 = Ausdruck2, ... [WHERE-Bedingungen] [ORDER BY-Ausdruck [ASC | DESC]] [LIMIT row_count]; // Mit order by und limit aktualisieren
- UPDATE [LOW_PRIORITY] [IGNORE] table1, table2, ... SET Spalte1 = Ausdruck1, Spalte2 = Ausdruck2, ... [WHERE-Bedingungen]; // Aktualisierung mehrerer Tabellen

Examples

Grundlegendes Update

Eine Zeile aktualisieren

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

Diese Abfrage aktualisiert den Inhalt der `email` - `customers` `luke_smith@email.com` `id` `email` in der `customers` - Tabelle auf den String `luke_smith@email.com` , wo der Wert von `id` 1. Die alten und neuen Inhalte der Datenbanktabelle gleich sind unten auf der linken Seite dargestellt bzw. rechts:

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

Alle Zeilen werden aktualisiert

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

Diese Abfrage aktualisiert den Inhalt des `lastname` für jeden Eintrag in der `customers` - Tabelle. Der alte und der neue Inhalt der Datenbanktabelle sind links bzw. rechts unten dargestellt:

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

Hinweis: In der UPDATE-Abfrage müssen bedingte Klauseln (WHERE) verwendet werden. Wenn Sie keine Bedingungsklausel verwenden, werden alle Datensätze des Attributs dieser Tabelle aktualisiert. In obigem Beispiel neuer Wert (Smith) von Nachname in Kundentabelle auf alle Zeilen gesetzt.

Aktualisieren Sie mit Join Pattern

Betrachten Sie eine Produktionstabelle namens `questions_mysql` und eine Tabelle `iwtQuestions` (importierte Arbeitstabelle), die den letzten Stapel importierter CSV-Daten aus einer `LOAD DATA INFILE` . Die Arbeitstabelle wird vor dem Import abgeschnitten, die Daten werden importiert und dieser Vorgang wird hier nicht angezeigt.

Aktualisieren Sie unsere Produktionsdaten mithilfe einer Verknüpfung mit unseren importierten Arbeitstischdaten.

```
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;
```

Die Aliase `q` und `i` werden verwendet, um die Tabellenverweise abzukürzen. Dies erleichtert die Entwicklung und Lesbarkeit.

`qId` , der Primärschlüssel, repräsentiert die Stackoverflow-Fragen-ID. Es werden vier Spalten aktualisiert, um passende Zeilen aus dem Join zu finden.

UPDATE mit ORDER BY und LIMIT

Wenn die `ORDER BY` Klausel in Ihrer Update-SQL-Anweisung angegeben ist, werden die Zeilen in der angegebenen Reihenfolge aktualisiert.

Wenn in Ihrer SQL-Anweisung eine `LIMIT` Klausel angegeben ist, wird dadurch die Anzahl der Zeilen, die aktualisiert werden können, begrenzt. Es gibt keine Begrenzung, wenn keine `LIMIT` Klausel angegeben ist.

`ORDER BY` und `LIMIT` können nicht für die Aktualisierung `LIMIT` Tabellen verwendet werden.

Die Syntax für das MySQL- `UPDATE` mit `ORDER BY` und `LIMIT` lautet

```
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
```

Im obigen Beispiel werden 10 Zeilen entsprechend der Reihenfolge der Mitarbeiter aktualisiert, die am `joiningDate` .

Mehrere Tabelle UPDATE

In `UPDATE` mehrere Tabellen werden Zeilen in jeder angegebenen Tabelle aktualisiert, die die Bedingungen erfüllen. Jede übereinstimmende Zeile wird einmal aktualisiert, auch wenn sie mehrmals den Bedingungen entspricht.

In mehreren Tabellen- `UPDATE` können `ORDER BY` und `LIMIT` nicht verwendet werden.

Syntax für Multi Table `UPDATE` ist,

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

Betrachten Sie zum Beispiel zwei Tabellen, `products` und `salesOrders` . In dem Fall reduzieren wir die Menge eines bestimmten Produkts aus dem bereits erteilten Kundenauftrag. Dann müssen wir diese Menge auch in unserer `products` erhöhen. Dies kann in einer einzelnen SQL-Aktualisierungsanweisung wie folgt durchgeführt werden.

```
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;
```

Im obigen Beispiel wird die Menge '5' aus der `salesOrders` Tabelle reduziert und in der `products` entsprechend den `WHERE` Bedingungen erhöht.

Bulk-Update

Bei der Aktualisierung mehrerer Zeilen mit unterschiedlichen Werten ist die Massenaktualisierung viel schneller.

```
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);
```

Bei Massenaktualisierungen kann nur eine Abfrage an den Server gesendet werden, anstatt eine Abfrage für jede zu aktualisierende Zeile. Die Fälle sollten alle möglichen Parameter enthalten, die in der `WHERE` Klausel nachgeschlagen werden.

AKTUALISIEREN online lesen: https://riptutorial.com/de/mysql/topic/2738/aktualisieren

Kapitel 3: ALTER TABELLE

Syntax

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

Bemerkungen

```
    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 CREATE TABLE- `options`)

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

Ref: MySQL 5.7 Referenzhandbuch / ... / ALTER TABLE-Syntax / 14.1.8 ALTER TABLE-Syntax

Examples

Storage Engine wechseln; Tisch neu aufbauen; Ändern Sie file_per_table

Wenn beispielsweise `t1` derzeit keine InnoDB-Tabelle ist, ändert diese Anweisung ihre Speicher-Engine in InnoDB:

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

Wenn die Tabelle bereits InnoDB ist, werden die Tabelle und ihre Indizes neu erstellt und haben einen ähnlichen Effekt wie `OPTIMIZE TABLE` . Möglicherweise verbessern Sie den Speicherplatz.

Wenn sich der Wert von `innodb_file_per_table` aktuell von dem Wert unterscheidet, der wirksam war, als `t1` wurde, wird dies in (oder von) file_per_table konvertiert.

ALTER SPALTE DER TABELLE

```
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
```

ALTER-Tabelle INDEX hinzufügen

Um die Leistung zu verbessern, möchten Sie möglicherweise Spalten zu Spalten hinzufügen

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

Ändern, um zusammengesetzte (mehrere Spalten) Indizes hinzuzufügen

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

Auto-Inkrement-Wert ändern

Das Ändern eines Auto-Inkrement-Werts ist hilfreich, wenn Sie nach einer massiven Löschung keine Lücke in einer AUTO_INCREMENT-Spalte wünschen.

Beispielsweise haben Sie viele unerwünschte (Ankündigungs-) Zeilen in Ihrer Tabelle gepostet, gelöscht, und Sie möchten die Lücke in den Auto-Increment-Werten beheben. Angenommen, der MAX-Wert der Spalte AUTO_INCREMENT ist jetzt 100. Sie können Folgendes verwenden, um den Wert für die automatische Erhöhung festzulegen.

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

Ändern des Typs einer Primärschlüsselspalte

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

Wenn Sie versuchen, den Typ dieser Spalte zu ändern, ohne zuvor den Primärschlüssel zu löschen, führt dies zu einem Fehler.

Spaltendefinition ändern

Um die Definition einer Datenbankspalte zu ändern, kann die folgende Abfrage beispielsweise verwendet werden, wenn wir dieses Datenbankschema haben

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

Um den Typ der `age` von `char` in `int` zu ändern, verwenden wir die folgende Abfrage:

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

Allgemeines Format ist:

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

Umbenennen einer MySQL-Datenbank

Es gibt keinen einzigen Befehl zum Umbenennen einer MySQL-Datenbank. Sie können dies jedoch durch eine einfache Problemumgehung erreichen, indem Sie ein Backup erstellen und wiederherstellen:

```
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>
```

Schritte:

1. Kopieren Sie die obigen Zeilen in einen Texteditor.
2. Ersetzen Sie alle Verweise auf `<old name>` , `<new name>` und `<password>` (+ optional `root` , um einen anderen Benutzer zu verwenden) durch die entsprechenden Werte.
3. Führen Sie nacheinander die Befehlszeile aus (vorausgesetzt, der MySQL-Ordner "bin" befindet sich im Pfad und gibt "y" ein, wenn Sie dazu aufgefordert werden).

Alternative Schritte:

Umbenennen (Verschieben) jeder Tabelle von einer Datenbank in die andere. Tun Sie dies für jede Tabelle:

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

Sie können diese Anweisungen erstellen, indem Sie Folgendes ausführen

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

Warnung. Versuchen Sie nicht, irgendeine Art von Tabelle oder Datenbank auszuführen, indem Sie einfach Dateien im Dateisystem verschieben. Das hat in den alten Tagen von MyISAM gut funktioniert, aber in den neuen Tagen von InnoDB und den Tablespaces wird es nicht funktionieren. Vor allem, wenn das "Data Dictionary" aus dem Dateisystem in System-InnoDB-Tabellen verschoben wird, wahrscheinlich in der nächsten Hauptversion. Um eine `PARTITION` einer InnoDB-Tabelle zu verschieben (im Gegensatz zum `DROPping`), müssen "transportable Tablespaces" verwendet werden. In naher Zukunft wird es nicht einmal eine Akte geben, nach der man greifen kann.

Vertauschen der Namen zweier MySQL-Datenbanken

Die folgenden Befehle können verwendet werden, um die Namen zweier MySQL-Datenbanken (`<db1>` und `<db2>`) `<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
```

Schritte:

1. Kopieren Sie die obigen Zeilen in einen Texteditor.
2. Ersetzen Sie alle Verweise auf `<db1>` , `<db2>` und `<password>` (+ optional `root` , um einen anderen Benutzer zu verwenden) durch die entsprechenden Werte.
3. Führen Sie nacheinander die Befehlszeile aus (vorausgesetzt, der MySQL-Ordner "bin" befindet sich im Pfad und gibt "y" ein, wenn Sie dazu aufgefordert werden).

Umbenennen einer MySQL-Tabelle

Das Umbenennen einer Tabelle kann in einem einzigen Befehl erfolgen:

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

Die folgende Syntax macht genau das gleiche:

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

Wenn Sie eine temporäre Tabelle umbenennen, muss die `ALTER TABLE` Version der Syntax verwendet werden.

Schritte:

1. Ersetzen Sie in der obigen Zeile `<old name>` und `<new name>` durch die entsprechenden Werte. *Anmerkung: Wenn die Tabelle in eine andere Datenbank `dbname` , der `dbname` . `tablename` Syntax kann für `<old name>` und / oder `<new name>` .*
2. Führen Sie es in der entsprechenden Datenbank in der MySQL-Befehlszeile oder auf einem Client wie MySQL Workbench aus. *Hinweis: Der Benutzer muss über ALTER- und DROP-Berechtigungen für die alte Tabelle und CREATE und INSERT für die neue Tabelle verfügen.*

Umbenennen einer Spalte in einer MySQL-Tabelle

Das Umbenennen einer Spalte kann in einer einzigen Anweisung erfolgen, aber neben dem neuen Namen muss auch die "Spaltendefinition" (dh der Datentyp und andere optionale Eigenschaften wie Nullwertigkeit, automatische Inkrementierung usw.) angegeben werden.

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

Schritte:

1. Öffnen Sie die MySQL-Befehlszeile oder einen Client wie MySQL Workbench.
2. Führen Sie die folgende Anweisung aus: `SHOW CREATE TABLE <table name>;` (Ersetzen von

`<table name>` durch den entsprechenden Wert).

3. Notieren Sie sich die gesamte Spaltendefinition für die umzubenennende Spalte *(dh alles, was nach dem Namen der Spalte, aber vor dem Komma, das sie vom nächsten Spaltennamen trennt) erscheint* .
4. Ersetzen Sie in der obigen Zeile `<old name>` , `<new name>` und `<column definition>` durch die entsprechenden Werte und führen Sie sie aus.

ALTER TABELLE online lesen: https://riptutorial.com/de/mysql/topic/2627/alter-tabelle

Kapitel 4: Ändere das Passwort

Examples

Ändern Sie das MySQL-Root-Passwort in Linux

So ändern Sie das root-Benutzerpasswort von MySQL:

Schritt 1: Stoppen Sie den MySQL Server.

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

Schritt 2: Starten Sie den MySQL-Server ohne das Berechtigungssystem.

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

oder wenn `mysqld_safe` nicht verfügbar ist,

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

Schritt 3: Verbinden Sie sich mit dem MySQL-Server.

```
mysql -u root
```

Schritt 4: Legen Sie ein neues Passwort für den Root-Benutzer fest.

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;
```

Anmerkung: Die `ALTER USER` Syntax wurde in MySQL 5.7.6 eingeführt.

Schritt 5: Starten Sie den MySQL Server neu.

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

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

- in CentOS, Fedora oder Red Hat Enterprise Linux:

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

Ändern Sie das MySQL-Root-Passwort in Windows

Wenn Sie das root-Passwort in Windows ändern möchten, müssen wir die folgenden Schritte ausführen:

Schritt 1: Starten Sie die Eingabeaufforderung mit einer der folgenden Methoden:

Perss `Crtl+R` oder Springen `Start Menu > Run` und geben Sie `cmd` ein und drücken Sie die Eingabetaste

Schritt 2: Wechseln Sie in das Verzeichnis, in dem `MYSQL` installiert ist. In meinem Fall ist es das

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

Schritt 3: Nun müssen wir die `mysql` Eingabeaufforderung starten

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

Schritt 4: Abfrage abfragen, um das `root` Passwort zu ändern

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

Verarbeiten

1. Stoppen Sie den MySQL (mysqld) Server / Daemon-Prozess.
2. Starten Sie den MySQL Server-Prozess, um die Option --skip-grant-tables zu aktivieren, damit kein Kennwort `mysqld_safe --skip-grant-tables &` : `mysqld_safe --skip-grant-tables &`
3. Stellen Sie als Root-Benutzer eine Verbindung zum MySQL-Server her: `mysql -u root`
4. Ändere das Passwort:

- (5.7.6 und neuer): `ALTER USER 'root'@'localhost' IDENTIFIED BY 'new-password';`
- (5.7.5 und älter oder MariaDB): `SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new-password); flush privileges; quit;`

5. Starten Sie den MySQL-Server neu.

Hinweis: Dies funktioniert nur, wenn Sie sich physisch auf demselben Server befinden.

Online-Dokument: http://dev.mysql.com/doc/refman/5.7/de/resetting-permissions.html

Ändere das Passwort online lesen: https://riptutorial.com/de/mysql/topic/2761/andere-das-passwort

Kapitel 5: Arithmetik

Bemerkungen

MySQL verwendet auf den meisten Maschinen eine 64-Bit-Gleitkomma-Arithmetik nach IEEE 754 für seine Berechnungen.

In ganzzahligen Kontexten wird die Ganzzahlarithmetik verwendet.

- `RAND()` ist kein perfekter Zufallszahlengenerator. Es wird hauptsächlich zur schnellen Erzeugung von Pseudozufallszahlen verwendet

Examples

Rechenzeichen

MySQL stellt die folgenden Rechenoperatoren zur Verfügung

Operator	Name	Beispiel
`+`	Zusatz	`SELECT 3+5;` -> 8 `SELECT 3.5+2.5;` -> 6,0 `SELECT 3.5+2;` -> 5,5
`-`	Subtraktion	`SELECT 3-5;` -> -2
`*`	Multiplikation	`SELECT 3 * 5;` -> 15
`/`	Einteilung	`SELECT 20 / 4;` -> 5 `SELECT 355 / 113;` -> 3.1416 `SELECT 10.0 / 0;` -> NULL
`DIV`	Integer Division	`SELECT 5 DIV 2;` -> 2
`%` oder `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

Wenn die Zahlen in Ihrer Arithmetik alle Ganzzahlen sind, verwendet MySQL den `BIGINT` (64-Bit-Vorzeichen) für seine Arbeit. Zum Beispiel:

`select (1024 * 1024 * 1024 * 1024 *1024 * 1024) + 1` -> 1,152,921,504,606,846,977

und

`select (1024 * 1024 * 1024 * 1024 *1024 * 1024 * 1024` -> `BIGINT` außerhalb des `BIGINT` Bereichs)

DOPPELT

Wenn die Zahlen in Ihrer Arithmetik gebrochen sind, verwendet MySQL die 64-Bit-Gleitkomma-Arithmetik nach IEEE 754 . Bei der Verwendung der Fließkomma-Arithmetik müssen Sie vorsichtig sein, da viele Fließkommazahlen von Natur aus eher Näherungswerte als genaue Werte sind .

Mathematische Konstanten

Pi

Im Folgenden wird der Wert von `PI` mit 6 Dezimalstellen zurückgegeben. Der tatsächliche Wert ist gut zu `DOUBLE` ;

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

Trigonometrie (SIN, COS)

Winkel sind in Radiant, nicht in Grad. Alle Berechnungen werden in einem 64-Bit-Fließkomma nach IEEE 754 durchgeführt . Alle Gleitkommaberechnungen unterliegen kleinen Fehlern, die als Maschinenfehler (Epsilon) bezeichnet werden . Versuchen Sie daher nicht, sie auf Gleichheit zu vergleichen. Es gibt keine Möglichkeit, diese Fehler bei der Verwendung von Gleitkommazahlen zu vermeiden. Sie sind in die Technologie eingebaut.

Wenn Sie in trigonometrischen Berechnungen `DECIMAL` Werte verwenden, werden diese implizit in Fließkommazahlen und dann wieder in Dezimalzahlen konvertiert.

Sinus

Gibt den Sinus einer Zahl X in Radiant zurück

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

Kosinus

Gibt den Cosinus von X zurück, wenn X im Bogenmaß angegeben wird

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

Tangente

Gibt den Tangens einer Zahl X in Bogenmaß zurück. Beachten Sie, dass das Ergebnis sehr nahe bei Null liegt, aber nicht genau Null. Dies ist ein Beispiel für eine Maschine ε.

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

Arc Cosine (inverser Cosinus)

Gibt den Arcuskosinus von X zurück, wenn X im Bereich von `-1 to 1`

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

Arc Sinus (inverser Sinus)

Gibt den Arcussinus von X zurück, wenn X im Bereich von `-1 to 1`

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

Bogentangens (inverser Tangens)

`ATAN(x)` gibt den Arkustangens einer einzelnen Zahl zurück.

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

`ATAN2(X, Y)` gibt den Arcustangens der beiden Variablen X und Y zurück. Sie ähnelt der Berechnung des Arcustangens von Y / X. Sie ist jedoch numerisch robuster: t funktioniert korrekt, wenn X nahe null ist und die Vorzeichen Beide Argumente werden verwendet, um den Quadranten des Ergebnisses zu bestimmen.

Es `ATAN()` , Formeln so zu schreiben, dass `ATAN2()` anstelle von `ATAN()` wo immer dies möglich ist.

```
 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
```

Kotangens

Gibt den Kotangens von X zurück

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

Umwandlung

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

Rundung (RUNDE, FLOOR, CEIL)

Runden Sie eine Dezimalzahl auf einen ganzzahligen Wert

Für genaue numerische Werte (z. B. `DECIMAL`): Wenn die erste Dezimalstelle einer Zahl 5 oder höher ist, rundet diese Funktion eine Zahl auf die nächste Ganzzahl *von Null* . Wenn diese Dezimalstelle 4 oder niedriger ist, wird diese Funktion auf den nächsten ganzzahligen Wert gerundet, der dem Wert *Null* am nächsten liegt.

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

Für ungefähre numerische Werte (z. B. `DOUBLE`): Das Ergebnis der Funktion `ROUND()` hängt von der C-Bibliothek ab. Bei vielen Systemen bedeutet dies, dass `ROUND()` die *Runde zur nächsten geraden* Regel verwendet:

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

Runden Sie eine Zahl auf

Zum `CEIL()` einer Nummer verwenden Sie entweder die Funktion `CEIL()` oder `CEILING()`

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

Eine Zahl abrunden

Um eine Zahl `FLOOR()` , verwenden Sie die Funktion `FLOOR()`

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

FLOOR und CEIL gehen in Richtung auf / von der Unendlichkeit weg:

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

Runden Sie eine Dezimalzahl auf eine bestimmte Anzahl von Dezimalstellen.

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

Die Diskussion von Auf und Ab und "5" gilt ebenfalls.

Erhöhe eine Zahl auf eine Potenz (POW)

Um eine Zahl `x` auf eine Potenz `y` zu erhöhen, verwenden Sie entweder die Funktionen `POW()` oder `POWER()`

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

Quadratwurzel (SQRT)

Verwenden Sie die `SQRT()` Funktion. Wenn die Zahl negativ ist, wird `NULL` zurückgegeben

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

Zufallszahlen (RAND)

Erzeugen Sie eine Zufallszahl

Verwenden Sie die Funktion `RAND()` um eine pseudozufällige Gleitkommazahl zwischen `0` und `1` zu generieren

Angenommen, Sie haben die folgende Abfrage

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

Dies wird so etwas zurückgeben

ich	RAND ()
1	0,6191438870682
2	0,93845168309142
3	0,83482678498591

Zufallszahl in einem Bereich

Um eine Zufallszahl im Bereich a <= n <= b zu generieren, können Sie die folgende Formel verwenden

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

Beispielsweise wird eine Zufallszahl zwischen 7 und 12 generiert

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

Eine einfache Möglichkeit, die Zeilen in einer Tabelle zufällig zurückzugeben:

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

Dies sind **Pseudozufallszahlen** .

Der Pseudozufallszahlengenerator in MySQL ist nicht kryptographisch sicher. Wenn Sie also MySQL verwenden, um Zufallszahlen zu generieren, die als Geheimnisse verwendet werden sollen, kann ein entschlossener Gegner, der weiß, dass Sie MySQL verwendet haben, Ihre Geheimnisse leichter erraten, als Sie vielleicht glauben.

Absolutwert und Vorzeichen (ABS, SIGN)

Liefert den absoluten Wert einer Zahl

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

Das `sign` einer Zahl vergleicht sie mit 0.

Zeichen	Ergebnis	Beispiel
-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
```

Arithmetik online lesen: https://riptutorial.com/de/mysql/topic/4516/arithmetik

Kapitel 6: AUSSICHT

Syntax

- CREATE VIEW view_name AS SELECT Spaltenname (s) FROM Tabellenname WHERE-Bedingung; /// Einfache Ansichtssyntax
- ERSTELLEN [ODER ERSETZEN] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] [DEFINER = {Benutzer | CURRENT_USER}] [SQL SECURITY {DEFINER | INVOKER}] VIEW view_name [(column_list)] AS select_statement [WITH [CASCADED | LOCAL] CHECK OPTION]; /// Vollständige Ansichtssyntax erstellen
- DROP VIEW [IF EXISTS] [Datenbankname.] Ansichtsname; /// Ansichtssyntax löschen

Parameter

Parameter	Einzelheiten
Ansichtsname	Name der Ansicht
SELECT-Anweisung	SQL-Anweisungen, die in die Ansichten gepackt werden sollen. Es kann eine SELECT-Anweisung sein, um Daten aus einer oder mehreren Tabellen abzurufen.

Bemerkungen

Views sind virtuelle Tabellen und enthalten nicht die zurückgegebenen Daten. Sie können Sie davon abhalten, immer wieder komplexe Abfragen zu schreiben.

- **Bevor eine Ansicht erstellt wird, besteht** ihre Angabe ausschließlich aus einer `SELECT` Anweisung. Die `SELECT` Anweisung darf keine Unterabfrage in der FROM-Klausel enthalten.
- **Sobald eine Ansicht wird** es weitgehend wie eine Tabelle gerade verwendet wird und sein kann `SELECT` ed aus wie ein Tisch.

Sie müssen Ansichten erstellen, wenn Sie einige Spalten Ihrer Tabelle vom anderen Benutzer einschränken möchten.

- Beispiel: In Ihrer Organisation möchten Sie, dass Ihre Manager nur wenige Informationen aus einer Tabelle mit dem Namen "Verkauf" anzeigen. Sie möchten jedoch nicht, dass Ihr Softwareentwickler alle Felder der Tabelle "Verkauf" anzeigen kann. Hier können Sie zwei verschiedene Ansichten für Ihre Manager und Ihre Software-Ingenieure erstellen.

Leistung . `VIEWs` sind syntaktischer Zucker. Die Leistung kann jedoch möglicherweise schlechter sein als die entsprechende Abfrage, bei der die Auswahl der Ansicht eingeklappt ist. Der Optimierer versucht, dies für Sie "einzuklappen", ist jedoch nicht immer erfolgreich. MySQL 5.7.6

enthält einige weitere Verbesserungen im Optimierer. Die Verwendung einer `VIEW` erzeugt jedoch keine *schnellere* Abfrage.

Examples

Erstellen Sie eine Ansicht

Privilegien

Die CREATE VIEW-Anweisung erfordert das CREATE VIEW-Privileg für die Ansicht und einige Privilegien für jede von der SELECT-Anweisung ausgewählte Spalte. Für Spalten, die an anderer Stelle in der SELECT-Anweisung verwendet werden, müssen Sie über das SELECT-Privileg verfügen. Wenn die OR REPLACE-Klausel vorhanden ist, müssen Sie auch über das DROP-Privileg für die Ansicht verfügen. CREATE VIEW erfordert möglicherweise auch das SUPER-Privileg, abhängig vom DEFINER-Wert, wie weiter unten in diesem Abschnitt beschrieben.

Wenn auf eine Ansicht verwiesen wird, erfolgt eine Berechtigungsprüfung.

Eine Ansicht gehört zu einer Datenbank. Standardmäßig wird eine neue Ansicht in der Standarddatenbank erstellt. Verwenden Sie einen vollständig qualifizierten Namen, um die Ansicht explizit in einer bestimmten Datenbank zu erstellen

Zum Beispiel:

db_name.view_name

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Hinweis - In einer Datenbank haben Basistabellen und Views denselben Namespace, sodass eine Basistabelle und eine View nicht denselben Namen haben können.

Ein Blick kann:

- aus vielen Arten von SELECT-Anweisungen erstellt werden
- beziehen Sie sich auf Basistabellen oder andere Ansichten
- Verwenden Sie Joins, UNION und Unterabfragen
- SELECT muss sich nicht einmal auf Tabellen beziehen

Ein anderes Beispiel

Im folgenden Beispiel wird eine Ansicht definiert, in der zwei Spalten aus einer anderen Tabelle sowie ein aus diesen Spalten berechneter Ausdruck ausgewählt werden:

```
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 |
+------+-------+-------+
```

Beschränkungen

- Vor MySQL 5.7.7 darf die SELECT-Anweisung keine Unterabfrage in der FROM-Klausel enthalten.
- Die SELECT-Anweisung kann sich nicht auf Systemvariablen oder benutzerdefinierte Variablen beziehen.
- In einem gespeicherten Programm kann sich die SELECT-Anweisung nicht auf Programmparameter oder lokale Variablen beziehen.
- Die SELECT-Anweisung kann sich nicht auf vorbereitete Anweisungsparameter beziehen.
- Jede Tabelle oder Sicht, auf die in der Definition verwiesen wird, muss vorhanden sein. Nachdem die Ansicht erstellt wurde, ist es möglich, eine Tabelle oder Ansicht zu löschen Die Definition bezieht sich auf. In diesem Fall führt die Verwendung der Ansicht zu einem Fehler. Um eine Sichtdefinition auf Probleme dieser Art zu überprüfen, verwenden Sie die Anweisung CHECK TABLE.
- Die Definition kann sich nicht auf eine TEMPORARY-Tabelle beziehen, und Sie können nicht
  Erstellen Sie eine TEMPORARY-Ansicht.
- Sie können einen Auslöser keiner Sicht zuordnen.
- Aliase für Spaltennamen in der SELECT-Anweisung werden gegen die maximale Spaltenlänge von 64 Zeichen (nicht den maximalen Alias) geprüft
  Länge von 256 Zeichen).
- Ein `VIEW` kann genauso gut wie das entsprechende `SELECT` optimiert werden. Eine Optimierung ist unwahrscheinlich.

Ein Blick aus zwei Tabellen

Eine Ansicht ist am nützlichsten, wenn Daten aus mehr als einer Tabelle abgerufen werden können.

```
CREATE VIEW myview AS
SELECT a.*, b.extra_data FROM main_table a
LEFT OUTER JOIN other_table b
ON a.id = b.id
```

In MySQL werden Ansichten nicht materialisiert. Wenn Sie jetzt die einfache Abfrage `SELECT * FROM myview` ausführen, führt mysql den LEFT JOIN hinter der Szene aus.

Eine einmal erstellte Ansicht kann mit anderen Ansichten oder Tabellen verbunden werden

Aktualisieren einer Tabelle über einen VIEW

Eine `VIEW` verhält sich sehr ähnlich wie eine Tabelle. Obwohl Sie eine Tabelle `UPDATE` können, können Sie eine Sicht in dieser Tabelle möglicherweise nicht aktualisieren. Wenn das `SELECT` in der Ansicht komplex genug ist, um eine temporäre Tabelle zu benötigen, ist `UPDATE` im Allgemeinen nicht zulässig.

Dinge wie `GROUP BY` , `UNION` , `HAVING` , `DISTINCT` und einige Unterabfragen verhindern, dass die Ansicht aktualisierbar ist. Details im *Referenzhandbuch* .

EINE ANSICHT ENTFERNEN

- Erstellen Sie eine Ansicht in der aktuellen Datenbank und legen Sie sie dort ab.

```
CREATE VIEW few_rows_from_t1 AS SELECT * FROM t1 LIMIT 10;
DROP VIEW few_rows_from_t1;
```

- Erstellen und löschen Sie eine Ansicht, die auf eine Tabelle in einer anderen Datenbank verweist.

```
CREATE VIEW table_from_other_db AS SELECT x FROM db1.foo WHERE x IS NOT NULL;
DROP VIEW table_from_other_db;
```

AUSSICHT online lesen: https://riptutorial.com/de/mysql/topic/1489/aussicht

Kapitel 7: Backticks

Examples

Backticks verwenden

Es gibt viele Beispiele, bei denen Backticks in einer Abfrage verwendet werden, aber für viele ist es noch unklar, wann oder wo Backticks `` .

Backticks werden hauptsächlich verwendet, um einen Fehler namens " *MySQL Reserviertes Wort* " zu verhindern. Wenn Sie eine Tabelle in PHPmyAdmin erstellen, werden Sie manchmal mit einer Warnung oder Warnung darüber konfrontiert, dass Sie ein " *MySQL-reserviertes Wort* " verwenden.

Wenn Sie beispielsweise eine Tabelle mit einer Spalte namens " `group` " erstellen, wird eine Warnung angezeigt. Das liegt daran, dass Sie die folgende Abfrage machen können:

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

Um sicherzustellen, dass in Ihrer Abfrage keine Fehlermeldung angezeigt wird, müssen Sie Backticks verwenden, damit die Abfrage wie folgt lautet:

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

Tabelle

Nicht nur Spaltennamen können von Backticks, sondern auch Tabellennamen umgeben sein. Zum Beispiel, wenn Sie mehrere Tabellen `JOIN` müssen.

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

Einfacher zu lesen

Wenn Sie Backticks um Tabellen- und Spaltennamen verwenden, können Sie die Abfrage auch leichter lesen.

Zum Beispiel, wenn Sie gewohnt sind, Abfragen in Kleinbuchstaben zu schreiben:

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

Bitte lesen Sie die MySQL-Handbuchseite mit dem Titel Schlüsselwörter und reservierte Wörter . Die mit einem (R) sind reservierte Wörter. Die anderen sind nur Schlüsselwörter. Die Reservierten erfordern besondere Vorsicht.

Backticks online lesen: https://riptutorial.com/de/mysql/topic/5208/backticks

Kapitel 8: Clustering

Examples

Disambiguation

"MySQL Cluster" Disambiguation ...

- NDB-Cluster - Eine spezialisierte, meist im Arbeitsspeicher befindliche Engine. Nicht weit verbreitet
- Galera Cluster alias Percona XtraDB Cluster alias PXC alias MariaDB mit Galera. - Eine sehr gute Hochverfügbarkeitslösung für MySQL. es geht über die Replikation hinaus.

Siehe einzelne Seiten zu diesen Varianten von "Cluster".

Für "Clustered Index" siehe Seite (n) auf `PRIMARY KEY` .

Clustering online lesen: https://riptutorial.com/de/mysql/topic/5130/clustering

Kapitel 9: Datenbanken erstellen

Syntax

- CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name [create_specification] /// Datenbank erstellen
- DROP {DATABASE | SCHEMA} [IF EXISTS] Datenbankname /// Zum Löschen der Datenbank

Parameter

Parameter	Einzelheiten
DATENBANK ERSTELLEN	Erzeugt eine Datenbank mit dem angegebenen Namen
SCHEMA ERSTELLEN	Dies ist ein Synonym für `CREATE DATABASE`
WENN NICHT EXISTEN	Wird verwendet, um Ausführungsfehler zu vermeiden, wenn die angegebene Datenbank bereits vorhanden ist
create_specification	Optionen für `create_specification` geben Datenbankmerkmale wie `CHARACTER SET` und `COLLATE` (Datenbanksortierung) an.

Examples

Erstellen Sie Datenbanken, Benutzer und Zuschüsse

Erstellen Sie eine Datenbank. Beachten Sie, dass das verkürzte Wort SCHEMA als Synonym verwendet werden kann.

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

Wenn die Datenbank bereits vorhanden ist, wird Fehler 1007 zurückgegeben. Um diesen Fehler zu umgehen, versuchen Sie Folgendes:

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

Ähnlich,

```
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
```

Aufgrund der oben genannten `IF EXISTS` werden häufig DDL-Anweisungen mit `IF EXISTS` .

Man kann eine Datenbank mit einem Standard-Zeichensatz und einer Kollatierung erstellen. Zum Beispiel:

```
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 */ |
+----------+-----------------------------------------------------------------+
```

Sehen Sie Ihre aktuellen Datenbanken an:

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

Legen Sie die derzeit aktive Datenbank fest und sehen Sie einige Informationen:

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

Das obige zeigt das Standardzeichen CHARACTER SET und die Sortierung für die Datenbank.

Erstellen Sie einen Benutzer:

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

Das Obige erstellt einen Benutzer John123, der aufgrund des Platzhalters `%` eine Verbindung mit einem beliebigen Hostnamen herstellen kann. Das Passwort für den Benutzer ist auf 'OpenSesame' gesetzt, das Hash ist.

Und ein anderes erstellen:

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

Zeigen Sie, dass die Benutzer erstellt wurden, indem Sie die spezielle `mysql` Datenbank untersuchen:

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

```
| user    | host | password                                  |
+---------+------+-------------------------------------------+
| John123 | %    | *E6531C342ED87 .....................      |
| John456 | %    | *B04E11FAAAE9A .....................      |
+---------+------+-------------------------------------------+
```

Beachten Sie, dass die Benutzer zu diesem Zeitpunkt erstellt wurden, jedoch keine Berechtigung zur Verwendung der Baseball-Datenbank haben.

Arbeiten Sie mit Berechtigungen für Benutzer und Datenbanken. Gewähren Sie dem Benutzer John123 die Berechtigung, über vollständige Berechtigungen für die Baseball-Datenbank zu verfügen, und nur die SELECT-Rechte für den anderen Benutzer:

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

Überprüfen Sie das oben:

```
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'@'%'
|
+-----------------------------------------------------------------------------------------------------
-----------+
```

Beachten Sie, dass die `GRANT USAGE` , die Sie immer sehen werden, lediglich bedeutet, dass sich der Benutzer anmelden kann. Das ist alles was das bedeutet.

MeineDatenbank

Sie *müssen* Ihre eigene Datenbank erstellen und keine Schreibzugriff auf die vorhandenen Datenbanken verwenden. Dies ist wahrscheinlich eines der ersten Dinge, die Sie nach dem ersten Verbindungsaufbau tun sollten.

```
CREATE DATABASE my_db;
USE my_db;
CREATE TABLE some_table;
INSERT INTO some_table ...;
```

Sie können auf Ihre Tabelle verweisen, indem Sie sich mit dem Datenbanknamen qualifizieren: `my_db.some_table` .

Systemdatenbanken

Die folgenden Datenbanken stehen für MySQL zur Verfügung. Sie können sie lesen (`SELECT`), aber Sie dürfen die Tabellen nicht darin schreiben (`INSERT` / `UPDATE` / `DELETE`). (Es gibt einige Ausnahmen.)

- `mysql` - Repository für `GRANT` Informationen und einige andere Dinge.
- `information_schema` - Die Tabellen hier sind "virtuell" in dem Sinne, dass sie sich tatsächlich durch In-Memory-Strukturen manifestieren. Ihr Inhalt enthält das Schema für alle Tabellen.
- `performance_schema` - ?? [bitte akzeptieren, dann bearbeiten]
- Andere?? (für MariaDB, Galera, TokuDB usw.)

Anlegen und Auswählen einer Datenbank

Wenn der Administrator beim Einrichten der Berechtigungen Ihre Datenbank für Sie erstellt, können Sie sie verwenden. Ansonsten müssen Sie es selbst erstellen:

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

Unter Unix ist bei Datenbanknamen die Groß- und Kleinschreibung zu beachten (im Gegensatz zu SQL-Schlüsselwörtern). Sie müssen daher immer als Menagerie auf Ihre Datenbank verweisen, nicht als Menagerie, MENAGERIE oder eine andere Variante. Dies gilt auch für Tabellennamen. (Unter Windows gilt diese Einschränkung nicht, obwohl Sie in einer Abfrage immer auf Datenbanken und Tabellen mit dem gleichen Buchstaben achten müssen. Aus verschiedenen Gründen empfiehlt es sich jedoch, immer den gleichen Buchstaben zu verwenden, der verwendet wurde Die Datenbank wurde erstellt.)

Durch das Erstellen einer Datenbank wird sie nicht zur Verwendung ausgewählt. Sie müssen das explizit tun. Verwenden Sie diese Anweisung, um die Menagerie zur aktuellen Datenbank zu machen:

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

Ihre Datenbank muss nur einmal erstellt werden. Sie müssen sie jedoch jedes Mal auswählen, wenn Sie eine mysql-Sitzung starten. Sie können dies tun, indem Sie eine USE-Anweisung ausgeben, wie im Beispiel gezeigt. Alternativ können Sie die Datenbank in der Befehlszeile auswählen, wenn Sie mysql aufrufen. Geben Sie einfach den Namen nach den Verbindungsparametern an, die Sie möglicherweise angeben müssen. Zum Beispiel:

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

Datenbanken erstellen online lesen: https://riptutorial.com/de/mysql/topic/600/datenbanken-erstellen

Kapitel 10: Datentypen

Examples

Implizites / automatisches Casting

```
select '123' * 2;
```

Um die **Multiplikation** mit `2` durchzuführen, wandelt MySQL die Zeichenfolge `123` automatisch in eine Zahl um.

Rückgabewert:

> 246

Die Umwandlung in eine Nummer beginnt von links nach rechts. Wenn die Konvertierung nicht möglich ist, lautet das Ergebnis `0`

```
select '123ABC' * 2
```

Rückgabewert:

> 246

```
select 'ABC123' * 2
```

Rückgabewert:

> 0

VARCHAR (255) - oder nicht

Vorgeschlagene max len

Zuerst werde ich einige gängige Zeichenfolgen erwähnen, die immer hexadezimal sind oder sonst auf ASCII beschränkt sind. Für diese sollten Sie `CHARACTER SET ascii` (`latin1` ist ok) angeben, damit kein Platz verschwendet wird:

```
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
```

Warum nicht einfach 255? Es gibt zwei Gründe, die übliche Verwendung von (255) für alles zu vermeiden.

- Wenn ein komplexes `SELECT` eine temporäre Tabelle erstellen muss (für eine Unterabfrage, `UNION` , `GROUP BY` usw.), wird die Verwendung der `MEMORY` Engine bevorzugt, die die Daten in den RAM-Speicher setzt. `VARCHARs` werden dabei jedoch in `CHAR` . Dadurch nimmt `VARCHAR(255) CHARACTER SET utf8mb4` 1020 Byte an. Dies kann dazu führen, dass auf die Festplatte verschüttet werden muss, was langsamer ist.
- In bestimmten Situationen prüft InnoDB die mögliche Größe der Spalten in einer Tabelle und entscheidet, dass diese zu groß ist, und eine `CREATE TABLE` wird abgebrochen.

VARCHAR versus **TEXT**

Hinweise zur Verwendung von `*TEXT` , `CHAR` und `VARCHAR` sowie einige `CHAR VARCHAR` :

- Verwenden `TINYTEXT` niemals `TINYTEXT` .
- Verwenden Sie fast nie `CHAR` - es ist eine feste Länge; Jedes Zeichen ist die maximale Länge des `CHARACTER SET` (zB 4 Bytes / Zeichen für utf8mb4).
- Verwenden Sie bei `CHAR CHARACTER SET ascii` sofern Sie nichts anderes wissen.
- `VARCHAR(n)` wird bei n *Zeichen* abgeschnitten. `TEXT` wird bei einigen *Bytes* abgeschnitten. (Aber willst du abgeschnitten werden?)
- `*TEXT` *kann* komplexe `SELECTs` verlangsamen, `SELECTs` Temp-Tabellen behandelt werden.

INT als AUTO_INCREMENT

Für `AUTO_INCREMENT` kann eine beliebige Größe von `INT` verwendet werden. `UNSIGNED` ist immer angemessen.

Denken Sie daran, dass bestimmte Operationen `AUTO_INCREMENT` IDs "verbrennen". Dies könnte zu einer unerwarteten Lücke führen. Beispiele: `INSERT IGNORE` und `REPLACE` . Sie *können* eine ID vorbelegen, *bevor sie* feststellen, dass sie nicht benötigt wird. Dies ist das erwartete Verhalten und Design der InnoDB-Engine und sollte deren Verwendung nicht abschrecken.

Andere

Es gibt bereits einen separaten Eintrag für "FLOAT, DOUBLE und DECIMAL" und "ENUM". Eine einzige Seite mit Datentypen ist wahrscheinlich unhandlich. Ich schlage vor, "Feldtypen" (oder sollte "Datentypen" heißen?), Um eine Übersicht zu erhalten.

- INTs
- FLOAT, DOUBLE und DECIMAL
- Zeichenfolgen (Zeichen, Text usw.)
- BINARY und BLOB
- DATETIME, TIMESTAMP und Freunde
- ENUM und SET
- Räumliche Daten

- JSON-Typ (MySQL 5.7.8 und höher)
- So stellen Sie Money und andere gängige Typen dar, die in vorhandene Datentypen eingebettet werden müssen

Gegebenenfalls sollte jede Themenseite neben Syntax und Beispielen Folgendes enthalten:

- Überlegungen beim ALTERing
- Größe (Bytes)
- Kontrast zu Nicht-MySQL-Engines (niedrige Priorität)
- Überlegungen zur Verwendung des Datentyps in einem PRIMARY KEY oder einem sekundären Schlüssel
- andere Best Practice
- andere Performance-Probleme

(Ich gehe davon aus, dass dieses "Beispiel" sich selbst zerstören wird, wenn meine Vorschläge erfüllt oder ein Veto eingelegt wurden.)

Einleitung (numerisch)

MySQL bietet eine Reihe verschiedener numerischer Typen. Diese können in zerlegt werden

Gruppe	Typen
Integer-Typen	`INTEGER` , `INT` , `SMALLINT` , `TINYINT` , `MEDIUMINT` , `BIGINT`
Feste Punkttypen	`DECIMAL` , `NUMERIC`
Fließkomma-Typen	`FLOAT` , `DOUBLE`
Bitwerttyp	`BIT`

Integer-Typen

Minimaler vorzeichenloser Wert ist immer 0.

Art	Lager (Bytes)	Mindestwert (Unterzeichnet)	Höchster Wert (Unterzeichnet)	Höchster Wert (Ohne Vorzeichen)
`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
`INT`	4	-2^{31}	$2^{31}-1$	$2^{32}-1$

Art	Lager (Bytes)	Mindestwert (Unterzeichnet)	Höchster Wert (Unterzeichnet)	Höchster Wert (Ohne Vorzeichen)
		-2.147.483.648	2,147,483,647	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 18,446,744,073,709,5

Feste Punkttypen

Die `DECIMAL` und `NUMERIC` Typen von MySQL speichern genaue numerische Datenwerte. Es wird empfohlen, diese Typen zu verwenden, um die exakte Genauigkeit zu erhalten, z. B. für Geld.

Dezimal

Diese Werte werden im Binärformat gespeichert. In einer Spaltendeklaration sollten die Genauigkeit und der Maßstab angegeben werden

Präzision steht für die Anzahl der signifikanten Stellen, die für Werte gespeichert werden.

Skala steht für die Anzahl der *nach* der Dezimalstelle gespeicherten Ziffern

```
salary DECIMAL(5,2)
```

5 steht für die `precision` und 2 für die `scale` . In diesem Beispiel liegt der Wertebereich, der in dieser Spalte gespeichert werden kann, zwischen `-999.99 to 999.99`

Wenn der Waagenparameter weggelassen wird, ist der Standardwert 0

Dieser Datentyp kann bis zu 65 Ziffern speichern.

Die Anzahl der von `DECIMAL(M,N)` Bytes beträgt *ungefähr* `M/2` .

Fließkomma-Typen

`FLOAT` und `DOUBLE` repräsentieren *ungefähre* Datentypen.

Art	Lager	Präzision	Angebot
SCHWEBEN	4 Bytes	23 signifikante Bits / ~ 7 Dezimalstellen	10 ^ + / - 38
DOPPELT	8 Bytes	53 signifikante Bits / ~ 16 Dezimalstellen	10 ^ + / - 308

`REAL` ist ein Synonym für `FLOAT` . `DOUBLE PRECISION` ist ein Synonym für `DOUBLE` .

Obwohl MySQL erlaubt auch (M, D) , Qualifier, verwenden Sie es *nicht.* (M, D) bedeutet, dass Werte mit bis zu M Gesamtziffern gespeichert werden können, wobei D nach dem Dezimalzeichen

liegen kann. *Zahlen werden zweimal gerundet oder abgeschnitten; Dies verursacht mehr Probleme als Nutzen.*

Da Gleitkommawerte nur Näherungswerte sind und nicht als exakte Werte gespeichert werden, können Versuche, sie bei Vergleichen als exakt zu behandeln, zu Problemen führen. Beachten Sie insbesondere, dass ein `FLOAT` Wert selten einem `DOUBLE` Wert entspricht.

Bitwerttyp

Der `BIT` Typ ist zum Speichern von `BIT` nützlich. `BIT(M)` erlaubt das Speichern von bis zu M-Bit-Werten, wobei M im Bereich von `1 to 64`

Sie können Werte auch mit `bit value` angeben.

```
b'111'      -> 7
b'10000000' -> 128
```

Manchmal ist es praktisch, "shift" zu verwenden, um einen Einzelbit-Wert zu `(1 << 7)` , beispielsweise `(1 << 7)` für 128.

Die maximale Gesamtgröße aller BIT-Spalten in einer `NDB` Tabelle beträgt 4096.

CHAR (n)

`CHAR(n)` ist eine Zeichenfolge mit einer *festen* Länge von `n` *Zeichen* . Wenn es sich um `CHARACTER SET utf8mb4` , belegt dies genau `4*n` *Bytes* , unabhängig davon, welcher Text darin enthalten ist.

Bei den meisten Anwendungsfällen für `CHAR(n)` um Zeichenfolgen, die englische Zeichen enthalten. Daher sollten sie `CHARACTER SET ascii` . (`latin1` wird genauso gut sein.)

```
country_code CHAR(2) CHARACTER SET ascii,
postal_code  CHAR(6) CHARACTER SET ascii,
uuid    CHAR(39) CHARACTER SET ascii,  -- more discussion elsewhere
```

DATE, DATETIME, TIMESTAMP, YEAR und TIME

Der `DATE` Datentyp umfasst das Datum, jedoch keine Zeitkomponente. Das Format ist `'YYYY-MM-DD'` mit einem Bereich von '1000-01-01' bis '9999-12-31'.

Der `DATETIME` Typ enthält die Uhrzeit im Format 'JJJJ-MM-TT HH: MM: SS'. Es hat einen Bereich von '1000-01-01 00:00:00' bis '9999-12-31 23:59:59'.

Der `TIMESTAMP` Typ ist ein ganzzahliger Typ, der Datum und Uhrzeit mit einem Wirkungsbereich von '1970-01-01 00:00:01' UTC bis '2038-01-19 03:14:07' UTC umfasst.

Der `YEAR` Typ steht für ein Jahr und reicht von 1901 bis 2155.

Der `TIME` Typ repräsentiert eine Zeit mit einem Format von 'HH: MM: SS' und enthält einen Bereich von '-838: 59: 59' bis '838: 59: 59'.

Lagerungssansprüche:

```
|-----------|--------------------|----------------------------------------|
| 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  |
|-----------|--------------------|----------------------------------------|
```

Sekundenbruchteile (ab Version 5.6.4):

```
|------------------------------|------------------|
| Fractional Seconds Precision | Storage Required |
|------------------------------|------------------|
|              0               |      0 bytes     |
|              1,2             |      1 byte      |
|              3,4             |      2 byte      |
|              5,6             |      3 byte      |
|------------------------------|------------------|
```

Siehe die MySQL-Handbuchseiten DATE, DATETIME und TIMESTAMP , Datenspeicheranforderungen und Sekundenbruchteile in Zeitwerten .

Datentypen online lesen: https://riptutorial.com/de/mysql/topic/4137/datentypen

Kapitel 11: Datum und Uhrzeit

Examples

Jetzt()

```
Select Now();
```

Zeigt das aktuelle Datum und die Uhrzeit des Servers an.

```
Update `footable` set mydatefield = Now();
```

Dadurch wird das Feld `mydatefield` mit dem aktuellen Datum und der Uhrzeit des Servers in der konfigurierten Zeitzone des Servers aktualisiert, z

```
'2016-07-21 12:00:00'
```

Datumsberechnung

```
NOW() + INTERVAL 1 DAY  -- This time tomorrow

CURDATE() - INTERVAL 4 DAY -- Midnight 4 mornings ago
```

Zeigt die gespeicherten MySQL-Fragen an, die vor 3 bis 10 Stunden (vor 180 bis 600 Minuten) gestellt wurden:

```
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 |
| ...      |                     |            |
+----------+---------------------+------------+
```

MySQL-Handbuchseiten für `TIMESTAMPDIFF()` .

Achtung Versuchen Sie nicht, Ausdrücke wie `CURDATE() + 1` für die `CURDATE() + 1` in MySQL zu

verwenden. Sie geben nicht das zurück, was Sie erwarten, besonders wenn Sie an das Oracle-Datenbankprodukt gewöhnt sind. Verwenden `CURDATE() + INTERVAL 1 DAY` stattdessen `CURDATE() + INTERVAL 1 DAY` .

Testen gegen einen Datumsbereich

Obwohl es sehr verlockend ist, `BETWEEN` ... `AND` ... für einen Zeitraum zu verwenden, ist es problematisch. Stattdessen vermeidet dieses Muster die meisten Probleme:

```
WHERE x >= '2016-02-25'
  AND x  < '2016-02-25' + INTERVAL 5 DAY
```

Vorteile:

- `BETWEEN` ist dabei inklusive, einschließlich des Enddatums oder des zweiten Datums.
- `23:59:59` ist unbeholfen und falsch, wenn Sie eine Auflösung im Mikrosekundenbereich von `DATETIME` .
- Dieses Muster vermeidet den Umgang mit Schaltjahren und anderen Datenberechnungen.
- Es funktioniert, ob `x` `DATE` , `DATETIME` oder `TIMESTAMP` .

SYSDATE (), JETZT (), CURDATE ()

```
  SELECT SYSDATE();
```

Diese Funktion gibt das aktuelle Datum und die aktuelle Uhrzeit als Wert im Format `'YYYY-MM-DD HH:MM:SS'` oder `YYYYMMDDHHMMSS` , je nachdem, ob die Funktion in einem String oder einem numerischen Kontext verwendet wird. Es gibt Datum und Uhrzeit in der aktuellen Zeitzone zurück.

```
  SELECT NOW();
```

Diese Funktion ist ein Synonym für `SYSDATE()` .

```
  SELECT CURDATE();
```

Diese Funktion gibt das aktuelle Datum ohne Zeit als Wert im Format `'YYYY-MM-DD'` oder `YYYYMMDD` , je nachdem, ob die Funktion in einem String oder in einem numerischen Kontext verwendet wird. Es gibt das Datum in der aktuellen Zeitzone zurück.

Datum aus angegebenem Datum oder DateTime-Ausdruck extrahieren

```
SELECT DATE('2003-12-31 01:02:03');
```

Die Ausgabe wird sein:

```
2003-12-31
```

Verwenden eines Index für eine Datums- und Uhrzeitsuche

Viele reale Datenbanktabellen haben viele Zeilen mit `DATETIME` `TIMESTAMP` Spaltenwerten, die viel Zeit umfassen, einschließlich Jahren oder sogar Jahrzehnten. Häufig ist es erforderlich, eine `WHERE` Klausel zu verwenden, um einen Teil dieser Zeitspanne abzurufen. Beispielsweise möchten wir Zeilen für das Datum 1. September 2016 aus einer Tabelle abrufen.

Ein ineffizienter Weg, dies zu tun, ist folgendes:

```
WHERE DATE(x) = '2016-09-01'   /* slow! */
```

Es ist ineffizient, weil es eine Funktion - `DATE()` - auf die Werte einer Spalte anwendet. Das bedeutet, MySQL muss jeden Wert von `x` prüfen, und ein Index kann nicht verwendet werden.

Ein besserer Weg, um die Operation durchzuführen, ist dies

```
WHERE x >= '2016-09-01'
  AND x <  '2016-09-01' + INTERVAL 1 DAY
```

Dieser wählt einen Wertebereich von `x` einer beliebigen Stelle auf dem betreffenden Tag liegt, bis aber *nicht einschließlich* (daher `<`) Mitternacht am nächsten Tag.

Wenn die Tabelle einen Index für die Spalte `x` , kann der Datenbankserver eine Bereichsüberprüfung für den Index durchführen. Das heißt, es kann den ersten relevanten Wert von x schnell finden und dann den Index nacheinander durchsuchen, bis der letzte relevante Wert gefunden wird. Ein Indexbereichsscan ist wesentlich effizienter als der für `DATE(x) = '2016-09-01` erforderliche vollständige Tabellenscan.

Seien Sie nicht versucht, dies zu verwenden, auch wenn es effizienter aussieht.

```
 WHERE x BETWEEN '2016-09-01' AND '2016-09-01' + INTERVAL 1 DAY  /*  wrong! */
```

Es hat dieselbe Effizienz wie der Entfernungsscan, aber es werden Zeilen mit `x` Werten ausgewählt, die genau um Mitternacht am 2. September 2016 fallen. Dies ist nicht das, was Sie wollen.

Datum und Uhrzeit online lesen: https://riptutorial.com/de/mysql/topic/1882/datum-und-uhrzeit

Kapitel 12: Dynamische Un-Pivot-Tabelle mit Prepared-Anweisung

Examples

Deaktivieren Sie einen dynamischen Satz von Spalten basierend auf den Bedingungen

Das folgende Beispiel ist eine sehr nützliche Grundlage, wenn Sie versuchen, Transaktionsdaten aus Gründen des BI / Reporting in nicht geschwenkte Daten zu konvertieren, wobei die nicht geschwenkten Dimensionen über eine dynamische Spaltengruppe verfügen.

In unserem Beispiel nehmen wir an, dass die Rohdatentabelle Mitarbeiterbeurteilungsdaten in Form markierter Fragen enthält.

Die Rohdatentabelle ist folgende:

```
create table rawdata

(
 PersonId VARCHAR(255)
,Question1Id INT(11)
,Question2Id INT(11)
,Question3Id INT(11)
)
```

Die Rohdatentabelle ist eine temporäre Tabelle im Rahmen des ETL-Verfahrens und kann eine unterschiedliche Anzahl von Fragen haben. Ziel ist es, für eine beliebige Anzahl von Fragen das gleiche Pivotierungsverfahren zu verwenden, d. H.

Unten ist ein Spielzeugbeispiel für eine Rohdatentabelle:

PersonId	Question1Id	Question2Id	Question3Id
Giannaros	1	3	1
Patra	2	4	3

Die bekannte, statische Methode zum Aufheben der Pivotierung der Daten in MySQL ist die Verwendung von 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;
```

In unserem Fall möchten wir einen Weg definieren, um eine beliebige Anzahl von QuestionId-Spalten aufzuheben. Dazu müssen wir eine vorbereitete Anweisung ausführen, die eine dynamische Auswahl der gewünschten Spalten ist. Um auswählen zu können, welche Spalten nicht geschwenkt werden müssen, verwenden wir eine GROUP_CONCAT-Anweisung und wählen die Spalten aus, für die der Datentyp auf 'int' gesetzt ist. In der GROUP_CONCAT enthalten wir auch alle zusätzlichen Elemente unserer auszuführenden SELECT-Anweisung.

```
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;
```

Bei einer anderen Gelegenheit hätten wir Spalten auswählen können, deren Spaltenname beispielsweise einem Muster entspricht

```
DATA_TYPE = 'Int'
```

benutzen

```
COLUMN_NAME LIKE 'Question%'
```

oder etwas geeignetes, das durch die ETL-Phase gesteuert werden kann.

Die vorbereitete Aussage wird wie folgt abgeschlossen:

```
set @temp3 = null;

select concat('INSERT INTO unpivoteddata',@temp2) INTO @temp3;

select @temp3;

prepare stmt FROM @temp3;
execute stmt;
deallocate prepare stmt;
```

Die unpivoteddata-Tabelle sieht folgendermaßen aus:

```
SELECT * FROM unpivoteddata
```

PersonId	QuestionId	QuestionValue
Giannaros	Question1Id	1
Patra	Question1Id	2
Giannaros	Question2Id	3
Patra	Question2Id	4
Giannaros	Question3Id	1
Patra	Question3Id	3

Das Auswählen von Spalten nach einer Bedingung und das Erstellen einer vorbereiteten Anweisung ist eine effiziente Methode, um Daten dynamisch zu entfernen.

Dynamische Un-Pivot-Tabelle mit Prepared-Anweisung online lesen:
https://riptutorial.com/de/mysql/topic/6491/dynamische-un-pivot-tabelle-mit-prepared-anweisung

Kapitel 13: EINFÜGEN

Syntax

1. INSERT [LOW_PRIORITY | VERZÖGERT | HIGH_PRIORITY] [IGNORE] [INTO] Tabellenname [PARTITION (Partitionsname, ...)] [(Spaltenname, ...)] {VALUES | VALUE} ({expr | DEFAULT}, ...), (...), ... [ON DUPLICATE KEY UPDATE Spaltenname = Ausdruck [, Spaltenname = Ausdruck] ...]

2. INSERT [LOW_PRIORITY | VERZÖGERT | HIGH_PRIORITY] [IGNORE] [INTO] Tabellenname [PARTITION (Partitionsname, ...)] SET Col_Name = {Ausdruck | DEFAULT}, ... [ON DUPLICATE KEY UPDATE Spaltenname = Ausdruck [, Spaltenname = Ausdruck] ...]

3. INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE] [INTO] Tabellenname [PARTITION (Partitionsname, ...)] [(Spaltenname, ...)] SELECT ... [ON DUPLICATE KEY UPDATE Spaltenname = Ausdruck [, Spaltenname = Ausdruck] ...]

4. Ein Ausdruck expr kann auf jede Spalte verweisen, die zuvor in einer Werteliste festgelegt wurde. Dies ist beispielsweise möglich, weil der Wert für col2 sich auf col1 bezieht, das zuvor zugewiesen wurde:
   INSERT INTO tbl_name (col1, col2) VALUES (15, col1 * 2);

5. INSERT-Anweisungen, die die VALUES-Syntax verwenden, können mehrere Zeilen einfügen. Fügen Sie dazu mehrere Listen von Spaltenwerten hinzu, die jeweils in Klammern stehen und durch Kommas getrennt sind. Beispiel:
   INSERT in tbl_name (a, b, c) WERTE (1,2,3), (4,5,6), (7,8,9);

6. Die Werteliste für jede Zeile muss in Klammern stehen. Die folgende Anweisung ist ungültig, da die Anzahl der Werte in der Liste nicht mit der Anzahl der Spaltennamen übereinstimmt:
   INSERT in tbl_name (a, b, c) WERTE (1,2,3,4,5,6,7,8,9);

7. INSERT ... SELECT Syntax
   INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE] [INTO] Tabellenname [PARTITION (Partitionsname, ...)] [(Spaltenname, ...)] SELECT ... [ON DUPLICATE KEY UPDATE Spaltenname = Ausdruck, ...]

8. Mit INSERT ... SELECT können Sie schnell viele Zeilen aus einer oder mehreren Tabellen in eine Tabelle einfügen. Zum Beispiel:
   INSERT INTO tbl_temp2 (fld_id) SELECT tbl_temp1.fld_order_id FROM tbl_temp1 WHERE tbl_temp1.fld_order_id> 100;

Bemerkungen

Offizielle INSERT-Syntax

Examples

Grundeinsatz

```
INSERT INTO `table_name` (`field_one`, `field_two`) VALUES ('value_one', 'value_two');
```

In diesem `table_name` Beispiel ist `table_name` wo die Daten hinzugefügt werden sollen, `field_one` und `field_two` sind Felder, für die Daten festgelegt werden sollen, und `value_one` und `value_two` sind die Daten, die gegen `field_one` bzw. `field_two` sind.

Es ist empfehlenswert, die Felder, in die Sie Daten einfügen, in Ihren Code aufzulisten. Wenn sich die Tabelle ändert und neue Spalten hinzugefügt werden, würde das Einfügen brechen, falls sie nicht vorhanden sind

INSERT, 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`);
```

Dies wird `INSERT` in `table_name` die angegebenen Werte, aber wenn der eindeutige Schlüssel bereits vorhanden ist , wird es die Aktualisierung `other_field_1` einen neuen Wert zu haben.
Beim Aktualisieren von doppelten Schlüsseln kann es manchmal nützlich sein, `VALUES()` zu verwenden, um auf den ursprünglichen Wert zuzugreifen, der an `INSERT` anstatt den Wert direkt `VALUES()` . Auf diese Weise können Sie mit `INSERT` und `UPDATE` verschiedene Werte einstellen. Siehe das Beispiel oben , wo `other_field_1` eingestellt ist `insert_value` auf `INSERT` oder `update_value` auf `UPDATE` während `other_field_2` immer gesetzt `other_value` .

Entscheidend für die Funktion "Insert on Duplicate Key Update" (IODKU) ist das Schema, das einen eindeutigen Schlüssel enthält, der einen doppelten Konflikt signalisiert. Dieser eindeutige Schlüssel kann ein Primärschlüssel sein oder nicht. Es kann sich dabei um einen eindeutigen Schlüssel in einer einzelnen Spalte oder um einen mehrspaltigen Schlüssel (zusammengesetzter Schlüssel) handeln.

Mehrere Zeilen einfügen

```
INSERT INTO `my_table` (`field_1`, `field_2`) VALUES
    ('data_1', 'data_2'),
    ('data_1', 'data_3'),
    ('data_4', 'data_5');
```

Dies ist eine einfache Möglichkeit, mit einer `INSERT` Anweisung mehrere Zeilen gleichzeitig hinzuzufügen.

Diese Art von 'Batch'-Insert ist viel schneller als das Einfügen von Zeilen nacheinander. Normalerweise ist das Einfügen von 100 Zeilen in eine einzelne Batch-Insertion 10-mal so schnell wie das Einfügen aller Zeilen.

Vorhandene Zeilen ignorieren

Beim Importieren großer Datensätze kann es unter bestimmten Umständen wünschenswert sein, Zeilen zu überspringen, die normalerweise dazu führen, dass die Abfrage aufgrund einer Spaltenbeschränkung fehlschlägt, z. Dies kann mit `INSERT IGNORE` .

Betrachten Sie die folgende Beispieldatenbank:

```
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  |
+----+--------+
```

Es ist wichtig zu wissen, dass *INSERT IGNORE* auch andere Fehler im *Hintergrund* überspringt. Hier heißt es in den offiziellen Dokumentationen von Mysql:

> Datenkonvertierungen, die Fehler auslösen würden, bricht die Anweisung ab, wenn IGNORE> nicht angegeben wird. Mit IGNORE werden ungültige Werte an die nächsten Werte angepasst und> eingefügt. Es werden Warnungen ausgegeben, die Anweisung bricht jedoch nicht ab.

Hinweis: - Der folgende Abschnitt wird der Vollständigkeit halber hinzugefügt, er gilt jedoch nicht als bewährte Methode (dies würde beispielsweise fehlschlagen, wenn eine weitere Spalte in die Tabelle eingefügt wurde).

Wenn Sie den Wert der entsprechenden Spalte für alle Spalten in der Tabelle angeben, können Sie die Spaltenliste in der `INSERT` Anweisung wie folgt ignorieren:

```
INSERT INTO `my_table` VALUES
        ('data_1', 'data_2'),
```

```
        ('data_1', 'data_3'),
        ('data_4', 'data_5');
```

INSERT SELECT (Einfügen von Daten aus einer anderen Tabelle)

Dies ist die grundlegende Methode zum Einfügen von Daten aus einer anderen Tabelle mit der SELECT-Anweisung.

```
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`;
```

Sie können `SELECT * FROM` , aber dann `tableA` und `tableB` *müssen* Spaltenanzahl passende und Datentypen entsprechen.

Spalten mit `AUTO_INCREMENT` werden wie in der `INSERT` with `VALUES` Klausel behandelt.

Diese Syntax erleichtert das Befüllen von (temporären) Tabellen mit Daten aus anderen Tabellen. Dies gilt umso mehr, wenn die Daten im Insert gefiltert werden sollen.

INSERT mit AUTO_INCREMENT + LAST_INSERT_ID ()

Wenn eine Tabelle einen `AUTO_INCREMENT PRIMARY KEY` , wird normalerweise keine in diese Spalte `PRIMARY KEY` . Geben Sie stattdessen alle anderen Spalten an und fragen Sie nach der neuen 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, ...);
```

Beachten Sie, dass `LAST_INSERT_ID()` an die Sitzung gebunden ist. Selbst wenn mehrere Verbindungen in dieselbe Tabelle `LAST_INSERT_ID()` , `LAST_INSERT_ID()` jede mit ihrer eigenen ID eine eigene ID.

Ihre Client-API verfügt wahrscheinlich über eine alternative Methode, um `LAST_INSERT_ID()` ohne tatsächlich `SELECT LAST_INSERT_ID()` und den Wert an den Client zurückzugeben, anstatt ihn in einer `@variable` in MySQL zu `@variable` . Dies ist normalerweise vorzuziehen.

Länger, detaillierteres Beispiel

Die "normale" Verwendung von IODKU ist das Auslösen eines "Duplikatschlüssels" basierend auf einem `UNIQUE` Schlüssel, nicht dem `AUTO_INCREMENT PRIMARY KEY` . Das Folgende zeigt, wie z. Beachten Sie, dass es *nicht* die nicht liefert `id` in der INSERT.

Setup für die folgenden Beispiele:

```
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 |
+----+--------+------+
```

Der Fall, dass IODKU ein "Update" durchführt und `LAST_INSERT_ID()` die relevante `id` `LAST_INSERT_ID()` :

```
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 |
+------------------+
```

Der Fall, in dem IODKU eine "Einfügung" durchführt und `LAST_INSERT_ID()` die neue `id` abruft:

```
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 |
+------------------+
```

Resultierender Tabelleninhalt:

```
SELECT * FROM iodku;
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  | 3333 |  -- IODKU changed this
|  3 | Dana   |  789 |  -- IODKU added this
+----+--------+------+
```

Verlorene AUTO_INCREMENT-IDs

Mehrere "Einfügen" -Funktionen können IDs "brennen". Hier ein Beispiel für die Verwendung von InnoDB (andere Engines funktionieren möglicherweise anders):

```
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'
  +----+--------+
```

Stellen Sie sich das (ungefähr) so vor: Zuerst sieht der Insert, wie viele Zeilen eingefügt werden *könnten* . Dann holen Sie sich so viele Werte aus dem auto_increment für diese Tabelle. Fügen Sie zum Schluss die Zeilen ein, verwenden Sie bei Bedarf die IDs und brennen Sie die restlichen Reste.

Der Rest kann nur wiederhergestellt werden, wenn das System heruntergefahren und neu gestartet wird. Beim Neustart wird effektiv `MAX(id)` ausgeführt. Dies kann IDs wiederverwenden, die von `DELETEs` der höchsten ID (s) verbrannt oder freigegeben wurden.

`REPLACE` kann jede `REPLACE` von `INSERT` (einschließlich `REPLACE` , `REPLACE DELETE` + `INSERT`) IDs brennen.

In InnoDB kann die globale (nicht Session!) Variable `innodb_autoinc_lock_mode` verwendet werden, um einige der `innodb_autoinc_lock_mode` zu steuern.

Beim "Normalisieren" langer Zeichenfolgen in eine `AUTO INCREMENT id` kann es leicht zum Brennen kommen. Dies kann *dazu* führen, dass die Größe des von Ihnen gewählten `INT` überläuft.

EINFÜGEN online lesen: https://riptutorial.com/de/mysql/topic/866/einfugen

Kapitel 14: Eins zu vielen

Einführung

Die Idee von eins zu vielen (1: M) bezieht sich auf das Verbinden von Zeilen miteinander, insbesondere in Fällen, in denen eine einzelne Zeile in einer Tabelle vielen Zeilen in einer anderen entspricht.

1: M ist einseitig, das heißt, wenn Sie eine 1: M-Beziehung abfragen, können Sie die Zeile 'eine' verwenden, um 'viele' Zeilen in einer anderen Tabelle auszuwählen, aber Sie können keine einzelne 'viele' Zeile verwenden Wählen Sie mehr als eine einzelne Zeile aus.

Bemerkungen

Wenn Sie mit einer 1: M-Beziehung arbeiten, müssen Sie in den meisten Fällen *Primärschlüssel* und *Fremdschlüssel* verstehen.

Ein Primärschlüssel ist eine Spalte in einer Tabelle, in der eine einzelne Zeile dieser Spalte eine einzelne Entität darstellt. Wenn Sie einen Wert in einer Primärschlüsselspalte auswählen, wird genau eine Zeile angezeigt. In den obigen Beispielen steht eine EMP_ID für einen einzelnen Mitarbeiter. Wenn Sie nach einer einzelnen EMP_ID abfragen, wird eine einzelne Zeile angezeigt, die den entsprechenden Mitarbeiter darstellt.

Ein Fremdschlüssel ist eine Spalte in einer Tabelle, die dem Primärschlüssel einer anderen Tabelle entspricht. In unserem obigen Beispiel ist die MGR_ID in der Tabelle EMPLOYEES ein Fremdschlüssel. Um zwei Tabellen zu verknüpfen, werden Sie im Allgemeinen auf der Grundlage des Primärschlüssels einer Tabelle und des Fremdschlüssels in einer anderen Tabelle zusammengefügt.

Examples

Beispiel Firmentabellen

Stellen Sie sich ein Unternehmen vor, in dem jeder Mitarbeiter, der Manager ist, einen oder mehrere Mitarbeiter verwaltet, und jeder Mitarbeiter nur einen Manager hat.

Daraus ergeben sich zwei Tabellen:

ANGESTELLTE

EMP_ID	VORNAME	NACHNAME	MGR_ID
E01	Johnny	Appleseed	M02
E02	Erin	Macklemore	M01

EMP_ID	VORNAME	NACHNAME	MGR_ID
E03	Colby	Papierkram	M03
E04	Ron	Sonswan	M01

MANAGER

MGR_ID	VORNAME	NACHNAME
M01	Laut	McQueen
M02	Herrisch	Hose
M03	Fass	Jones

Erhalten Sie die Mitarbeiter, die von einem einzelnen Manager verwaltet werden

```
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' ;
```

Ergebnisse in:

EMP_ID	VORNAME	NACHNAME
E02	Erin	Macklemore
E04	Ron	Sonswan

Letztendlich wird für jeden Manager, nach dem wir fragen, ein oder mehrere Mitarbeiter zurückgeschickt.

Holen Sie sich den Manager für einen einzelnen Mitarbeiter

Beachten Sie die obigen Beispieltabellen, wenn Sie dieses Beispiel betrachten.

```
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	VORNAME	NACHNAME
M03	Fass	Jones

Da dies das umgekehrte Beispiel ist, wissen wir, dass wir für jeden Mitarbeiter, den wir abfragen, immer nur einen entsprechenden Manager sehen werden.

Eins zu vielen online lesen: https://riptutorial.com/de/mysql/topic/9600/eins-zu-vielen

Kapitel 15: ENUM

Examples

Warum ENUM?

ENUM bietet eine Möglichkeit, ein Attribut für eine Zeile bereitzustellen. Attribute mit einer kleinen Anzahl nicht numerischer Optionen funktionieren am besten. Beispiele:

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

Die Werte sind Strings:

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

TINYINT als Alternative

Sagen wir, wir haben

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

Eine Alternative ist

```
type TINYINT UNSIGNED
```

Plus

```
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
```

Das ist sehr ähnlich wie ein Many-to-Many-Tisch.

Vergleich und ob besser oder schlechter als ENUM:

- (schlechter) INSERT: müssen den `type` nachschlagen
- (schlechter) SELECT: Sie müssen sich mit JOIN verbinden, um die Zeichenfolge zu erhalten (ENUM gibt Ihnen die Zeichenfolge ohne Aufwand).
- (besser) Hinzufügen neuer Typen: Einfach in diese Tabelle einfügen. Mit ENUM müssen Sie eine ALTER TABLE ausführen.
- (gleich) Beide Verfahren (für bis zu 255 Werte) benötigen nur 1 Byte.

- (gemischt) Es gibt auch ein Problem der Datenintegrität: `TINYINT` ungültige Werte zu. `ENUM` setzt sie jedoch auf einen speziellen Leerzeichenfolgenwert (sofern der strikte SQL-Modus nicht aktiviert ist; in diesem Fall werden sie abgelehnt). Bessere Datenintegrität kann mit `TINYINT` werden, indem aus einem Fremdschlüssel eine Lookup-Tabelle erstellt wird, die bei entsprechenden Abfragen / `TINYINT` mit dem `TINYINT` auf die andere Tabelle verbunden ist. (`FOREIGN KEYs` sind nicht frei.)

VARCHAR als Alternative

Sagen wir, wir haben

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

Eine Alternative ist

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

Dies ist insofern recht offen, als neue Typen trivial hinzugefügt werden.

Vergleich und ob besser oder schlechter als ENUM:

- (gleich) INSERT: einfach die Zeichenfolge angeben
- (schlechter?) Bei INSERT wird ein Tippfehler unbemerkt bleiben
- (gleich) SELECT: Die aktuelle Zeichenfolge wird zurückgegeben
- (schlechter) Es wird viel mehr Speicherplatz verbraucht

Eine neue Option hinzufügen

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

Anmerkungen

- Wie in allen Fällen von MODIFY COLUMN müssen Sie `NOT NULL` und alle anderen bereits vorhandenen Qualifikationsmerkmale angeben.
- *Wenn* Sie am *Ende* der Liste hinzufügen *und* die Liste `ALTER` als 256 Elemente enthält, wird der `ALTER` durch einfaches Ändern des Schemas ausgeführt. Das heißt, es wird keine lange Tabellenkopie geben. (Alte Versionen von MySQL hatten diese Optimierung nicht.)

NULL vs. NOT NULL

Beispiele dafür, was passiert, wenn NULL und 'bad-value' in null- und nicht nullfähige Spalten gespeichert werden Zeigt auch die Verwendung des Gusses über `+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)
```

Was steht in der Tabelle nach diesen Einsätzen? Dies verwendet "+0", um in numerische Werte umzuwandeln, was gespeichert ist.

```
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)
```

ENUM online lesen: https://riptutorial.com/de/mysql/topic/4425/enum

Kapitel 16: Extrahieren Sie Werte aus dem JSON-Typ

Einführung

MySQL 5.7.8+ unterstützt den nativen JSON-Typ. Sie haben verschiedene Möglichkeiten, Json-Objekte zu erstellen, Sie können jedoch auch auf verschiedene Arten auf Mitglieder zugreifen und sie lesen.

Die Hauptfunktion ist `JSON_EXTRACT` , daher sind `->` und `->>` Operatoren freundlicher.

Syntax

- JSON_EXTRACT (json_doc, Pfad [, ...])
- JSON_EXTRACT (json_doc, Pfad)
- JSON_EXTRACT (json_doc, Pfad1, Pfad2)

Parameter

Parameter	Beschreibung
json_doc	gültiges JSON-Dokument
Pfad	Mitgliederpfad

Bemerkungen

Erwähnt in MySQL 5.7 Referenzhandbuch

- Mehrere übereinstimmende Werte nach Pfadargument (en)

  Wenn es möglich ist, dass diese Argumente mehrere Werte zurückgeben, werden die übereinstimmenden Werte als Array automatisch in der Reihenfolge der Pfade, in denen sie erzeugt wurden, als Array deklariert. Andernfalls ist der Rückgabewert der einzelne übereinstimmende Wert.

- `NULL` Ergebnis wenn:
  - Jeder Kamerad ist NULL
  - Pfad nicht übereinstimmend

  Gibt NULL zurück, wenn ein Argument NULL ist oder keine Pfade einen Wert im Dokument suchen.

Examples

Lesen Sie den JSON-Array-Wert

@Myjson-Variable als JSON-Typ erstellen (mehr lesen):

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

`SELECT` einige Mitglieder aus!

```
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"]
```

JSON-Extraktionsoperatoren

`path` durch `->` oder `->>` Operatoren extrahieren, während `->>` UNQUOTED-Wert ist:

```
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"]
 --^^^ ^^^
```

`col->>path` ist also gleich `JSON_UNQUOTE(JSON_EXTRACT(col,path))` :

> Wie bei -> wird der Operator - >> in der Ausgabe von EXPLAIN immer erweitert, wie das folgende Beispiel zeigt:

```
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)
```

Lesen Sie über den Inline-Pfadauszug (+)

Extrahieren Sie Werte aus dem JSON-Typ online lesen:
https://riptutorial.com/de/mysql/topic/9042/extrahieren-sie-werte-aus-dem-json-typ

Kapitel 17: Fehler 1055: ONLY_FULL_GROUP_BY: etwas ist nicht in der GROUP BY-Klausel enthalten ...

Einführung

Vor kurzem haben neue Versionen von MySQL-Servern begonnen, 1055-Fehler für Abfragen zu generieren, die früher funktionierten. In diesem Thema werden diese Fehler erläutert. Das MySQL-Team hat daran gearbeitet, die nicht standardmäßige Erweiterung von `GROUP BY` , oder es zumindest zu erschweren, dass Entwickler von Abfrageschreiben davon gebrannt werden.

Bemerkungen

MySQL enthält seit langem eine notorische, nicht standardmäßige Erweiterung für `GROUP BY` , die ein seltsames Verhalten im Namen der Effizienz ermöglicht. Diese Erweiterung hat es unzähligen Entwicklern auf der ganzen Welt ermöglicht, `GROUP BY` im Produktionscode zu verwenden, ohne vollständig zu verstehen, was sie tun.

Es ist insbesondere nicht `SELECT *` , `SELECT *` in einer `GROUP BY` Abfrage zu verwenden, da eine Standard- `GROUP BY` Klausel das Auflisten der Spalten erfordert. Viele Entwickler haben das leider getan.

Lesen Sie dies. https://dev.mysql.com/doc/refman/5.7/de/group-by-handling.html

Das MySQL-Team hat versucht, dieses Missverständnis zu beheben, ohne den Produktionscode zu beeinträchtigen. In `sql_mode` ein `sql_mode` Flag mit dem Namen `ONLY_FULL_GROUP_BY` , um das Standardverhalten zu erzwingen. In einer kürzlich veröffentlichten Version wurde das Flag standardmäßig aktiviert. Wenn Sie Ihr lokales MySQL auf 5.7.14 aktualisiert haben, wurde das Flag aktiviert und Ihr Produktionscode, abhängig von der alten Erweiterung, funktioniert nicht mehr.

Wenn Sie vor kurzem begonnen haben, 1055 Fehler zu erhalten, welche Entscheidungen treffen Sie?

1. Korrigieren Sie die fehlerhaften SQL-Abfragen, oder fordern Sie die Autoren dazu auf.
2. Führen Sie mit der von Ihnen verwendeten Anwendungssoftware eine Version von MySQL-kompatibel aus
3. Ändern Sie den `sql_mode` Ihres Servers, um den neu eingestellten `ONLY_FULL_GROUP_BY` Modus zu `ONLY_FULL_GROUP_BY` .

Sie können den Modus durch einen `SET` Befehl ändern.

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

Sie sollten den Trick ausführen, wenn Sie es tun, nachdem Ihre Anwendung eine Verbindung zu MySQL hergestellt hat.

Sie können die init-Datei auch in Ihrer MySQL-Installation finden, die `sql_mode=` und in `ONLY_FULL_GROUP_BY` auslassen und den Server neu starten.

Examples

Verwendung und Missbrauch von 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
```

zeigt die Zeilen in einer Tabelle namens `item` und die Anzahl der zugehörigen Zeilen in einer Tabelle mit dem Namen `uses` . Das funktioniert gut, aber leider nicht Standard-SQL-92.

Warum nicht? weil die `SELECT` Klausel (und die `ORDER BY` Klausel) in `GROUP BY` Abfragen Spalten enthalten müssen, die sind

1. in der `GROUP BY` Klausel erwähnt, oder
2. Aggregatfunktionen wie `COUNT()` , `MIN()` und dergleichen.

Die `SELECT` Klausel dieses Beispiels erwähnt `item.name` , eine Spalte, die keines dieser Kriterien erfüllt. MySQL 5.6 und frühere `ONLY_FULL_GROUP_BY` lehnen diese Abfrage ab, wenn der SQL-Modus `ONLY_FULL_GROUP_BY` enthält.

Diese Beispielabfrage kann so gemacht werden, dass sie dem SQL-92-Standard entspricht, indem die `GROUP BY` Klausel wie `GROUP BY` wird.

```
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
```

Der spätere SQL-99-Standard ermöglicht einer `SELECT` Anweisung, nicht aggregierte Spalten aus dem Gruppenschlüssel auszulassen, wenn das DBMS eine funktionale Abhängigkeit zwischen diesen und den Gruppenschlüsselspalten nachweisen kann. Da `item.name` funktional von `item.item_id` , ist das ursprüngliche Beispiel gültiges SQL-99. MySQL hat in Version 5.7 einen funktionellen Abhängigkeitsbeweiser erhalten . Das ursprüngliche Beispiel funktioniert unter `ONLY_FULL_GROUP_BY` .

Missbrauch von GROUP BY zur Rückgabe unvorhersehbarer Ergebnisse:

Murphy's Law

```
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
```

zeigt die Zeilen in einer Tabelle namens item und die Anzahl der zugehörigen Zeilen in einer Tabelle mit dem Namen use an. Es zeigt auch den Wert einer Spalte mit der Bezeichnung " `uses.category` .

Diese Abfrage funktioniert in MySQL (bevor das Flag `ONLY_FULL_GROUP_BY`). Es verwendet die nicht standardmäßige Erweiterung von MySQL für `GROUP BY` .

Aber die Abfrage hat ein Problem: wenn mehrere Zeilen in der `uses` die übereinstimmen `ON` Zustand in der `JOIN` - Klausel, MySQL gibt die `category` Spalte von nur einer dieser Zeilen. Welche reihe Der Verfasser der Abfrage und der Benutzer der Anwendung erfahren dies nicht im Voraus. Formal gesehen ist es *unvorhersehbar* : MySQL kann jeden gewünschten Wert zurückgeben.

Unvorhersehbar ist wie *zufällig,* mit einem signifikanten Unterschied. Man könnte erwarten, dass sich eine *zufällige* Wahl von Zeit zu Zeit ändert. Wenn eine Auswahl zufällig war, können Sie sie daher beim Debuggen oder Testen finden. Das *unvorhersehbare* Ergebnis ist schlimmer: MySQL gibt jedes Mal dasselbe Ergebnis zurück, wenn Sie die Abfrage verwenden, *bis dies nicht der Fall ist.* Manchmal führt eine neue Version des MySQL-Servers zu einem anderen Ergebnis. Manchmal ist es eine wachsende Tabelle, die das Problem verursacht. Was schief gehen kann, wird schief gehen und wenn Sie es nicht erwarten. Das nennt man Murphys Gesetz .

Das MySQL-Team hat daran gearbeitet, dass Entwickler diesen Fehler noch schwieriger machen können. Neuere Versionen von MySQL in der 5.7-Sequenz haben ein `sql_mode` Flag namens `ONLY_FULL_GROUP_BY` . Wenn dieses Flag gesetzt ist, gibt der MySQL-Server den Fehler 1055 zurück und lehnt die Ausführung dieser Art von Abfrage ab.

Missbrauch von GROUP BY mit SELECT * und deren Behebung.

Manchmal sieht eine Abfrage mit einem `*` in der `SELECT` Klausel so aus.

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

Eine solche Abfrage muss `ONLY_FULL_GROUP_BY` werden, um den Standard `ONLY_FULL_GROUP_BY` erfüllen.

Dazu benötigen wir eine Unterabfrage, die `GROUP BY` korrekt verwendet, um den Wert `number_of_uses` für jede `item_id` . Diese Unterabfrage ist kurz und süß, weil es nur auf suchen braucht `uses` Tabelle.

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

Dann können wir diese Unterabfrage mit der Join - `item` - Tabelle.

```
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
```

Dadurch kann die `GROUP BY` Klausel einfach und korrekt sein, und wir können auch den `*` -Spezifizierer verwenden.

Hinweis: Trotzdem vermeiden kluge Entwickler die Verwendung des `*` -Spezifizierers auf jeden Fall. Es ist normalerweise besser, die gewünschten Spalten in einer Abfrage aufzulisten.

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
```

Zeigt die Zeilen in einer Tabelle mit dem Namen `item` , die Anzahl der zugehörigen Zeilen und einen der Werte in der zugehörigen Tabelle mit dem Namen `uses` .

Sie können sich diese `ANY_VALUE()` Funktion als eine Art Aggregatfunktion `ANY_VALUE()` . Anstatt einen Zähler, eine Summe oder ein Maximum zurückzugeben, weist es den MySQL-Server an, beliebig einen Wert aus der betreffenden Gruppe auszuwählen. Es ist eine Möglichkeit, den Fehler 1055 zu umgehen.

Seien Sie vorsichtig, wenn Sie `ANY_VALUE()` in Abfragen in Produktionsanwendungen verwenden.

Es sollte wirklich `SURPRISE_ME()` heißen. Es gibt den Wert einer Zeile in der GROUP BY-Gruppe zurück. Welche Zeile zurückgegeben wird, ist unbestimmt. Das heißt, es liegt vollständig am MySQL-Server. Formal gibt es einen unvorhersehbaren Wert zurück.

Der Server wählt keinen zufälligen Wert, der ist schlimmer. Bei jeder Ausführung der Abfrage wird derselbe Wert zurückgegeben, bis dies nicht der Fall ist. Sie kann sich ändern oder nicht, wenn eine Tabelle wächst oder schrumpft oder wenn der Server mehr oder weniger RAM hat oder wenn sich die Serverversion ändert oder wenn der Mars sich im Hintergrund befindet (was immer dies bedeutet) oder ohne Grund.

Du wurdest gewarnt.

Fehler 1055: ONLY_FULL_GROUP_BY: etwas ist nicht in der GROUP BY-Klausel enthalten ...

online lesen: https://riptutorial.com/de/mysql/topic/8245/fehler-1055--only-full-group-by--etwas-ist-nicht-in-der-group-by-klausel-enthalten----

Kapitel 18: Fehlercodes

Examples

Fehlercode 1064: Syntaxfehler

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

Nachricht zurücksenden:

> Fehlercode: 1064. Sie haben einen Fehler in Ihrer SQL-Syntax. Überprüfen Sie das Handbuch, das Ihrer MySQL-Server-Version entspricht, auf die richtige Syntax, die in der Nähe von 'von Person' in Zeile 2 verwendet werden soll.

Wenn Sie eine Meldung "1064 error" von MySQL erhalten, kann die Abfrage nicht ohne Syntaxfehler analysiert werden. Mit anderen Worten kann die Abfrage keinen Sinn ergeben.

Das Zitat in der Fehlermeldung beginnt mit dem ersten Zeichen der Abfrage, das MySQL nicht ermitteln kann, wie es zu analysieren ist. In diesem Beispiel kann MySQL im Kontext `from Person` keinen Sinn ergeben. In diesem Fall gibt es unmittelbar vor `from Person` ein zusätzliches Komma. Das Komma weist MySQL an, eine andere Spaltenbeschreibung in der `SELECT` Klausel zu erwarten

Ein Syntaxfehler sagt immer `... near '...'`. Die Sache am Anfang der Anführungszeichen ist sehr nahe dort, wo der Fehler liegt. Um einen Fehler zu finden, sehen Sie sich das erste Token in den Anführungszeichen und das letzte Token vor den Anführungszeichen an.

Manchmal kommt man in die `... near ''` ; das heißt, nichts in den Zitaten. Das erste Zeichen, das MySQL nicht herausfinden kann, steht am Ende oder am Anfang der Anweisung. Dies deutet darauf hin, dass die Abfrage unausgeglichene Anführungszeichen (`'` oder `"`) oder unausgeglichene Klammern enthält oder dass Sie die Anweisung zuvor nicht korrekt beendet haben.

Bei einer gespeicherten Routine haben Sie möglicherweise vergessen, `DELIMITER` ordnungsgemäß zu verwenden.

Wenn Sie also den Fehler 1064 erhalten, schauen Sie sich den Text der Abfrage an und suchen Sie den in der Fehlermeldung genannten Punkt. Untersuchen Sie den Text der Abfrage visuell an diesem Punkt.

Wenn Sie jemanden um Hilfe bei der Behebung von Fehler 1064 bitten, geben Sie am besten sowohl den Text der gesamten Abfrage als auch den Text der Fehlermeldung an.

Fehlercode 1175: Sicheres Update

Dieser Fehler tritt auf, wenn Sie versuchen, Datensätze zu aktualisieren oder zu löschen, ohne die `WHERE` Klausel zu verwenden, die die `KEY` Spalte verwendet.

Um das Löschen oder Aktualisieren trotzdem auszuführen, geben Sie Folgendes ein:

```
SET SQL_SAFE_UPDATES = 0;
```

Um den abgesicherten Modus wieder zu aktivieren, geben Sie Folgendes ein:

```
SET SQL_SAFE_UPDATES = 1;
```

Fehlercode 1215: Fremdschlüsseleinschränkung kann nicht hinzugefügt werden

Dieser Fehler tritt auf, wenn Tabellen nicht ausreichend strukturiert sind, um die vom Entwickler geforderte Überprüfung der fremden Schlüssel (Foreign Key, `FK`) zu prüfen.

```
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;
```

Hinweis 1: Ein derartiger SCHLÜSSEL wird automatisch erstellt, wenn dies aufgrund der FK-Definition in der folgenden Zeile erforderlich ist. Der Entwickler kann es überspringen und der KEY (aka index) wird bei Bedarf hinzugefügt. Ein Beispiel dafür, wie es vom Entwickler übersprungen wird, ist unten in `someOther` .

So weit so gut, bis zum unten genannten Aufruf.

```
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;
```

> Fehlercode: 1215. Fremdschlüsseleinschränkung kann nicht hinzugefügt werden

In diesem Fall schlägt der Vorgang fehl, da in der *referenzierten* Tabelle `getTogethers` kein Index `getTogethers` , um die schnelle Suche nach einem `eventDT` `getTogethers` zu behandeln. In der nächsten Aussage zu lösen.

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

Die Tabelle `getTogethers` wurde geändert, und nun wird die Erstellung von `someOther` erfolgreich sein.

Auf der MySQL-Handbuchseite mit FOREIGN KEY-Einschränkungen :

> MySQL erfordert Indizes für Fremdschlüssel und referenzierte Schlüssel, damit Fremdschlüsselüberprüfungen schnell durchgeführt werden können und keine Tabellensuche erforderlich ist. In der referenzierenden Tabelle muss ein Index vorhanden sein, in dem die Fremdschlüsselspalten als erste Spalten in derselben Reihenfolge aufgeführt sind. Ein solcher Index wird automatisch für die referenzierende Tabelle erstellt, wenn er nicht vorhanden ist.
>
> Entsprechende Spalten im Fremdschlüssel und der referenzierte Schlüssel müssen ähnliche Datentypen haben. Die Größe und das Vorzeichen von Integer-Typen müssen gleich sein. Die Länge der Zeichenfolgentypen muss nicht gleich sein. Für nicht-binäre (Zeichen-) Zeichenfolgenspalten müssen der Zeichensatz und die Sortierung identisch sein.
>
> InnoDB erlaubt einem Fremdschlüssel, auf Indexspalten oder Spaltengruppen zu verweisen. In der referenzierten Tabelle muss jedoch ein Index vorhanden sein, in dem die referenzierten Spalten als erste Spalten in derselben Reihenfolge aufgeführt sind.

Beachten Sie, dass der letzte Punkt oben über die erste (ganz linke) Spalte und das Fehlen einer Primärschlüssel-Anforderung (obwohl dringend empfohlen).

Bei erfolgreicher Erstellung einer *referenzierten* (untergeordneten) Tabelle sind alle automatisch für Sie erstellten Schlüssel mit einem Befehl wie dem folgenden sichtbar:

```
SHOW CREATE TABLE someOther;
```

Andere häufige Fälle, in denen dieser Fehler auftritt, sind, wie oben in den Dokumenten erwähnt, jedoch hervorzuheben:

- Scheinbar triviale Unterschiede in `INT` die signiert sind und auf `INT UNSIGNED` .
- Entwickler, die Probleme mit dem Verständnis von mehrspaltigen (zusammengesetzten) Schlüsseln und den ersten (am weitesten links liegenden) Bestellanforderungen haben.

1045 Zugriff verweigert

Siehe die Diskussionen unter "GRANT" und "Wiederherstellen des Root-Passworts".

1236 "unmögliche Position" in der Replikation

Normalerweise bedeutet dies, dass der Master abgestürzt ist und dass `sync_binlog` war. Die Lösung ist, `CHANGE MASTER to POS=0` der nächsten Binlog-Datei (siehe Master) auf dem Slave zu ändern.

Die Ursache: Der Master sendet Replikationselemente an den Slave, bevor er in sein Binlog

gespült wird (wenn `sync_binlog=OFF`). Wenn der Master vor dem Flush abstürzt, ist der Slave bereits logisch am Ende der Datei im Binlog vorbeigefahren. Wenn der Master erneut startet, startet er ein neues Binlog. Daher ist das Wechseln an den Anfang des Binlog die beste verfügbare Lösung.

Eine längerfristige Lösung ist `sync_binlog=ON` , wenn Sie sich die zusätzlichen E / A leisten können, die dadurch verursacht werden.

(Wenn Sie mit GTID laufen, ...?)

2002, 2003 Verbindung kann nicht hergestellt werden

Prüfen Sie, ob ein Firewall-Problem den Port 3306 blockiert.

Einige mögliche Diagnosen und / oder Lösungen

- Läuft der Server tatsächlich?
- "service firewalld stop" und "systemctl disable firewalld"
- Telnet-Master 3306
- Überprüfen Sie die `bind-address`
- Überprüfen Sie das `skip-name-resolve`
- Überprüfen Sie die Steckdose.

1067, 1292, 1366, 1411 - Ungültiger Wert für Anzahl, Datum, Standard usw.

1067 Dies hängt wahrscheinlich mit den `TIMESTAMP` , die sich im Laufe der Zeit geändert haben. Siehe `TIMESTAMP defaults` auf der Seite Datum & `TIMESTAMP defaults` . (was es noch nicht gibt)

1292/1366 DOUBLE / Integer Auf Buchstaben oder andere Syntaxfehler **prüfen** . Stellen Sie sicher, dass die Spalten ausgerichtet sind. Vielleicht denken Sie, Sie stecken in einem `VARCHAR` aber es ist an einer numerischen Spalte ausgerichtet.

1292 DATETIME In der Vergangenheit oder in der Zukunft zu weit **geprüft** . Überprüfen Sie zwischen 2 Uhr morgens und 3 Uhr morgens an einem Morgen, wenn sich die Sommerzeit geändert hat. Überprüfen Sie, `+00` Syntax fehlerhaft ist, z. B. `+00` Zeitzonen.

1292 VARIABLE Überprüfen Sie die zulässigen Werte für die `VARIABLE` Sie `VARIABLE SET` .

1292 LOAD DATA Schauen Sie sich die Zeile an, die 'bad' ist. Überprüfen Sie die Escape-Symbole usw. Schauen Sie sich die Datentypen an.

1411 STR_TO_DATE Datum falsch formatiert?

126, 127, 134, 144, 145

Wenn Sie versuchen, auf die Datensätze aus der MySQL-Datenbank zuzugreifen, erhalten Sie möglicherweise diese Fehlermeldungen. Diese Fehlermeldungen sind auf eine Beschädigung der MySQL-Datenbank zurückzuführen. Es folgen die Typen

```
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
```

MySQL-Fehler, Virenbefall, Serverabsturz, fehlerhaftes Herunterfahren, beschädigte Tabelle sind der Grund für diese Beschädigung. Wenn es beschädigt wird, ist es nicht mehr zugänglich und Sie können nicht mehr darauf zugreifen. Um Zugang zu erhalten, ist der beste Weg, um Daten aus einer aktualisierten Sicherung abzurufen. Wenn Sie jedoch keine aktualisierte oder gültige Sicherung haben, können Sie sich für MySQL Repair entscheiden.

Wenn der Tabellen-Engine-Typ `MyISAM` , wenden Sie `CHECK TABLE` und dann `REPAIR TABLE` an.

Denken Sie dann ernsthaft über die Umstellung auf InnoDB nach, damit dieser Fehler nicht mehr auftritt.

Syntax

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

139

Fehler 139 kann bedeuten, dass die Anzahl und Größe der Felder in der Tabellendefinition einen gewissen Grenzwert überschreitet. Problemumgehungen:

- Überdenken Sie das Schema erneut
- Normalisieren Sie einige Felder
- Partitionieren Sie die Tabelle vertikal

1366

Dies bedeutet normalerweise, dass die Behandlung von Zeichensätzen zwischen Client und Server nicht konsistent war. Siehe ... für weitere Unterstützung.

126, 1054, 1146, 1062, 24

(Pause) Mit der Einbeziehung dieser 4 Fehlernummern denke ich, dass diese Seite ungefähr 50% der typischen Fehler abgedeckt hat, die Benutzer erhalten.

(Ja, dieses 'Beispiel' muss überarbeitet werden.)

24 Datei kann nicht geöffnet werden (zu viele geöffnete Dateien)

`open_files_limit` kommt von einer Betriebssystemeinstellung. `table_open_cache` muss kleiner sein.

Diese können diesen Fehler verursachen:

- Fehler beim `DEALLOCATE PREPARE` in einer gespeicherten Prozedur.

- PARTITIONed-Tabelle (n) mit einer großen Anzahl von Partitionen und innodb_file_per_table = ON. Es sollte empfohlen werden, nicht mehr als 50 Partitionen in einer Tabelle zu haben (aus verschiedenen Gründen). (Wenn "native Partitionen" verfügbar werden, kann sich dieser Hinweis ändern.)

Die offensichtliche Problemumgehung besteht darin, das Betriebssystem-Limit zu erhöhen: Um mehr Dateien zuzulassen, ändern Sie `ulimit` oder `/etc/security/limits.conf` oder in `sysctl.conf` (kern.maxfiles & kern.maxfilesperproc) oder etwas anderes (vom Betriebssystem abhängig). Erhöhen `open_files_limit` dann `open_files_limit` und `table_open_cache` .

Seit 5.6.8 hat `open_files_limit` basierend auf `max_connections` automatische `max_connections` , es ist jedoch in Ordnung, die Standardeinstellung zu ändern.

1062 - Doppelter Eintrag

Dieser Fehler tritt hauptsächlich aus den folgenden zwei Gründen auf

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

   Die Primärschlüsselspalte ist eindeutig und akzeptiert den doppelten Eintrag nicht. Wenn Sie also versuchen, eine neue Zeile einzufügen, die bereits in Ihrer Tabelle vorhanden ist, wird dieser Fehler ausgegeben.

   Um dieses `AUTO_INCREMENT` zu lösen, legen Sie die Primärschlüsselspalte als `AUTO_INCREMENT` . Wenn Sie versuchen, eine neue Zeile einzufügen, ignorieren Sie die Primärschlüsselspalte oder fügen Sie einen `NULL` Wert in den Primärschlüssel ein.

```
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. *Eindeutiges Datenfeld* - `Error Code: 1062. Duplicate entry 'A' for key 'code'`

   Sie können eine Spalte als eindeutig zuweisen. Wenn Sie versuchen, eine neue Zeile mit bereits vorhandenem Wert für diese Spalte einzufügen, wird dieser Fehler ausgegeben.

   Um diesen Fehler zu `INSERT IGNORE` , verwenden Sie `INSERT IGNORE` anstelle des normalen `INSERT` . Wenn die neue Zeile, die Sie einzufügen versuchen, einen vorhandenen Datensatz nicht dupliziert, fügt MySQL diesen wie gewohnt ein. Wenn der Datensatz ein Duplikat ist, wird er vom `IGNORE` Schlüsselwort `IGNORE` , ohne dass ein Fehler generiert wird.

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

Fehlercodes online lesen: https://riptutorial.com/de/mysql/topic/895/fehlercodes

Kapitel 19: Gespeicherte Routinen (Prozeduren und Funktionen)

Parameter

Parameter	Einzelheiten
KEHRT ZURÜCK	Gibt den Datentyp an, der von einer Funktion zurückgegeben werden kann.
RÜCKKEHR	Die tatsächliche Variable oder der tatsächliche Wert nach der `RETURN` Syntax wird an den Ort zurückgegeben, von dem aus die Funktion aufgerufen wurde.

Bemerkungen

Eine gespeicherte Routine ist entweder eine Prozedur oder eine Funktion.

Eine Prozedur wird mit einer CALL-Anweisung aufgerufen und kann Werte nur mit Ausgabevariablen zurückgeben.

Eine Funktion kann wie jede andere Funktion aus einer Anweisung heraus aufgerufen werden und kann einen Skalarwert zurückgeben.

Examples

Erstellen Sie eine Funktion

Die folgende (triviale) Beispielfunktion gibt einfach den konstanten `INT` Wert `12` .

```
DELIMITER ||
CREATE FUNCTION functionname()
RETURNS INT
BEGIN
    RETURN 12;
END;
||
DELIMITER ;
```

In der ersten Zeile wird festgelegt, in was das Trennzeichen (`DELIMITER ||`) geändert werden soll. Dies muss festgelegt werden, bevor eine Funktion erstellt wird. Andernfalls, wenn die Standardeinstellung `DELIMITER ||` wird `;` dann der erste `;` Dies wird im Funktionshauptteil als Ende der `CREATE` Anweisung betrachtet, was normalerweise nicht erwünscht ist.

Nachdem die `CREATE FUNCTION` ausgeführt wurde, sollten Sie das Trennzeichen auf den

Standardwert von setzen `;` wie im obigen Beispiel nach dem Funktionscode (`DELIMITER ;`).

Ausführung dieser Funktion ist wie folgt:

```
SELECT functionname();
+----------------+
| functionname() |
+----------------+
|             12 |
+----------------+
```

Ein etwas komplexeres (aber immer noch triviales) Beispiel nimmt einen Parameter und fügt ihm eine Konstante hinzu:

```
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 |
+-----------+
```

Beachten Sie die Verwendung eines anderen Arguments als die `DELIMITER` Direktive. Sie können tatsächlich eine beliebige Zeichenfolge verwenden, die nicht im Hauptteil der `CREATE` Anweisung enthalten ist. In der Regel wird jedoch ein doppeltes, nicht alphanumerisches Zeichen wie `\\` , `||` oder `$$` .

Es ist empfehlenswert, den Parameter immer vor und nach einer Funktion, Prozedur oder Triggererstellung oder -aktualisierung zu ändern, da für einige GUI keine Änderungen des Trennzeichens erforderlich sind, während für Abfragen über die Befehlszeile immer das Setzen des Trennzeichens erforderlich ist.

Prozedur mit einer konstruierten Vorbereitung erstellen

```
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
```

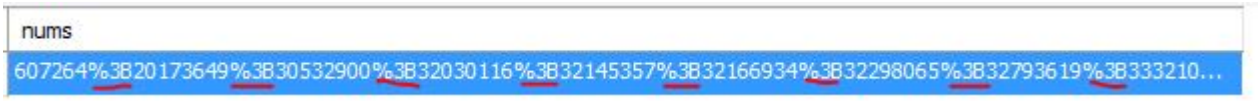
```
1160 or whatever it is

    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 ;
```

Die Erstellung der gespeicherten Prozedur zeigt das Umschließen mit einem DELIMITER, der in vielen Client-Tools erforderlich ist.

Aufrufbeispiel:

```
call displayNext100WithName(1,"questions_mysql");
```

Beispielausgabe mit dem Trennzeichen `%3B` (Semikolon):



Gespeicherte Prozedur mit den Parametern 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 ;
```

Ruft die gespeicherte Prozedur auf (CALL):

```
SET @z = 30;
call sp_nested_loop(10, 20, @x, @y, @z);
SELECT @x, @y, @z;
```

Ergebnis:

```
+------+------+------+
|  @x  |  @y  |  @z  |
+------+------+------+
|  10  |  200 |  240 |
+------+------+------+
```

Ein `IN` Parameter übergibt einen Wert an eine Prozedur. Die Prozedur kann den Wert ändern, die Änderung ist jedoch für den Aufrufer nicht sichtbar, wenn die Prozedur zurückkehrt.

Ein `OUT` Parameter übergibt einen Wert aus der Prozedur an den Aufrufer. Sein Anfangswert ist NULL innerhalb der Prozedur und sein Wert ist für den Aufrufer sichtbar, wenn die Prozedur zurückkehrt.

Ein `INOUT` Parameter wird vom Aufrufer initialisiert und kann von der Prozedur geändert werden. Jede durch die Prozedur vorgenommene Änderung ist für den Aufrufer sichtbar, wenn die Prozedur zurückkehrt.

Ref: http://dev.mysql.com/doc/refman/5.7/de/create-procedure.html

Cursor

Mithilfe von Cursorn können Sie die Abfrageergebnisse einer nach dem anderen durchlaufen. `DECLARE` Befehl `DECLARE` wird verwendet, um den Cursor zu `DECLARE` und einer bestimmten SQL-Abfrage `DECLARE` :

```
DECLARE student CURSOR FOR SELECT name FROM studend;
```

Angenommen, wir verkaufen Produkte einiger Arten. Wir möchten zählen, wie viele Produkte von jedem Typ vorhanden sind.

Unsere Daten:

```
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');
```

Wir können das Ziel mithilfe der gespeicherten Prozedur mit dem Cursor erreichen:

```
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 ;
```

Wann können Sie die Prozedur aufrufen mit:

```
CALL product_count();
```

Ergebnis wäre in der Tabelle `product_type_count` :

```
type   | count
---------------
dress  |   2
food   |   3
```

Während dies ein gutes Beispiel für einen `CURSOR` , beachten Sie, wie der gesamte Körper des Verfahrens durch nur ersetzt werden kann

```
INSERT INTO product_type_count
        (type, count)
    SELECT type, COUNT(*)
        FROM product
        GROUP BY type;
```

Das wird viel schneller laufen.

Mehrere ResultSets

Im Gegensatz zu einer `SELECT` Anweisung gibt eine `Stored Procedure` mehrere Ergebnismengen zurück. Dies erfordert, dass ein anderer Code verwendet wird, um die Ergebnisse eines `CALL` in Perl, PHP usw. zu erfassen.

(Benötigen Sie hier oder anderswo speziellen Code!)

Erstellen Sie eine Funktion

```
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 ;
```

Der RETURNS-Datentyp ist ein beliebiger MySQL-Datentyp.

Gespeicherte Routinen (Prozeduren und Funktionen) online lesen:
https://riptutorial.com/de/mysql/topic/1351/gespeicherte-routinen--prozeduren-und-funktionen-

Kapitel 20: Gruppiere nach

Syntax

1. SELECT Ausdruck1, Ausdruck2, ... Ausdruck_n,
2. aggregate_function (Ausdruck)
3. FROM Tische
4. [WO Bedingungen]
5. GROUP BY Ausdruck1, Ausdruck2, ... Ausdruck_n;

Parameter

Parameter	EINZELHEITEN
expression1, expression2, ... expression_n	Die Ausdrücke, die nicht in einer Aggregatfunktion eingeschlossen sind und in der GROUP BY-Klausel enthalten sein müssen.
aggregate_function	Eine Funktion wie SUMME, COUNT, MIN, MAX oder AVG.
Tische	Die Tabellen, von denen Sie Datensätze abrufen möchten. In der FROM-Klausel muss mindestens eine Tabelle aufgeführt sein.
WHERE Bedingungen	Wahlweise. Die Bedingungen, die erfüllt sein müssen, damit die Datensätze ausgewählt werden.

Bemerkungen

Die MySQL GROUP BY-Klausel wird in einer SELECT-Anweisung verwendet, um Daten über mehrere Datensätze hinweg zu sammeln und die Ergebnisse nach einer oder mehreren Spalten zu gruppieren.

Sein Verhalten wird zum Teil durch den Wert der Variablen `ONLY_FULL_GROUP_BY` bestimmt . Wenn dies aktiviert ist, geben `SELECT` Anweisungen, die nach einer beliebigen Spalte gruppieren, die nicht in der Ausgabe enthalten ist, einen Fehler zurück. (Dies ist der Standard ab 5.7.5 .) Sowohl das Setzen als auch das Nicht-Setzen dieser Variablen kann für naive Benutzer oder Benutzer, die an andere DBMS gewöhnt sind, Probleme verursachen.

Examples

GROUP BY US-Funktion verwenden

```
SELECT product, SUM(quantity) AS "Total quantity"
FROM order_details
GROUP BY product;
```

Gruppieren mit MIN-Funktion

Angenommen, eine Tabelle mit Angestellten, in der jede Zeile ein Angestellter ist, der einen `name` , eine `department` und ein `salary` .

```
SELECT department, MIN(salary) AS "Lowest salary"
FROM employees
GROUP BY department;
```

Dies würde Ihnen sagen, welche Abteilung den Mitarbeiter mit dem niedrigsten Gehalt enthält und wie hoch dieses Gehalt ist. Den `name` des Mitarbeiters mit dem niedrigsten Gehalt in jeder Abteilung herauszufinden, ist ein anderes Problem, das über den Rahmen dieses Beispiels hinausgeht. Siehe "Max. Gruppe".

GROUP BY COUNT-Funktion verwenden

```
SELECT department, COUNT(*) AS "Man_Power"
FROM employees
GROUP BY department;
```

GROUP BY mit HAVING

```
SELECT department, COUNT(*) AS "Man_Power"
FROM employees
GROUP BY department
HAVING COUNT(*) >= 10;
```

Die Verwendung von `GROUP BY ... HAVING` zum Filtern von Aggregatsätzen entspricht der Verwendung von `SELECT ... WHERE` zum Filtern einzelner Datensätze.

Man könnte auch `HAVING Man_Power >= 10` sagen, da `HAVING` "Aliase" versteht.

Gruppe Durch Verwendung von Group Concat

Group Concat wird in MySQL verwendet, um verkettete Werte von Ausdrücken mit mehr als einem Ergebnis pro Spalte abzurufen . Das bedeutet, dass für eine Spalte viele Zeilen ausgewählt werden müssen, z. B. `Name(1):Score(*)`

Name	Ergebnis
Adam	A +
Adam	EIN-
Adam	B

Name	Ergebnis
Adam	C +
Rechnung	D-
John	EIN-

```
SELECT Name, GROUP_CONCAT(Score ORDER BY Score desc SEPERATOR ' ') AS Grades
FROM   Grade
GROUP BY Name
```

Ergebnisse:

```
+------+------------+
| Name | Grades     |
+------+------------+
| Adam | C+ B A- A+ |
| Bill | D-         |
| John | A-         |
+------+------------+
```

GROUP BY mit AGGREGATE-Funktionen

Tabelle ORDERS

```
+---------+------------+----------+-------+--------+
| 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 |
+---------+------------+----------+-------+--------+
```

- **ANZAHL**

Gibt die **Anzahl der Zeilen zurück** , die bestimmte Kriterien in der `WHERE` Klausel erfüllen.

ZB: Anzahl der Bestellungen für jeden Kunden.

```
SELECT customer, COUNT(*) as orders
FROM orders
GROUP BY customer
ORDER BY customer
```

Ergebnis:

```
+----------+--------+
| customer | orders |
+----------+--------+
| Bob      |      3 |
| Carly    |      2 |
| Fred     |      2 |
| Jenny    |      1 |
| Tess     |      2 |
+----------+--------+
```

- **SUMME**

Gibt die **Summe** der ausgewählten Spalte zurück.

ZB: Summe der Summe und der Artikel für jeden Kunden.

```
SELECT customer, SUM(total) as sum_total, SUM(items) as sum_items
FROM orders
GROUP BY customer
ORDER BY customer
```

Ergebnis:

```
+----------+-----------+-----------+
| customer | sum_total | sum_items |
+----------+-----------+-----------+
| Bob      |      2100 |        31 |
| Carly    |      1800 |        15 |
| Fred     |       600 |         3 |
| Jenny    |       200 |         2 |
| Tess     |     14000 |        58 |
+----------+-----------+-----------+
```

- **AVG**

Gibt den **Durchschnittswert** einer Spalte mit numerischem Wert zurück.

ZB: Durchschnittlicher Bestellwert für jeden Kunden.

```
SELECT customer, AVG(total) as avg_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Ergebnis:

```
+----------+-----------+
| customer | avg_total |
+----------+-----------+
| Bob      |       700 |
| Carly    |       900 |
| Fred     |       300 |
| Jenny    |       200 |
| Tess     |      7000 |
+----------+-----------+
```

- **MAX**

Gibt den **höchsten** Wert einer bestimmten Spalte oder eines bestimmten Ausdrucks zurück.

ZB: Höchste Auftragssumme für jeden Kunden.

```
SELECT customer, MAX(total) as max_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Ergebnis:

```
+----------+-----------+
| customer | max_total |
+----------+-----------+
| Bob      |      1300 |
| Carly    |      1000 |
| Fred     |       500 |
| Jenny    |       200 |
| Tess     |     11500 |
+----------+-----------+
```

- **MINDEST**

Gibt den **niedrigsten** Wert einer bestimmten Spalte oder eines bestimmten Ausdrucks zurück.

ZB: Niedrigste Auftragssumme für jeden Kunden.

```
SELECT customer, MIN(total) as min_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Ergebnis:

```
+----------+-----------+
| customer | min_total |
+----------+-----------+
| Bob      |       300 |
| Carly    |       800 |
| Fred     |       100 |
| Jenny    |       200 |
| Tess     |      2500 |
+----------+-----------+
```

Gruppiere nach online lesen: https://riptutorial.com/de/mysql/topic/3523/gruppiere-nach

Kapitel 21: Indizes und Schlüssel

Syntax

- - Erstellen Sie einen einfachen Index

  CREATE INDEX *Indexname* ON *Tabellenname* (*Spaltenname1* [, *Spaltenname2* , ...])

- - Erstellen Sie einen eindeutigen Index

  CREATE UNIQUE INDEX *Indexname* ON *Tabellenname* (*Spaltenname1* [, *Spaltenname2* , ...)

- - Index löschen

  DROP INDEX **index_name** ON **tabelle_name** [**algorithm_option** | **lock_option**] ...

  algorithm_option: ALGORITHM [=] {DEFAULT | INPLACE | COPY}

  lock_option: LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}

Bemerkungen

Konzepte

Ein Index in einer MySQL-Tabelle funktioniert wie ein Index in einem Buch.

Nehmen wir an, Sie haben ein Buch über Datenbanken und möchten Informationen zum Speicher finden. Ohne einen Index (vorausgesetzt, es gibt keine andere Hilfe, z. B. ein Inhaltsverzeichnis), müssen Sie die Seiten einzeln durchgehen, bis Sie das Thema gefunden haben (das ist ein "vollständiger Tabellenscan"). Auf der anderen Seite enthält ein Index eine Liste mit Schlüsselwörtern, sodass Sie den Index nachschlagen und feststellen können, dass Speicher auf den Seiten 113-120, 231 und 354 erwähnt wird. Dann könnten Sie diese Seiten direkt durchsuchen, ohne zu suchen (das ist eine Suche mit einem Index, etwas schneller).

Natürlich hängt der Nutzen des Index von vielen Faktoren ab - einige Beispiele, die das obige Gleichnis verwenden:

- Wenn Sie ein Buch über Datenbanken hatten und das Wort "Datenbank" indizierten, wird es möglicherweise auf den Seiten 1-59, 61-290 und 292-400 erwähnt. Das sind viele Seiten, und in einem solchen Fall ist der Index keine große Hilfe und es könnte schneller sein, die Seiten einzeln zu durchlaufen. (In einer Datenbank ist dies "schlechte Selektivität".)
- Bei einem 10-seitigen Buch ist es nicht sinnvoll, einen Index zu erstellen, da am Ende ein 10-seitiges Buch mit einem 5-seitigen Index anfangen kann, was einfach dumm ist - scannen Sie einfach die 10 Seiten und machen Sie es fertig .

- Der Index muss auch nützlich sein - generell gibt es keinen Sinn für die Indexierung, beispielsweise die Häufigkeit des Buchstabens "L" pro Seite.

Examples

Index erstellen

```
-- Create an index for column 'name' in table 'my_table'
CREATE INDEX idx_name ON my_table(name);
```

Erstellen Sie einen eindeutigen Index

Ein eindeutiger Index verhindert das Einfügen von duplizierten Daten in eine Tabelle. `NULL` Werte können in die Spalten eingefügt werden, die Teil des eindeutigen Index sind (da sich ein `NULL` Wert definitionsgemäß von jedem anderen Wert unterscheidet, einschließlich eines anderen `NULL` Werts)

```
-- Creates a unique index for column 'name' in table 'my_table'
CREATE UNIQUE INDEX idx_name ON my_table(name);
```

Index löschen

```
-- Drop an index for column 'name' in table 'my_table'
DROP INDEX idx_name ON my_table;
```

Zusammengesetzten Index erstellen

Dadurch wird ein zusammengesetzter Index für beide Schlüssel, `mystring` und `mydatetime` und Abfragen mit beiden Spalten in der `WHERE` Klausel werden beschleunigt.

```
CREATE INDEX idx_mycol_myothercol ON my_table(mycol, myothercol)
```

Hinweis: Die Reihenfolge ist wichtig! Wenn die Suchabfrage nicht beide Spalten in der `WHERE` Klausel enthält, kann nur der Index ganz links verwendet werden. In diesem Fall wird eine Abfrage mit `mycol` in der `WHERE` den Index verwenden, wird eine Abfrage für die Suche `myothercol` **ohne** auch für die Suche `mycol` **nicht.** Weitere Informationen finden Sie in diesem Blogbeitrag .

Hinweis: Aufgrund der Arbeitsweise von BTREE sollten Spalten, die normalerweise in Bereichen abgefragt werden, den Wert ganz rechts einnehmen. Beispielsweise werden `DATETIME` Spalten normalerweise abgefragt wie `WHERE datecol > '2016-01-01 00:00:00'` . BTREE-Indizes behandeln Bereiche sehr effizient, jedoch nur, wenn die als Bereich abgefragte Spalte die letzte im zusammengesetzten Index ist.

AUTO_INCREMENT-Taste

```
CREATE TABLE (
```

```
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    ...
    PRIMARY KEY(id),
    ...  );
```

Hauptanmerkungen:

- Beginnt mit 1 und erhöht sich automatisch um 1, wenn Sie es unter `INSERT` nicht angeben oder als `NULL` angeben.
- Die IDs unterscheiden sich immer voneinander, aber ...
- Machen Sie keine Annahmen (keine Lücken, fortlaufend erzeugt, nicht wiederverwendet usw.), die sich auf die Werte der id beziehen, außer dass sie zu einem bestimmten Zeitpunkt eindeutig sind.

Subtile Notizen:

- Beim Neustart des Servers wird der Wert "next" als `MAX(id)+1` berechnet.
- Wenn bei der letzten Operation vor dem Herunterfahren oder Absturz die höchste ID gelöscht wurde, *kann* diese ID erneut verwendet werden (dies ist von der Engine abhängig). *Vertrauen Sie also nicht, dass auto_increments dauerhaft eindeutig ist* . Sie sind zu jedem Zeitpunkt nur einzigartig.
- Für Multi-Master- oder Cluster-Lösungen siehe `auto_increment_offset` und `auto_increment_increment` .
- Es ist in Ordnung, etwas anderes als den `PRIMARY KEY` und einfach `INDEX(id)` . (Dies ist in einigen Situationen eine Optimierung.)
- Die Verwendung von `AUTO_INCREMENT` als " `PARTITION` Taste" ist selten von Vorteil. mach etwas anderes.
- Verschiedene Operationen *können* Werte "verbrennen". Dies geschieht, wenn Werte vorab zugewiesen werden und dann nicht verwendet werden: `INSERT IGNORE` (mit dup-Schlüssel), `REPLACE` (`DELETE` plus `INSERT`) und andere. `ROLLBACK` ist ein weiterer Grund für Lücken in IDs.
- Bei der Replikation können Sie nicht darauf vertrauen, dass die IDs in aufsteigender Reihenfolge beim Slave ankommen. Obwohl die IDs in fortlaufender Reihenfolge zugewiesen werden, werden die InnoDB-Anweisungen in der `COMMIT` Reihenfolge an die Slaves gesendet.

Indizes und Schlüssel online lesen: https://riptutorial.com/de/mysql/topic/1748/indizes-und-schlussel

Kapitel 22: Installieren Sie den Mysql-Container mit Docker-Compose

Examples

Einfaches Beispiel mit Docker-Compose

Dies ist ein einfaches Beispiel zum Erstellen eines MySQL-Servers mit Docker

1. Erstellen Sie **docker-compose.yml** :

Hinweis: Wenn Sie für alle Ihre Projekte denselben Container verwenden möchten, müssen Sie in Ihrem HOME_PATH einen PFAD erstellen. Wenn Sie es für jedes Projekt erstellen möchten, können Sie ein **Andockverzeichnis** in Ihrem Projekt erstellen.

```
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.- führe es aus:

```
cd PATH_TO_DOCKER-COMPOSE.YML
docker-compose up -d
```

3.- Verbinden Sie sich mit dem Server

```
mysql -h 127.0.0.1 -u root -P 3306 -p rootpw
```

Hurra!!

4.- Server stoppen

```
docker-compose stop
```

Installieren Sie den Mysql-Container mit Docker-Compose online lesen: https://riptutorial.com/de/mysql/topic/4458/installieren-sie-den-mysql-container-mit-docker-compose

Kapitel 23: JOINS: Join 3 Tabelle mit dem gleichen Namen von ID.

Examples

Verknüpfen Sie 3 Tabellen in einer Spalte mit demselben Namen

```
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)
)
```

Nach dem Erstellen der Tabellen können Sie eine Auswahlabfrage ausführen, um die IDs aller drei Tabellen zu erhalten, die gleich sind

```
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
```

JOINS: Join 3 Tabelle mit dem gleichen Namen von ID. online lesen: https://riptutorial.com/de/mysql/topic/9921/joins--join-3-tabelle-mit-dem-gleichen-namen-von-id-

Kapitel 24: JSON

Einführung

Ab MySQL 5.7.8 unterstützt MySQL einen nativen JSON-Datentyp, der einen effizienten Zugriff auf Daten in JSON-Dokumenten (JavaScript Object Notation) ermöglicht.
https://dev.mysql.com/doc/refman/5.7/de/json.html

Bemerkungen

Ab MySQL 5.7.8 wird MySQL mit einem JSON-Typ ausgeliefert. Viele Devs haben JSON-Daten für eine Protokollzeit in Textspalten gespeichert. Der JSON-Typ unterscheidet sich jedoch. Die Daten werden nach der Überprüfung im Binärformat gespeichert. Dadurch wird der Aufwand für die Analyse des Textes bei jedem Lesevorgang vermieden.

Examples

Erstellen Sie eine einfache Tabelle mit einem Primärschlüssel und einem JSON-Feld

```
CREATE TABLE table_name (
    id INT NOT NULL AUTO_INCREMENT,
    json_col JSON,
    PRIMARY KEY(id)
);
```

Fügen Sie eine einfache JSON ein

```
INSERT INTO
    table_name (json_col)
VALUES
    ('{"City": "Galle", "Description": "Best damn city in the world"}');
```

Das ist so einfach, wie es sein kann, aber beachten Sie, dass, weil JSON-Wörterbuchschlüssel in doppelte Anführungszeichen gesetzt werden müssen, das Ganze in einfache Anführungszeichen gesetzt werden sollte. Wenn die Abfrage erfolgreich ist, werden die Daten in einem binären Format gespeichert.

Fügen Sie gemischte Daten in ein JSON-Feld ein.

Dadurch wird ein Json-Wörterbuch eingefügt, bei dem eines der Member ein Array von Strings in der Tabelle ist, die in einem anderen Beispiel erstellt wurde.

```
INSERT INTO myjson(dict)
VALUES('{"opening":"Sicilian","variations":["pelikan","dragon","najdorf"]}');
```

Beachten Sie, dass Sie bei der Verwendung von einfachen und doppelten Anführungszeichen vorsichtig sein müssen. Das Ganze muss in einfache Anführungszeichen gesetzt werden.

JSON-Feld aktualisieren

Im vorherigen Beispiel haben wir gesehen, wie gemischte Datentypen in ein JSON-Feld eingefügt werden können. Was ist, wenn wir dieses Feld aktualisieren möchten? Wir werden *scheveningen* zu den Array- `variations` im vorherigen Beispiel hinzufügen.

```
UPDATE
    myjson
SET
    dict=JSON_ARRAY_APPEND(dict,'$.variations','scheveningen')
WHERE
    id = 2;
```

Anmerkungen:

1. Das `$.variations` Array in unserem Json-Wörterbuch. Das $ -Symbol steht für die Json-Dokumentation. Eine vollständige Erklärung der von mysql erkannten json-Pfade finden Sie unter https://dev.mysql.com/doc/refman/5.7/de/json-path-syntax.html
2. Da wir noch kein Beispiel für die Abfrage mit json-Feldern haben, wird in diesem Beispiel der Primärschlüssel verwendet.

Wenn wir `SELECT * FROM myjson` wir sehen

```
+----+-------------------------------------------------------------------------------------
-+
| id | dict
|
+---+--------------------------------------------------------------------------------------
+
| 2  | {"opening": "Sicilian", "variations": ["pelikan", "dragon", "najdorf", "scheveningen"]}
|
+----+-------------------------------------------------------------------------------------
-+
1 row in set (0.00 sec)
```

CAST-Daten an JSON-Typ

Dies konvertiert gültige Json-Strings in den MySQL-JSON-Typ:

```
 SELECT CAST('[1,2,3]' as JSON) ;
 SELECT CAST('{"opening":"Sicilian","variations":["pelikan","dragon","najdorf"]}' as JSON);
```

Erstellen Sie ein Json-Objekt und ein Array

`JSON_OBJECT` erstellt JSON-Objekte:

```
 SELECT JSON_OBJECT('key1',col1 , 'key2',col2 , 'key3','col3') as myobj;
```

`JSON_ARRAY` erstellt auch ein JSON-Array:

```
 SELECT JSON_ARRAY(col1,col2,'col3') as myarray;
```

Hinweis: myobj.key3 und myarray [2] sind als feste Zeichenfolge "col3".

Auch gemischte JSON-Daten:

```
 SELECT JSON_OBJECT("opening","Sicilian",
"variations",JSON_ARRAY("pelikan","dragon","najdorf") ) as mymixed ;
```

JSON online lesen: https://riptutorial.com/de/mysql/topic/2985/json

Kapitel 25: Kommentar Mysql

Bemerkungen

Der `--` Kommentarstil, der ein Leerzeichen benötigt, unterscheidet sich vom SQL-Standard , der keinen Platz benötigt.

Examples

Kommentare hinzufügen

Es gibt drei Arten von Kommentaren:

```
 # 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
*/
```

Beispiel:

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

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

Die `--` Methode erfordert , dass ein Raum folgt die `--` vor dem Kommentar beginnt, sonst wird es als Befehl interpretiert werden und in der Regel einen Fehler verursachen.

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

Tabellendefinitionen kommentieren

```
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.';
```

Die Verwendung von `=` nach `COMMENT` ist optional. (Offizielle Dokumente)

Diese Kommentare werden im Gegensatz zu den anderen mit dem Schema gespeichert und können über `SHOW CREATE TABLE` oder aus `information_schema` abgerufen werden.

Kommentar Mysql online lesen: https://riptutorial.com/de/mysql/topic/2337/kommentar-mysql

Kapitel 26: Konfiguration und Abstimmung

Bemerkungen

Die Konfiguration erfolgt auf drei verschiedene Arten:

- Befehlszeilenoptionen
- die `my.cnf` Konfigurationsdatei
- Setzen von Variablen vom Server aus

Befehlszeilenoptionen haben die Form `mysqld --long-parameter-name=value --another-parameter`. Die gleichen Parameter können in der Konfigurationsdatei `my.conf` abgelegt werden. *Einige* Parameter können mithilfe von Systemvariablen in MySQL konfiguriert werden. In der offiziellen Dokumentation finden Sie eine vollständige Liste der Parameter.

Variablen können Strich haben `-` oder unterstreichen `_`. Um das `=` können Leerzeichen stehen. Große Zahlen können durch `K`, `M`, `G` für Kilo-, Mega- und Giga- ergänzt werden. Eine Einstellung pro Zeile.

Flags: Normalerweise sind `ON` und `1` gleichbedeutend, ebenso für `OFF` und `0`. Einige Flaggen haben nichts hinter sich.

Wenn Sie die Einstellungen in `my.cnf`, müssen sich alle Einstellungen für den *Server* im Abschnitt `[mysqld]` also nicht blind Einstellungen am Ende der Datei hinzu. (Hinweis: Bei Tools, die mehreren MySQL-Instanzen erlauben, eine my.cnf gemeinsam zu nutzen, können die Abschnittsnamen unterschiedlich sein.)

Examples

InnoDB-Leistung

Es gibt Hunderte von Einstellungen, die in my.cnf platziert werden können. Für den 'Lite'-Benutzer von MySQL sind sie nicht so wichtig.

Sobald Ihre Datenbank einmalig ist, sollten Sie die folgenden Parameter einstellen:

```
innodb_buffer_pool_size
```

Dies sollte auf etwa 70% des *verfügbaren* Arbeitsspeichers festgelegt werden (wenn Sie über mindestens 4 GB RAM verfügen; ein kleinerer Prozentsatz, wenn Sie eine kleine VM oder einen antiken Computer haben). Die Einstellung steuert den von InnoDB ENGINE verwendeten Cache-Speicher. Daher ist es für die Leistung von InnoDB sehr wichtig.

Parameter zum Einfügen großer Daten

Wenn Sie Bilder oder Videos in der Spalte speichern müssen, müssen wir den Wert entsprechend

Ihrer Anwendung ändern

max_allowed_packet = 10M

M ist Mb, G in Gb, K in Kb

Erhöhen Sie das Zeichenfolgenlimit für group_concat

`group_concat` wird verwendet, um Nicht-Null-Werte in einer `group` zu verketten. Die maximale Länge der resultierenden Zeichenfolge kann mit der Option `group_concat_max_len` :

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

Durch das Festlegen der `GLOBAL` Variablen wird eine dauerhafte Änderung sichergestellt, während durch das Festlegen der `SESSION` Variable der Wert für die aktuelle Sitzung festgelegt wird.

Minimale InnoDB-Konfiguration

Dies ist ein absolutes Minimum für MySQL-Server, die InnoDB-Tabellen verwenden. Bei InnoDB ist kein Abfrage-Cache erforderlich. Speicherplatz freigeben, wenn eine Tabelle oder Datenbank `DROP` ed ist. Wenn Sie SSDs verwenden, ist das Flushing ein redundanter Vorgang (SDDs sind nicht sequenziell).

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

Parallelität

Stellen Sie sicher, dass wir mehr als die Standard-4-Threads erstellen können, indem Sie `innodb_thread_concurrency` auf unendlich (0) setzen. Dadurch kann InnoDB anhand einer optimalen Ausführung entscheiden.

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

Festplattenauslastung

Legen Sie die Kapazität (normale Last) und die Kapazität_max (absolutes Maximum) von IOPS für MySQL fest. Der Standardwert 200 ist für HDDs in Ordnung, aber heutzutage werden SSDs, die Tausende IOPS-fähig sind, wahrscheinlich dazu veranlasst, diese Anzahl anzupassen. Es gibt viele Tests, die Sie zur Bestimmung von IOPS ausführen können. Die oben genannten Werte sollten annähernd die Grenze sein, *wenn Sie einen dedizierten MySQL Server betreiben* . Wenn Sie andere Dienste auf demselben Computer ausführen, sollten Sie die Zuteilung entsprechend vornehmen.

```
innodb_io_capacity = 2500
```

```
innodb_io_capacity_max = 3000
```

RAM-Auslastung

Stellen Sie den verfügbaren Arbeitsspeicher auf MySQL ein. Während die Faustregel 70-80% beträgt, hängt dies wirklich davon ab, ob Ihre Instanz für MySQL reserviert ist und wie viel RAM verfügbar ist. *Verschwenden* Sie keinen RAM (dh Ressourcen), wenn Sie viel zur Verfügung haben.

```
innodb_buffer_pool_size = 10G
```

Sichere MySQL-Verschlüsselung

Die Standardverschlüsselung `aes-128-ecb` verwendet den EZB-Modus (Electronic Codebook). Fügen Sie stattdessen der Konfigurationsdatei Folgendes hinzu:

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

Konfiguration und Abstimmung online lesen:
https://riptutorial.com/de/mysql/topic/3134/konfiguration-und-abstimmung

Kapitel 27: Konvertierung von MyISAM nach InnoDB

Examples

Grundlegende Konvertierung

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

Dadurch wird die Tabelle konvertiert, es werden jedoch keine Unterschiede zwischen den Engines berücksichtigt. Die meisten Unterschiede spielen keine Rolle, besonders für kleine Tische. Bei belebten Tischen sollten jedoch andere Überlegungen in Betracht gezogen werden. *Überlegungen zur Konvertierung*

Alle Tabellen in einer Datenbank konvertieren

Um alle Tabellen in einer Datenbank zu konvertieren, verwenden Sie Folgendes:

```
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';
```

> **HINWEIS:** Sie sollten mit Ihrer Datenbank verbunden sein, damit die Funktion `DATABASE()` funktioniert. Andernfalls wird `NULL` . Dies gilt vor allem für den Standard-MySQL-Client, der mit dem Server geliefert wird, da eine Verbindung ohne Angabe einer Datenbank möglich ist.

Führen Sie diese SQL-Anweisung aus, um alle `MyISAM` Tabellen in Ihrer Datenbank abzurufen.

Kopieren Sie schließlich die Ausgabe und führen Sie SQL-Abfragen daraus aus.

Konvertierung von MyISAM nach InnoDB online lesen:
https://riptutorial.com/de/mysql/topic/3135/konvertierung-von-myisam-nach-innodb

Kapitel 28: LADEN DATEN INFILE

Syntax

1. LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'Dateiname'
2. INTO TABLE Tabellenname
3. [Zeichensatz-Zeichensatz]
4. [{FELDER | SPALTEN} [BEENDET DURCH 'string'] [[OPTIONAL] UMFASST DURCH 'char']]
5. [LINES [STARTING BY 'string'] [TERMINATED BY 'string']]
6. [IGNORE-Nummer {LINES | REIHEN}]
7. [(col_name_oder_user_var, ...)]
8. [SET col_name = Ausdruck, ...]

Examples

Verwenden von LOAD DATA INFILE, um große Datenmengen in die Datenbank zu laden

Stellen Sie sich das folgende Beispiel vor, vorausgesetzt, Sie verfügen über eine durch;

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

Erstellen Sie die Tabelle zum Einfügen.

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

Verwenden Sie die folgende Abfrage, um die Werte in diese Tabelle einzufügen.

```
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)
```

Betrachten Sie den Fall, in dem das Datumsformat kein Standard ist.

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

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

In diesem Fall können Sie das Format der `dob` Spalte vor dem Einfügen ändern.

```
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');
```

In diesem Beispiel von LOAD DATA INFILE werden nicht alle verfügbaren Funktionen angegeben.

Sie können weitere Referenzen auf LOAD DATA INFILE siehe hier .

Importieren Sie eine CSV-Datei in eine MySQL-Tabelle

Mit dem folgenden Befehl werden CSV-Dateien in eine MySQL-Tabelle mit denselben Spalten importiert, wobei die CSV-Zitat- und Escape-Regeln beachtet werden.

```
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
```

Laden Sie Daten mit Duplikaten

Wenn Sie den Befehl `LOAD DATA INFILE` , um eine Tabelle mit vorhandenen Daten zu `LOAD DATA INFILE` , werden Sie häufig feststellen, dass der Import aufgrund von Duplikaten fehlschlägt. Es gibt mehrere Möglichkeiten, dieses Problem zu überwinden.

LOAD DATA LOKAL

Wenn diese Option auf Ihrem Server aktiviert wurde, kann mit ihr eine Datei geladen werden, die auf dem Client-Computer und nicht auf dem Server vorhanden ist. Ein Nebeneffekt ist, dass doppelte Zeilen für eindeutige Werte ignoriert werden.

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

LOAD DATA INFILE 'fname' REPLACE

Wenn das Schlüsselwort replace verwendet wird, führen doppelte eindeutige oder Primärschlüssel dazu, dass die vorhandene Zeile durch neue ersetzt wird

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

LOAD DATA INFILE 'fname' IGNORE

Im Gegensatz zu `REPLACE` werden vorhandene Zeilen beibehalten und neue ignoriert. Dieses Verhalten ähnelt dem oben beschriebenen `LOCAL` . Die Datei muss jedoch nicht auf dem Clientcomputer vorhanden sein.

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

Laden über Zwischentabelle

Das Ignorieren oder Ersetzen aller Duplikate ist manchmal nicht die ideale Option. Möglicherweise müssen Sie Entscheidungen basierend auf dem Inhalt anderer Spalten treffen. In diesem Fall ist es am besten, in eine Zwischentabelle zu laden und von dort zu übertragen.

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

Import Export

einführen

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

Export

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

LADEN DATEN INFILE online lesen: https://riptutorial.com/de/mysql/topic/2356/laden-daten-infile

Kapitel 29: Leistungsoptimierung

Syntax

- Verwenden Sie DISTINCT und GROUP BY nicht in demselben SELECT.
- Nicht über OFFSET paginieren, "sich erinnern, wo Sie aufgehört haben".
- WO (a, b) = (22,33) optimiert überhaupt nicht.
- Sagen Sie explizit ALL oder DISTINCT nach UNION - es erinnert Sie daran, zwischen dem schnelleren ALL oder dem langsameren DISTINCT zu wählen.
- Verwenden Sie SELECT * nicht, insbesondere wenn Sie keine TEXT- oder BLOB-Spalten haben. Es gibt Overhead in TMP-Tabellen und Übertragung.
- Es ist schneller, wenn GROUP BY und ORDER BY genau dieselbe Liste haben können.
- Verwenden Sie nicht FORCE INDEX. Es kann heute helfen, wird aber wahrscheinlich morgen weh tun.

Bemerkungen

Siehe auch Diskussionen zu ORDER BY, LIKE, REGEXP usw. Hinweis: Dies muss mit Links und weiteren Themen bearbeitet werden.

Kochbuch zur Erstellung optimaler Indizes .

Examples

Fügen Sie den richtigen Index hinzu

Dies ist ein riesiges Thema, aber es ist auch das wichtigste "Leistungsproblem".

Die wichtigste Lektion für einen Anfänger ist das Erlernen von "zusammengesetzten" Indizes. Hier ein kurzes Beispiel:

```
INDEX(last_name, first_name)
```

ist hervorragend für diese:

```
WHERE last_name = '...'
WHERE first_name = '...' AND last_name = '...'   -- (order in WHERE does not matter)
```

aber nicht für

```
WHERE first_name = '...'   -- order in INDEX _does_ matter
WHERE last_name = '...' OR first_name = '...'   -- "OR" is a killer
```

Stellen Sie den Cache richtig ein

`innodb_buffer_pool_size` sollte etwa 70% des verfügbaren Arbeitsspeichers `innodb_buffer_pool_size` .

Vermeiden Sie ineffiziente Konstrukte

```
x IN ( SELECT ... )
```

verwandeln Sie sich in einen `JOIN`

Wenn möglich, vermeiden Sie `OR` .

Verbergen Sie eine indizierte Spalte in einer Funktion nicht, z. B. `WHERE DATE(x) = ...` ; umformulieren als `WHERE x = ...`

Sie können `WHERE LCASE(name1) = LCASE(name2)` generell vermeiden, indem Sie eine geeignete Sortierung verwenden.

Verwenden Sie `OFFSET` für "Paginierung", sondern "Erinnern Sie sich, wo Sie `OFFSET` ".

Vermeiden Sie `SELECT * ...` (es sei denn, Sie debuggen).

Anmerkung zu Maria Deleva, Barranka, Batsu: Dies ist ein Platzhalter; Bitte entfernen Sie diese Elemente, wenn Sie umfassende Beispiele erstellen. Nachdem Sie diejenigen getan haben, die Sie können, werde ich mich um den Rest kümmern und / oder sie werfen.

Negative

Hier sind einige Dinge, die die Leistung wahrscheinlich nicht verbessern. Sie stammen aus veralteten Informationen und / oder Naivität.

- InnoDB hat sich soweit verbessert, dass MyISAM wahrscheinlich nicht besser ist.
- `PARTITIONing` bietet selten Leistungsvorteile; Es kann sogar die Leistung beeinträchtigen.
- Wenn Sie `query_cache_size` als 100 `query_cache_size` wird *die* Leistung normalerweise `query_cache_size` .
- Das Erhöhen `my.cnf` Werte in `my.cnf` kann zu 'Swapping' führen, was ein *ernstes* Leistungsproblem darstellt.
- "Prefix-Indizes" (wie `INDEX(foo(20))`)) sind im Allgemeinen unbrauchbar.
- `OPTIMIZE TABLE` ist fast immer unbrauchbar. (Und es beinhaltet das Sperren der Tabelle.)

Habe einen INDEX

Das Wichtigste für die Beschleunigung einer Abfrage in einer nicht-kleinen Tabelle ist der geeignete Index.

```
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
```

Versteck dich nicht in der Funktion

Ein häufiger Fehler ist das Ausblenden einer indizierten Spalte in einem Funktionsaufruf. Zum Beispiel kann ein Index nicht helfen:

```
WHERE DATE(dt) = '2000-01-01'
```

Anstelle von `INDEX(dt)` können diese den Index verwenden:

```
WHERE dt = '2000-01-01'  -- if `dt` is datatype `DATE`
```

Dies funktioniert für `DATE` , `DATETIME` , `TIMESTAMP` und sogar `DATETIME(6)` (Mikrosekunden):

```
WHERE dt >= '2000-01-01'
  AND dt  < '2000-01-01' + INTERVAL 1 DAY
```

ODER

Im Allgemeinen tötet `OR` Optimierung.

```
WHERE a = 12 OR b = 78
```

kann `INDEX(a,b)` nicht verwenden und kann `INDEX(a), INDEX(b)` über "Index Merge" verwenden. Index Merge ist besser als nichts, aber nur kaum.

```
WHERE x = 3 OR x = 5
```

wird in umgewandelt

```
WHERE x IN (3, 5)
```

die einen Index mit verwenden `x` drin.

Unterabfragen

Unterabfragen gibt es in verschiedenen Ausführungen und sie haben unterschiedliche Optimierungsmöglichkeiten. Beachten Sie zunächst, dass Unterabfragen entweder "korreliert" oder "unkorreliert" sein können. Korreliert bedeutet, dass sie von einem Wert außerhalb der

Unterabfrage abhängen. Dies bedeutet im Allgemeinen , dass die Unterabfrage *muss* für jeden Außenwert neu bewertet werden.

Diese korrelierte Unterabfrage ist oft ziemlich gut. Hinweis: Es muss höchstens ein Wert zurückgegeben werden. Es ist oft nützlich als Alternative zu einem `LEFT JOIN` , wenn auch nicht unbedingt schneller.

```
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 ...
```

Dies ist normalerweise nicht korreliert:

```
SELECT ...
    FROM ( SELECT ... ) AS a
    JOIN b ON ...
```

Hinweise zum `FROM-SELECT` :

- Wenn es 1 Reihe zurückgibt, großartig.
- Ein gutes Paradigma (wieder "1 Zeile") ist, dass die Unterabfrage `( SELECT @n := 0 )` , wodurch eine @ -Variable für die Verwendung im Rest der Abfrage initialisiert wird.
- Wenn es viele Zeilen zurückgibt *und* der `JOIN` auch `( SELECT ... )` mit vielen Zeilen ist, kann die Effizienz schrecklich sein. Vor 5.6 gab es keinen Index, also wurde es ein `CROSS JOIN` . 5.6+ beinhaltet das Ableiten des besten Index für die temporären Tabellen und das Generieren dieses Werts, um ihn nach der Auswahl mit `SELECT` wegzuwerfen.

JOIN + GROUP BY

Ein häufiges Problem, das zu einer ineffizienten Abfrage führt, sieht etwa so aus:

```
SELECT ...
    FROM a
    JOIN b  ON ...
    WHERE ...
    GROUP BY a.id
```

Zuerst erweitert der `JOIN` die Anzahl der Zeilen. dann spult die `GROUP BY` die Anzahl der Reihen in `a` .

Es gibt möglicherweise keine guten Entscheidungen, um dieses Explosions-Implode-Problem zu lösen. Eine mögliche Option besteht darin, den `JOIN` in eine korrelierte Unterabfrage in `SELECT` umzuwandeln. Dadurch entfällt auch `GROUP BY` .

Leistungsoptimierung online lesen: https://riptutorial.com/de/mysql/topic/4292/leistungsoptimierung

Kapitel 30: Limit und Offset

Syntax

- SELECT Spalte_1 [, Spalte_2]
  FROM table_1
  ORDER BY order_column
  LIMIT row_count [OFFSET row_offset]
- SELECT Spalte_1 [, Spalte_2]
  FROM table_1
  ORDER BY order_column
  LIMIT [row_offset,] row_count

Bemerkungen

"Limit" kann "Maximale Anzahl von Zeilen in einer Tabelle" bedeuten.

"Offset" bedeutet Auswahl aus `row` (nicht durch Primärschlüsselwert oder Felddatenwert zu verwechseln)

Examples

Limit- und Offset-Beziehung

Berücksichtigt die folgende `users` :

Ich würde	Nutzername
1	Benutzer1
2	User2
3	Benutzer3
4	User4
5	Benutzer5

Um die Anzahl der Zeilen in der Ergebnismenge einer `SELECT` Abfrage zu beschränken , kann die `LIMIT` Klausel zusammen mit einer oder zwei positiven Ganzzahlen als Argumente verwendet werden (Null eingeschlossen).

`LIMIT` Klausel mit einem Argument

Wenn ein Argument verwendet wird, ist die Ergebnismenge nur auf die auf folgende Weise angegebene Anzahl beschränkt:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2
```

Ich würde	Nutzername
1	Benutzer1
2	User2

Wenn der Wert des Arguments `0` , ist die Ergebnismenge leer.

`ORDER BY` auch, dass die `ORDER BY` Klausel wichtig sein kann, um die ersten Zeilen der Ergebnismenge anzugeben, die angezeigt werden sollen (wenn Sie nach einer anderen Spalte sortieren).

`LIMIT` Klausel mit zwei Argumenten

Wenn zwei Argumente in einer `LIMIT` Klausel verwendet werden:

- Das **erste** Argument steht für die Zeile, aus der die Zeilen der Ergebnismenge dargestellt werden. Diese Zahl wird häufig als **Versatz angegeben** , da sie die Zeile darstellt, die vor der ersten Zeile der eingeschränkten Ergebnismenge liegt. Dadurch kann das Argument `0` als Wert erhalten und somit die erste Zeile der uneingeschränkten Ergebnismenge berücksichtigen.
- Das **zweite** Argument gibt die maximale Anzahl von Zeilen an, die in der Ergebnismenge zurückgegeben werden sollen (ähnlich dem Beispiel des einen Arguments).

Daher die Abfrage:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2, 3
```

Stellt die folgende Ergebnismenge dar:

Ich würde	Nutzername
3	Benutzer3
4	User4
5	Benutzer5

Beachten Sie, dass wenn das **Offset-** Argument `0` , die Ergebnismenge einer `LIMIT` Klausel mit einem Argument entspricht. Dies bedeutet, dass die folgenden zwei Abfragen:

```
SELECT * FROM users ORDER BY id ASC LIMIT 0, 2

SELECT * FROM users ORDER BY id ASC LIMIT 2
```

Produzieren Sie die gleiche Ergebnismenge:

Ich würde	Nutzername
1	Benutzer1
2	User2

Schlüsselwort `OFFSET` : alternative Syntax

Eine alternative Syntax für die `LIMIT` Klausel mit zwei Argumenten besteht in der Verwendung des Schlüsselworts `OFFSET` nach dem ersten Argument auf folgende Weise:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2 OFFSET 3
```

Diese Abfrage würde die folgende Ergebnismenge zurückgeben:

Ich würde	Nutzername
3	Benutzer3
4	User4

Beachten Sie, dass in dieser alternativen Syntax die Positionen der Argumente ausgetauscht werden:

- Das **erste** Argument steht für die Anzahl der Zeilen, die in der Ergebnismenge zurückgegeben werden sollen.
- Das **zweite** Argument steht für den Versatz.

Limit und Offset online lesen: https://riptutorial.com/de/mysql/topic/548/limit-und-offset

Kapitel 31: LÖSCHEN

Syntax

- DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM Tabelle [WHERE-Bedingungen] [ORDER BY-Ausdruck [ASC | DESC]] [LIMIT number_rows]; /// Syntax zum Löschen von Zeilen aus einer Tabelle

Parameter

Parameter	Einzelheiten
NIEDRIGE PRIORITÄT	Wenn `LOW_PRIORITY` ist, wird das Löschen so lange verzögert, bis keine Prozesse aus der Tabelle gelesen werden
IGNORIEREN	Wenn `IGNORE` wird, werden alle während des Löschvorgangs `IGNORE` Fehler ignoriert
Tabelle	Die Tabelle, aus der Sie Datensätze löschen möchten
WHERE Bedingungen	Die Bedingungen, die erfüllt sein müssen, damit die Datensätze gelöscht werden. Wenn keine Bedingungen angegeben werden, werden alle Datensätze aus der Tabelle gelöscht
ORDER BY-Ausdruck	Wenn `ORDER BY` ist, werden Datensätze in der angegebenen Reihenfolge gelöscht
GRENZE	Er steuert die maximale Anzahl von Datensätzen, die aus der Tabelle gelöscht werden sollen. `number_rows` werden gelöscht.

Examples

Mit Where-Klausel löschen

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

Dadurch werden alle Zeilen aus der Tabelle `field_one` bei denen der Inhalt von `field_one` für diese Zeile mit 'value_one' übereinstimmt.

Die `WHERE` Klausel funktioniert genauso wie eine select, also können Dinge wie `>` , `<` , `<>` oder `LIKE` verwendet werden.

Hinweis: In der Löschabfrage müssen bedingte Klauseln (WHERE, LIKE) verwendet werden. Wenn Sie keine Bedingungsklauseln verwenden, werden alle Daten aus dieser Tabelle gelöscht.

Löschen Sie alle Zeilen aus einer Tabelle

```
DELETE FROM table_name ;
```

Dadurch werden alle Zeilen und alle Zeilen aus der Tabelle gelöscht. Es ist das grundlegendste Beispiel für die Syntax. Es zeigt auch, dass `DELETE` Anweisungen wirklich mit besonderer Vorsicht verwendet werden sollten, da sie eine Tabelle leeren können, wenn die `WHERE` Klausel weggelassen wird.

LIMITing löscht

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

Dies funktioniert auf dieselbe Weise wie das Beispiel 'Delete with Where', aber der Löschvorgang wird abgebrochen, sobald die begrenzte Anzahl von Zeilen entfernt wurde.

Wenn Sie Zeilen zum Löschen einschränken, beachten Sie, dass die erste Zeile gelöscht wird, die den Kriterien entspricht. Es ist möglicherweise nicht das, was Sie erwarten würden, da die Ergebnisse unsortiert werden können, wenn sie nicht explizit angeordnet werden.

Multi-Table löscht

Die `DELETE` Anweisung von MySQL kann das `JOIN` Konstrukt verwenden, wodurch auch angegeben werden kann, aus welchen Tabellen gelöscht werden soll. Dies ist nützlich, um verschachtelte Abfragen zu vermeiden. In Anbetracht des Schemas:

```
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');
```

Ich würde	Name	Geschlecht
1	Kathy	f
2	John	m
3	Paul	m

Ich würde	Name	Geschlecht
4	Kim	f

Ich würde	ownerId	Name	Farbe
1	1	Rover	Beige
2	2	Bläschen	lila
4	1	Rover2	Weiß

Wenn wir Pauls Haustiere entfernen möchten, die Aussage

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

kann umgeschrieben werden als:

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

1 Zeile gelöscht
Spot wird aus Pets gelöscht

`p1` und `p2` sind Aliasnamen für die Tabellennamen, besonders nützlich für lange Tabellennamen und für eine gute Lesbarkeit.

So entfernen Sie sowohl die Person als auch das Haustier:

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

2 Zeilen gelöscht
Spot wird aus Pets gelöscht
Paul wird aus den Leuten gestrichen

fremde Schlüssel

Wenn die DELETE-Anweisung Tabellen mit einer foreing-Schlüsseleinschränkung enthält, kann

das Optimierungsprogramm die Tabellen in einer Reihenfolge verarbeiten, die der Beziehung nicht folgt. Hinzufügen eines Fremdschlüssels zur Definition von `pets`

```
ALTER TABLE pets ADD CONSTRAINT `fk_pets_2_people` FOREIGN KEY (ownerId) references people(id)
ON DELETE CASCADE;
```

Die Engine versucht möglicherweise, die Einträge von `people` vor `pets` zu löschen, wodurch der folgende Fehler verursacht wird:

```
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`))
```

Die Lösung in diesem Fall besteht darin, die Zeile von `people` zu löschen und sich auf die `ON DELETE` Funktionen von `InnoDB` verlassen, um die Löschung zu verbreiten:

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

2 Zeilen gelöscht
Paul wird aus den Leuten gestrichen
Spot wird in der Kaskade von Pets gelöscht

Eine andere Lösung ist die vorübergehende Deaktivierung der Überprüfung der Schlüssel.

```
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;
```

Grundlegendes löschen

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

Die `WHERE` Klausel ist optional, aber ohne sie werden alle Zeilen gelöscht.

LÖSCHEN vs TRUNCATE

```
TRUNCATE tableName;
```

Dadurch werden alle Daten gelöscht und der `AUTO_INCREMENT` Index zurückgesetzt. Es ist viel schneller als `DELETE FROM tableName` in einem großen Datensatz. Dies kann während der Entwicklung / beim Testen sehr nützlich sein.

Wenn Sie eine Tabelle ***abschneiden,*** löscht der SQL Server die Daten nicht, löscht die Tabelle und erstellt sie neu. Dadurch werden die Seiten freigegeben, sodass die abgeschnittenen Daten vor den überschriebenen Seiten wiederhergestellt werden können. (Der Speicherplatz kann für `innodb_file_per_table=OFF` nicht sofort wieder hergestellt werden.)

Multi-Table LÖSCHEN

In MySQL kann angegeben werden, aus welcher Tabelle die übereinstimmenden Zeilen gelöscht werden müssen

```
-- 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'
```

LÖSCHEN online lesen: https://riptutorial.com/de/mysql/topic/1487/loschen

Kapitel 32: LÖST AUS

Syntax

- CREATE [DEFINER = {Benutzer | CURRENT_USER}] TRIGGER trigger_name trigger_time trigger_event ON tbl_name Für jede Zeile [trigger_order] trigger_body
- trigger_time: {VOR VOR | NACH DEM }
- trigger_event: {INSERT | UPDATE | LÖSCHEN}
- trigger_order: {FOLLOWS | PRECEDES} other_trigger_name

Bemerkungen

Wenn Sie bei anderen DBs bereits Trigger verwenden, müssen Sie zwei Punkte beachten:

FÜR JEDE REIHE

`FOR EACH ROW` ist ein obligatorischer Teil der Syntax

Sie können keine *Anweisung* auslösen (einmal durch Abfrage) wie Oracle. Es handelt sich eher um ein leistungsbezogenes Problem als um ein wirklich fehlendes Feature

TRIGGER ERSTELLEN ODER ERSETZEN

Das `CREATE OR REPLACE` wird von MySQL nicht unterstützt

MySQL erlaubt diese Syntax nicht, Sie müssen stattdessen Folgendes verwenden:

```
DELIMITER $$

DROP TRIGGER IF EXISTS myTrigger;
$$
CREATE TRIGGER myTrigger
-- ...

$$
DELIMITER ;
```

Seien Sie vorsichtig, dies ist **keine atomare Transaktion** :

- Sie verlieren den alten Auslöser, wenn `CREATE` fehlschlägt
- `LOCK TABLES myTable WRITE;` Last können andere Operationen zwischen `DROP` und `CREATE` werden. Verwenden Sie `LOCK TABLES myTable WRITE;` zuerst, um Dateninkonsistenz und `UNLOCK TABLES;` zu vermeiden `UNLOCK TABLES;` nach dem `CREATE` , um die Tabelle freizugeben

Examples

Grundauslöser

Tabelle erstellen

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)
```

Trigger erstellen

```
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)
```

Die CREATE TRIGGER-Anweisung erstellt einen Auslöser mit dem Namen ins_sum, der der Kontentabelle zugeordnet ist. Sie enthält auch Klauseln, die die Auslöseraktionszeit, das auslösende Ereignis und die Vorgehensweise bei der Aktivierung des Auslösers angeben

Wert einfügen

Um den Trigger zu verwenden, setzen Sie die Akkumulatorvariable (@sum) auf Null, führen Sie eine INSERT-Anweisung aus und sehen Sie dann, welchen Wert die Variable danach hat:

```
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 diesem Fall ist der Wert von @sum nach der Ausführung der INSERT-Anweisung 14.98 + 1937.50 - 100 oder 1852.48.

Abzug auslösen

```
mysql> DROP TRIGGER test.ins_sum;
```

Wenn Sie eine Tabelle löschen, werden auch alle Auslöser für die Tabelle gelöscht.

Arten von Auslösern

Zeitliche Koordinierung

Es gibt zwei Modifikatoren für die Auslöseraktion:

- `BEFORE` Trigger aktiviert wird, bevor die Anforderung ausgeführt wird.
- `AFTER` Auslösen des Feuers nach der Änderung.

Auslöseereignis

Es gibt drei Ereignisse, an die Trigger angehängt werden können:

- `INSERT`
- `UPDATE`
- `DELETE`

Vor dem Trigger-Beispiel einfügen

```
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 ;
```

Vor dem Update-Trigger-Beispiel

```
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 ;
```

Nach dem Löschbeispiel

```
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 ;
```

LÖST AUS online lesen: https://riptutorial.com/de/mysql/topic/3069/lost-aus

Kapitel 33: MyISAM Engine

Bemerkungen

Im Laufe der Jahre hat sich InnoDB so weit verbessert, dass es fast immer besser ist als MyISAM, zumindest die derzeit unterstützten Versionen. Fazit: Verwenden Sie MyISAM nicht, außer vielleicht für Tabellen, die wirklich temporär sind.

Ein Vorteil von MyISAM gegenüber InnoDB ist nach wie vor: Es ist auf dem Datenträger 2x-3x kleiner.

Als InnoDB zum ersten Mal auf den Markt kam, war MyISAM noch eine brauchbare Engine. Mit XtraDB und 5.6 wurde InnoDB in den meisten Benchmarks jedoch "besser" als MyISAM.

Es wird vermutet, dass die nächste Hauptversion MyISAM überflüssig machen wird, indem wirklich temporäre InnoDB-Tabellen erstellt und die Systemtabellen in InnoDB verschoben werden.

Examples

ENGINE = MyISAM

```
CREATE TABLE foo (
     ...
) ENGINE=MyISAM;
```

MyISAM Engine online lesen: https://riptutorial.com/de/mysql/topic/4710/myisam-engine

Kapitel 34: Mysql Performance-Tipps

Examples

Wählen Sie Statement Optimization aus

Nachfolgend finden Sie einige Tipps, die Sie beim Schreiben einer Auswahlabfrage in MySQL berücksichtigen sollten, die uns helfen und unsere Abfragezeit verkürzen kann:

1. Wann immer wir wo in einer großen Tabelle verwenden, sollten wir sicherstellen, dass die Spalte in der where-Klausel index ist oder nicht. Bsp .: - Wählen Sie * aus dem Mitarbeiter, bei dem Benutzer-ID> 2000 ist. Benutzer-ID (falls indiziert) beschleunigt die Auswertung des Abfrageslot. Indizes sind auch bei Verknüpfungen und Fremdschlüsseln sehr wichtig.

2. Wenn Sie einen kleineren Inhaltsbereich benötigen, anstatt ganze Daten aus der Tabelle abzurufen, versuchen Sie, die Begrenzung zu verwenden. Schreiben Sie stattdessen Ex: - Wählen Sie * vom Mitarbeiter aus. Wenn Sie nur die ersten 20 Mitarbeiter von Lakhs benötigen, verwenden Sie einfach die Grenze Ex: - Wählen Sie * aus dem Mitarbeiter LIMIT 20 aus.

3. Sie können Ihre Abfrage auch optimieren, indem Sie den gewünschten Spaltennamen in der Ergebnismenge angeben. Schreiben Sie stattdessen Ex: - Wählen Sie * vom Mitarbeiter aus. Geben Sie einfach den Namen der Spalte an, aus der Sie Daten benötigen, wenn Ihre Tabelle viele Spalten enthält und Sie Daten für einige davon haben möchten. Beispiel: - Wählen Sie die ID und den Namen des Mitarbeiters aus.

4. Indexspalte, wenn Sie verwenden, um in where-Klausel nach NULL zu suchen. Wenn Sie eine Anweisung als SELECT * FROM Tabellenname WHERE key_col IS NULL haben; Wenn key_col indiziert wird, wird die Abfrage schneller ausgewertet.

Speicherlayout für InnoDB-Tabellen optimieren

1. In InnoDB kostet der lange PRIMARY KEY (entweder eine einzelne Spalte mit einem langen Wert oder mehrere Spalten, die einen langen zusammengesetzten Wert bilden) viel Speicherplatz. Der Primärschlüsselwert für eine Zeile wird in allen sekundären Indexdatensätzen dupliziert, die auf dieselbe Zeile zeigen. Erstellen Sie eine AUTO_INCREMENT-Spalte als Primärschlüssel, wenn Ihr Primärschlüssel lang ist.
2. Verwenden Sie den Datentyp VARCHAR anstelle von CHAR zum Speichern von Zeichenfolgen mit variabler Länge oder für Spalten mit vielen NULL-Werten. Eine CHAR (N) -Spalte benötigt immer N Zeichen, um Daten zu speichern, auch wenn die Zeichenfolge kürzer ist oder der Wert NULL ist. Kleinere Tabellen passen besser in den Pufferpool und reduzieren die Platten-E / A.

   Wenn Sie das COMPACT-Zeilenformat (das Standard-InnoDB-Format) und Zeichensätze mit variabler Länge (z. B. utf8 oder sjis) verwenden, belegen CHAR (N) -Spalten einen variablen Speicherplatz, aber immer noch mindestens N Byte.

3. Bei großen Tabellen, die viele sich wiederholende Text- oder numerische Daten enthalten, sollten Sie das COMPRESSED-Zeilenformat verwenden. Es sind weniger Festplatten-E / A erforderlich, um Daten in den Pufferpool zu laden oder vollständige Tabellenscans durchzuführen. Bevor Sie eine dauerhafte Entscheidung treffen, messen Sie den Umfang der Komprimierung, die Sie durch die Verwendung des Zeilenformats COMPRESSED gegenüber COMPACT erreichen können. *Vorbehalt:* Benchmarks zeigen selten eine bessere Komprimierung als die 2: 1-Komprimierung, und im buffer_pool für COMPRESSED gibt es viel Aufwand.
4. Wenn Ihre Daten eine stabile Größe erreicht haben oder eine wachsende Tabelle um einige Dutzend oder einige Hundert Megabyte gestiegen ist, sollten Sie die Anweisung OPTIMIZE TABLE verwenden, um die Tabelle neu zu ordnen und überflüssigen Speicherplatz zu komprimieren. Die reorganisierten Tabellen erfordern weniger Platten-E / A, um vollständige Tabellenscans durchzuführen. Dies ist eine unkomplizierte Technik, die die Leistung verbessern kann, wenn andere Techniken, wie beispielsweise die Verbesserung der Indexnutzung oder das Optimieren von Anwendungscode, nicht praktikabel sind. *Vorbehalt* : Unabhängig von der Tabellengröße sollte OPTIMIZE TABLE nur selten ausgeführt werden. Dies liegt daran, dass es teuer ist und den Tisch selten genug verbessert, um es wert zu sein. InnoDB ist ziemlich gut darin, seine B + Trees von viel verschwendetem Speicherplatz freizuhalten.

   OPTIMIZE TABLE kopiert den Datenteil der Tabelle und erstellt die Indizes neu. Die Vorteile ergeben sich aus dem verbesserten Packen von Daten in Indizes und einer geringeren Fragmentierung in den Tablespaces und auf der Festplatte. Die Vorteile variieren je nach Daten in jeder Tabelle. Möglicherweise gibt es für einige einige und nicht für andere signifikante Gewinne, oder die Gewinne nehmen mit der Zeit ab, bis Sie die Tabelle das nächste Mal optimieren. Dieser Vorgang kann langsam sein, wenn die Tabelle groß ist oder die neu erstellten Indizes nicht in den Pufferpool passen. Der erste Durchlauf, nachdem einer Tabelle viele Daten hinzugefügt wurden, ist häufig viel langsamer als spätere Durchläufe.

Erstellen eines zusammengesetzten Index

In vielen Situationen ist ein zusammengesetzter Index besser als ein Index mit einer einzelnen Spalte. Um einen optimalen zusammengesetzten Index zu erstellen, füllen Sie ihn mit Spalten in dieser Reihenfolge.

- `=` Spalte (n) zuerst aus der `WHERE` Klausel. (zB `INDEX(a,b,...)` für `WHERE a=12 AND b='xyz' ...`)
- `IN` Spalte (n); Der Optimierer kann möglicherweise den Index überspringen.
- Ein "Bereich" (zB `x BETWEEN 3 AND 9` , `name LIKE 'J%'`) Es wird nichts weiter als die erste Bereichsspalte verwendet.
- Alle Spalten in `GROUP BY`
- Alle Spalten in der Reihenfolge `ORDER BY` . Funktioniert nur, wenn alle `ASC` oder alle `DESC` oder Sie 8.0 verwenden.

Anmerkungen und Ausnahmen:

- Dupliziere keine Spalten.

- Überspringen Sie alle Fälle, die nicht zutreffen.
- Wenn Sie nicht alle Spalten von `WHERE` , müssen Sie nicht zu `GROUP BY` usw. weitergehen.
- Es gibt Fälle, in denen es nützlich ist, nur die `ORDER BY` Spalten zu indizieren, wobei `WHERE` ignoriert wird.
- "Hide" eine Spalte in einer Funktion nicht (zB `DATE(x) = ...` kann `x` im Index nicht verwenden.)
- Die "Präfix" `text_col(99)` (z. B. `text_col(99)`) ist unwahrscheinlich hilfreich. kann weh tun

Weitere Details und Tipps .

Mysql Performance-Tipps online lesen: https://riptutorial.com/de/mysql/topic/5752/mysql-performance-tipps

Kapitel 35: MySQL-Admin

Examples

Ändern Sie das root-Passwort

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

Datenbank löschen

Nützlich für das Scripting, um alle Tabellen zu löschen und die Datenbank zu löschen:

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

Mit äußerster Vorsicht verwenden.

`DROP` Datenbank als SQL-Skript (Sie benötigen DROP-Berechtigungen für diese Datenbank):

```
DROP DATABASE database_name
```

oder

```
DROP SCHEMA database_name
```

Atomic RENAME & Table Reload

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

Während der Ausführung von RENAME TABLE können keine anderen Sitzungen auf die beteiligten Tabellen zugreifen, sodass die Umbenennungsoperation keinen Parallelitätsproblemen unterliegt.

Atomic Rename ist speziell für das vollständige Neuladen einer Tabelle, ohne auf `DELETE` zu warten und das Laden abzuschließen:

```
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;
```

MySQL-Admin online lesen: https://riptutorial.com/de/mysql/topic/2991/mysql-admin

Kapitel 36: MySQL-Client

Syntax

- mysql [OPTIONEN] [Datenbankname]

Parameter

Parameter	Beschreibung
`-D --database=name`	Name der Datenbank
`--delimiter=str`	Setzen Sie das Trennzeichen für die Anweisung. Die Standardeinstellung ist ';'
`-e --execute='command'`	Befehl ausführen
`-h --host=name`	Hostname, zu dem eine Verbindung hergestellt werden soll
`-p --password=name`	Kennwort *Anmerkung: Zwischen `-p` und dem Kennwort ist kein Leerzeichen*
`-p` (ohne Passwort)	Das Passwort wird abgefragt
`-P --port=#`	Port-Nummer
`-s --silent`	Silent-Modus, produzieren weniger Leistung. Verwenden Sie `\t` als Spaltentrennzeichen
`-ss`	wie `-s` , aber Spaltennamen weglassen
`-S --socket=path`	Geben Sie den Socket (Unix) oder den Named Pipe (Windows) an, der beim Herstellen einer Verbindung mit einer lokalen Instanz verwendet werden soll
`--skip-column-names`	Spaltennamen weglassen
`-u --user=name`	Nutzername
`-U --safe-updates --i-am-a-dummy`	`sql_safe_updates=ON` mit der Variablen `sql_safe_updates=ON` . Dies erlaubt nur `DELETE` und `UPDATE` , die Schlüssel explizit verwenden
`-V --version`	Version ausdrucken und beenden

Examples

Basis-Login

So greifen Sie von der Befehlszeile aus auf MySQL zu:

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

Dies kann verkürzt werden auf:

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

Wenn Sie den `password` weglassen, fragt MySQL bei der ersten Eingabe nach dem erforderlichen Kennwort. Wenn Sie ein `password` angeben, gibt der Client eine Warnung "unsicher" aus:

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

Für lokale Verbindungen kann `--socket` verwendet werden, um auf die Socket-Datei zu zeigen:

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

Wird der `socket` Parameter nicht angegeben, versucht der Client, eine Verbindung zu einem Server auf dem lokalen Computer herzustellen. Der Server muss laufen, um eine Verbindung herstellen zu können.

Befehle ausführen

Dieses Beispiel zeigt, wie Befehle ausgeführt werden, die in Zeichenfolgen oder Skriptdateien gespeichert sind, ohne dass eine interaktive Aufforderung erforderlich ist. Dies ist besonders nützlich, wenn ein Shell-Skript mit einer Datenbank interagieren muss.

Befehl aus einer Zeichenfolge ausführen

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

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

Um die Ausgabe als `--silent` Raster zu formatieren, verwenden Sie den Parameter `--silent` :

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

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

Die Kopfzeilen weglassen:

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

1       Kathy   f
2       John    m
```

Ausführen aus Skriptdatei:

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

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

Schreibe die Ausgabe in eine Datei

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

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

MySQL-Client online lesen: https://riptutorial.com/de/mysql/topic/5619/mysql-client

Kapitel 37: MySQL-Gewerkschaften

Syntax

- SELECT Spaltenname (n) FROM table1 UNION SELECT Spaltenname (s) FROM table2;
- SELECT Spaltenname (n) FROM table1 UNION ALL SELECT Spaltenname (s) FROM table2;
- SELECT Spaltenname (n) FROM table1 WHERE Spaltenname = "XYZ" UNION ALL SELECT Spaltenname (s) FROM Tabelle2 WHERE Spaltenname = "XYZ";

Bemerkungen

`UNION DISTINCT` ist das gleiche wie `UNION` ; es ist langsamer als `UNION ALL` wegen eines Duplizierungsdurchlaufs. Es `DISTINCT` , `DISTINCT` oder `ALL` immer zu buchstabieren, um zu signalisieren, dass Sie darüber nachgedacht haben, was Sie tun sollen.

Examples

Unionsbetreiber

Der UNION-Operator wird verwendet, um die Ergebnismenge (*nur unterschiedliche Werte*) von zwei oder mehr SELECT-Anweisungen zu kombinieren.

Abfrage: (Zum Auswählen aller verschiedenen Städte (*nur unterschiedliche Werte*) aus den Tabellen "Kunden" und "Lieferanten")

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Ergebnis:

```
Number of Records: 10

City
------
Aachen
Albuquerque
Anchorage
Annecy
Barcelona
Barquisimeto
Bend
Bergamo
Berlin
Bern
```

Union ALL

UNION ALL, um alle (doppelten Werte) Städte aus den Tabellen "Kunden" und "Lieferanten" auszuwählen.

Abfrage:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

Ergebnis:

```
Number of Records: 12

City
-------
Aachen
Albuquerque
Anchorage
Ann Arbor
Annecy
Barcelona
Barquisimeto
Bend
Bergamo
Berlin
Berlin
Bern
```

UNION ALL mit WO

UNION ALL, um alle (auch doppelte Werte) deutsche Städte aus den Tabellen "Kunden" und "Lieferanten" auszuwählen. Hier ist `Country="Germany"` in der where-Klausel anzugeben.

Abfrage:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

Ergebnis:

```
Number of Records: 14
```

Stadt	Land
Aachen	Deutschland

Berlin	Deutschland
Berlin	Deutschland
Brandenburg	Deutschland
Cunewalde	Deutschland
Cuxhaven	Deutschland
Frankfurt	Deutschland
Frankfurt aM	Deutschland
Köln	Deutschland
Leipzig	Deutschland
Mannheim	Deutschland
München	Deutschland
Münster	Deutschland
Stuttgart	Deutschland

MySQL-Gewerkschaften online lesen: https://riptutorial.com/de/mysql/topic/5376/mysql-gewerkschaften

Kapitel 38: mysqlimport

Parameter

Parameter	Beschreibung
`--delete -D`	leeren Sie die Tabelle, bevor Sie die Textdatei importieren
`--fields-optionally-enclosed-by`	Definiere das Zeichen, das die Felder zitiert
`--fields-terminated-by`	Feldabschlusszeichen
`--ignore -i`	Ignorieren Sie die aufgenommene Zeile bei Duplikaten
`--lines-terminated-by`	Zeilenabschluss definieren
`--password -p`	Passwort
`--port -P`	Hafen
`--replace -r`	Überschreiben Sie die alte Eingabezeile bei Duplikatschlüsseln
`--user -u`	Nutzername
`--where -w`	Geben Sie eine Bedingung an

Bemerkungen

`mysqlimport` verwendet den Namen der importierten Datei nach dem Entfernen der Erweiterung, um die Zieltabelle zu bestimmen.

Examples

Grundlegende Verwendung

Angenommen, die durch Tabulatoren getrennte Datei `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
```

Verwenden eines benutzerdefinierten Feldbegrenzers

Angenommen, die Textdatei employee.txt

1 | Arthur Dent
2 | Marvin
3 | Zaphod Beeblebrox

```
$ mysqlimport --fields-terminated-by='|' mycompany employee.txt
```

Verwenden eines benutzerdefinierten Zeilenbegrenzers

Dieses Beispiel ist nützlich für Windows-ähnliche Endungen:

```
$ mysqlimport --lines-terminated-by='\r\n' mycompany employee.txt
```

Umgang mit doppelten Schlüsseln

Angesichts der Tabelle `Employee`

Ich würde	Name
3	Yooden Vranx

Und die Datei `employee.txt`

1 `\t` Arthur Dent
2 `\t` Marvin
3 `\t` Zaphod Beeblebrox

Die Option `--ignore` ignoriert die Eingabe für doppelte Schlüssel

```
$ mysqlimport --ignore mycompany employee.txt
```

Ich würde	Name
1	Arthur Dent
2	Marvin
3	Yooden Vranx

Die Option `--replace` überschreibt den alten Eintrag

```
$ mysqlimport --replace mycompany employee.txt
```

Ich würde	Name
1	Arthur Dent
2	Marvin
3	Zaphod Beeblebrox

Bedingter Import

```
$ mysqlimport --where="id>2" mycompany employee.txt
```

Importieren Sie eine Standard-CSV

```
$ mysqlimport
    --fields-optionally-enclosed-by='"'
    --fields-terminated-by=,
    --lines-terminated-by="\r\n"
    mycompany employee.csv
```

mysqlimport online lesen: https://riptutorial.com/de/mysql/topic/5215/mysqlimport

Kapitel 39: MySQL-Sperrtabelle

Syntax

- LOCK TABLES Tabellenname [READ | SCHREIBEN]; // Tabelle sperren
- ENTSPERREN SIE TABELLEN; // Tabellen entsperren

Bemerkungen

Sperren wird verwendet, um Parallelitätsprobleme zu lösen. Das Sperren ist nur erforderlich, wenn eine Transaktion ausgeführt wird, die zuerst einen Wert aus einer Datenbank liest und diesen Wert später in die Datenbank schreibt. Sperren sind niemals für in sich geschlossene Einfügungs-, Aktualisierungs- oder Löschvorgänge erforderlich.

Es gibt zwei Arten von Schlössern

READ LOCK - wenn ein Benutzer nur aus einer Tabelle liest.

WRITE LOCK - wenn ein Benutzer eine Tabelle liest und schreibt.

Wenn ein Benutzer eine `WRITE LOCK` für eine Tabelle hält, kann kein anderer Benutzer diese Tabelle lesen oder schreiben. Wenn ein Benutzer eine `READ LOCK` für eine Tabelle hält, können andere Benutzer auch eine `READ LOCK` lesen oder halten, aber kein Benutzer kann eine `WRITE LOCK` für diese Tabelle schreiben oder halten.

Wenn die Standard-Speicher-Engine InnoDB ist, verwendet MySQL automatisch das Sperren auf Zeilenebene, sodass mehrere Transaktionen dieselbe Tabelle gleichzeitig für Lesen und Schreiben verwenden können, ohne sich gegenseitig warten zu müssen.

Für alle Speicher-Engines außer InnoDB verwendet MySQL die Tabellensperre.

Weitere Informationen zur Tabellensperre finden Sie hier

Examples

Mysql Locks

Tabellensperren können ein wichtiges Werkzeug für `ENGINE=MyISAM`, sind jedoch für `ENGINE=InnoDB` selten nützlich. Wenn Sie versucht sind, Tabellensperren mit InnoDB zu verwenden, sollten Sie überdenken, wie Sie mit Transaktionen arbeiten.

MySQL ermöglicht Client-Sitzungen das explizite Anfordern von Tabellensperren, um mit anderen Sitzungen für den Zugriff auf Tabellen zusammenzuarbeiten, oder um zu verhindern, dass andere Sitzungen Tabellen in Zeiten ändern, in denen eine Sitzung exklusiven Zugriff auf sie erfordert. Eine Sitzung kann Sperren nur für sich selbst erwerben oder freigeben. Eine Sitzung kann keine

Sperren für eine andere Sitzung abrufen oder Sperren aufheben, die von einer anderen Sitzung gehalten werden.

Sperren können verwendet werden, um Transaktionen zu emulieren oder die Aktualisierung von Tabellen zu beschleunigen. Dies wird später in diesem Abschnitt näher erläutert.

Befehl: `LOCK TABLES table_name READ|WRITE;`

Sie können einer einzelnen Tabelle nur den Sperrtyp zuweisen.

Beispiel (Lesesperre):

```
LOCK TABLES table_name READ;
```

Beispiel (WRITE LOCK):

```
LOCK TABLES table_name WRITE;
```

Um zu sehen, ob die Sperre angewendet wird oder nicht, verwenden Sie den folgenden Befehl

```
SHOW OPEN TABLES;
```

Verwenden Sie den folgenden Befehl, um alle Sperren zu löschen oder zu entfernen:

```
UNLOCK TABLES;
```

BEISPIEL:

```
LOCK TABLES products WRITE:
INSERT INTO products(id,product_name) SELECT id,old_product_name FROM old_products;
UNLOCK TABLES;
```

Im obigen Beispiel kann eine externe Verbindung keine Daten in die Produkttabelle schreiben, bis das Tabellenprodukt entsperrt wird

BEISPIEL:

```
LOCK TABLES products READ:
INSERT INTO products(id,product_name) SELECT id,old_product_name FROM old_products;
UNLOCK TABLES;
```

Im obigen Beispiel kann eine externe Verbindung keine Daten aus der Produkttabelle lesen, bis das Tabellenprodukt entsperrt wird

Zeilenebenensperre

Wenn die Tabellen InnoDB verwenden, verwendet MySQL automatisch das Sperren auf Zeilenebene, so dass mehrere Transaktionen dieselbe Tabelle gleichzeitig für Lesen und Schreiben verwenden können, ohne sich gegenseitig warten zu müssen.

Wenn zwei Transaktionen, die versuchen, dieselbe Zeile zu ändern, und beide eine Sperrung auf Zeilenebene verwenden, wartet eine der Transaktionen, bis die andere Transaktion abgeschlossen ist.

Das Sperren auf Zeilenebene kann auch mit der Anweisung `SELECT ... FOR UPDATE` für jede zu `SELECT ... FOR UPDATE` Zeile erhalten werden.

Betrachten Sie zwei Verbindungen, um das Sperren auf Zeilenebene im Detail zu erklären

Verbindung 1

```
START TRANSACTION;
SELECT ledgerAmount FROM accDetails WHERE id = 1 FOR UPDATE;
```

In Verbindung 1 wird die `SELECT ... FOR UPDATE` Anweisung `SELECT ... FOR UPDATE` .

Verbindung 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1;
```

Wenn `innodb_lock_wait_timeout` versucht, dieselbe Zeile in Verbindung 2 zu aktualisieren, wird darauf gewartet, dass Verbindung 1 die Transaktion beendet oder eine Fehlermeldung wird gemäß der Einstellung `innodb_lock_wait_timeout` (50 Sekunden) angezeigt.

```
Error Code: 1205. Lock wait timeout exceeded; try restarting transaction
```

Um Details zu dieser Sperre `SHOW ENGINE INNODB STATUS` , führen Sie `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:
```

Verbindung 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 250 WHERE id=2;
1 row(s) affected
```

Während der Aktualisierung wird jedoch eine andere Zeile in Verbindung 2 fehlerfrei ausgeführt.

Verbindung 1

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 750 WHERE id=1;
COMMIT;
1 row(s) affected
```

Jetzt wird die Zeilensperre freigegeben, da die Transaktion in Verbindung 1 festgeschrieben wird.

Verbindung 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1;
1 row(s) affected
```

Das Update wird ohne Fehler in Verbindung 2 ausgeführt, nachdem die Zeilensperre von Verbindung 1 durch Beenden der Transaktion freigegeben wurde.

MySQL-Sperrtabelle online lesen: https://riptutorial.com/de/mysql/topic/5233/mysql-sperrtabelle

Kapitel 40: Neuen Benutzer erstellen

Bemerkungen

Um eine Liste der MySQL-Benutzer anzuzeigen, verwenden Sie den folgenden Befehl:

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

Examples

Erstellen Sie einen MySQL-Benutzer

Um einen neuen Benutzer zu erstellen, müssen wir die folgenden einfachen Schritte ausführen:

Schritt 1: Melden Sie sich als `root` bei `MySQL` an

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

Schritt 2: Es erscheint die Eingabeaufforderung von mysql

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

Hier haben wir erfolgreich einen neuen Benutzer erstellt. Dieser Benutzer hat jedoch keine `permissions` . Um dem Benutzer `permissions` zuzuweisen, verwenden Sie folgenden Befehl:

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

Geben Sie das Passwort an

Die grundlegende Verwendung ist:

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

In Situationen, in denen es nicht ratsam ist, das Kennwort im Klartext zu `PASSWORD` , ist es auch möglich, den `PASSWORD` , wie er von der Funktion `PASSWORD()` zurückgegeben wird, direkt mit der Direktive `PASSWORD` anzugeben:

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

Erstellen Sie einen neuen Benutzer und erteilen Sie dem Schema alle Berechtigungen

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

Achtung: Hiermit können Sie einen neuen Root-Benutzer erstellen

Benutzer umbenennen

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

Wenn Sie versehentlich einen Benutzer erstellen, können Sie seinen Namen ändern

Neuen Benutzer erstellen online lesen: https://riptutorial.com/de/mysql/topic/3508/neuen-benutzer-erstellen

Kapitel 41: NULL

Examples

Verwendet für NULL

- Daten noch nicht bekannt - wie `end_date` , `rating`
- Optionale Daten - wie `middle_initial` (obwohl dies möglicherweise besser ist als die leere Zeichenfolge)
- 0/0 - Das Ergebnis bestimmter Berechnungen, z. B. Null geteilt durch Null.
- NULL ist nicht gleich "" (leerer String) oder 0 (bei Ganzzahl).
- Andere?

NULLs testen

- `IS NULL` / `IS NOT NULL` - `= NULL` funktioniert nicht wie erwartet.
- `x <=> y` ist ein "nullsicherer" Vergleich.

In einem `LEFT JOIN` Test werden Zeilen von `a LEFT JOIN` , für die es in `b` *keine* entsprechende Zeile gibt.

```
SELECT ...
    FROM a
    LEFT JOIN b ON ...
    WHERE b.id IS NULL
```

NULL online lesen: https://riptutorial.com/de/mysql/topic/6757/null

Kapitel 42: Partitionierung

Bemerkungen

- **RANGE partitioning** . Diese Art der Partitionierung weist den Partitionen Zeilen zu, basierend auf Spaltenwerten, die in einen bestimmten Bereich fallen.
- **LIST Partitionierung** . Ähnlich der Partitionierung nach RANGE, nur dass die Partition basierend auf Spalten ausgewählt wird, die mit einem Satz diskreter Werte übereinstimmen.
- **HASH-Partitionierung** Bei dieser Art der Partitionierung wird eine Partition basierend auf dem Wert ausgewählt, der von einem benutzerdefinierten Ausdruck zurückgegeben wird, der Spaltenwerte in Zeilen bearbeitet, die in die Tabelle eingefügt werden sollen. Die Funktion kann aus einem beliebigen in MySQL gültigen Ausdruck bestehen, der einen nicht negativen Ganzzahlwert ergibt. Eine Erweiterung dieses Typs, `LINEAR HASH` , ist ebenfalls verfügbar.
- **KEY Partitionierung** . Diese Art der Partitionierung ähnelt der Partitionierung durch HASH, mit der Ausnahme, dass nur eine oder mehrere auszuwertende Spalten bereitgestellt werden und der MySQL-Server seine eigene Hash-Funktion bereitstellt. Diese Spalten können andere Werte als ganzzahlige Werte enthalten, da die von MySQL bereitgestellte Hash-Funktion unabhängig vom Spaltendatentyp ein ganzzahliges Ergebnis garantiert. Eine Erweiterung dieses Typs, `LINEAR KEY` , ist ebenfalls verfügbar.

Examples

RANGE Partitionierung

Eine nach Bereichen partitionierte Tabelle wird so partitioniert, dass jede Partition Zeilen enthält, für die der Wert des Partitionierungsausdrucks innerhalb eines bestimmten Bereichs liegt. Die Bereiche sollten zusammenhängend sein, sich jedoch nicht überlappen, und werden mit dem Operator `VALUES LESS THAN` definiert. Angenommen, Sie erstellen eine Tabelle wie die folgende, um Personaldatensätze für eine Kette von 20 Videospeichern mit den Nummern 1 bis 20 zu speichern:

```
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
);
```

Diese Tabelle kann je nach Bedarf auf verschiedene Arten nach Bereichen partitioniert werden. Eine Möglichkeit wäre die Verwendung der `store_id` Spalte. Beispielsweise können Sie sich entscheiden, die Tabelle auf vier Arten zu partitionieren, indem Sie eine `PARTITION BY RANGE` Klausel

wie hier gezeigt hinzufügen:

```
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` für einen ganzzahligen Wert, der immer größer als der größtmögliche ganzzahlige Wert ist (in der mathematischen Sprache dient er als kleinste obere Grenze).

basierend auf dem offiziellen MySQL-Dokument .

LISTE Partitionierung

Die Listenpartitionierung ähnelt in vielerlei Hinsicht der Bereichspartitionierung. Wie bei der Partitionierung nach RANGE muss jede Partition explizit definiert werden. Der Hauptunterschied zwischen den beiden Partitionierungsarten besteht darin, dass bei der Listenpartitionierung jede Partition basierend auf der Zugehörigkeit eines Spaltenwerts in einer Wertelistengruppe definiert und ausgewählt wird und nicht in einem Satz zusammenhängender Bereiche von Werte. Dazu verwenden Sie `PARTITION BY LIST(expr)` wobei `expr` ein Spaltenwert oder ein Ausdruck ist, der auf einem Spaltenwert basiert und einen ganzzahligen Wert `VALUES IN (value_list)` Anschließend wird jede Partition mit Hilfe von `VALUES IN (value_list)` , wobei `value_list` ist kommagetrennte Liste von Ganzzahlen.

Für die folgenden Beispiele gehen wir davon aus, dass die grundlegende Definition der zu partitionierenden Tabelle von der hier gezeigten `CREATE TABLE` Anweisung bereitgestellt wird:

```
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
);
```

Angenommen, es gibt 20 Videotheken, die auf vier Franchises verteilt sind, wie in der folgenden Tabelle gezeigt.

Region	ID-Nummern speichern
Norden	3, 5, 6, 9, 17
Osten	1, 2, 10, 11, 19, 20
Westen	4, 12, 13, 14, 18

Region	ID-Nummern speichern
Zentral	7, 8, 15, 16

Um diese Tabelle so zu partitionieren, dass Zeilen für Geschäfte, die zu derselben Region gehören, in derselben Partition gespeichert werden

```
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)
);
```

basierend auf dem offiziellen MySQL-Dokument .

HASH-Partitionierung

Die Partitionierung durch HASH dient in erster Linie dazu, eine gleichmäßige Verteilung der Daten auf eine vorbestimmte Anzahl von Partitionen sicherzustellen. Bei der Bereichs- oder Listenpartitionierung müssen Sie explizit angeben, in welcher Partition ein bestimmter Spaltenwert oder ein Satz von Spaltenwerten gespeichert werden soll. Bei der Hash-Partitionierung übernimmt MySQL dies für Sie. Sie müssen nur einen Spaltenwert oder -ausdruck angeben, der auf einem zu hashenden Spaltenwert und der Anzahl der Partitionen basiert, in die die partitionierte Tabelle aufgeteilt werden soll.

Die folgende Anweisung erstellt eine Tabelle, die Hashing für die store_id-Spalte verwendet und in 4 Partitionen unterteilt ist:

```
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;
```

> Wenn Sie keine `PARTITIONS` Klausel `PARTITIONS` , beträgt die Anzahl der Partitionen standardmäßig 1.

basierend auf dem offiziellen MySQL-Dokument .

Partitionierung online lesen: https://riptutorial.com/de/mysql/topic/5128/partitionierung

Kapitel 43: Passen Sie PS1 an

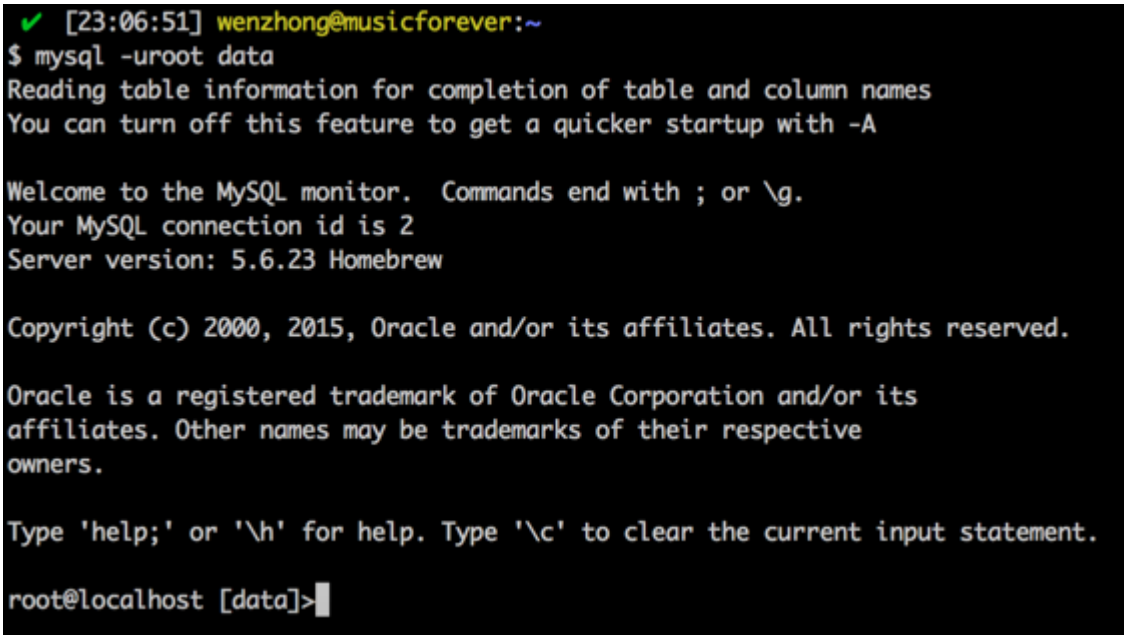
Examples

Passen Sie das MySQL PS1 an die aktuelle Datenbank an

Fügen Sie in der .bashrc oder .bash_profile Folgendes hinzu:

```
export MYSQL_PS1="\u@\h [\d]>"
```

MySQL-Client PROMPT den aktuellen Benutzer @ Host [Datenbank] anzeigen lassen.



Benutzerdefiniertes PS1 über MySQL-Konfigurationsdatei

In `mysqld.cnf` oder gleichwertig:

```
[mysql]
prompt = '\u@\h [\d]> '
```

Dies erzielt einen ähnlichen Effekt, ohne sich mit den `.bashrc` .

Passen Sie PS1 an online lesen: https://riptutorial.com/de/mysql/topic/5795/passen-sie-ps1-an

Kapitel 44: Pivot-Abfragen

Bemerkungen

Die Erstellung von Pivot-Abfragen in MySQL `GROUP_CONCAT()` auf der `GROUP_CONCAT()` Funktion. Wenn das Ergebnis des Ausdrucks, der die Spalten der Pivot-Abfrage erstellt, voraussichtlich groß ist, muss der Wert der Variablen `group_concat_max_len` erhöht werden:

```
set session group_concat_max_len = 1024 * 1024; -- This should be enough for most cases
```

Examples

Erstellen einer Pivot-Abfrage

MySQL bietet keine integrierte Möglichkeit zum Erstellen von Pivot-Abfragen. Diese können jedoch mit vorbereiteten Anweisungen erstellt werden.

Angenommen, die Tabelle `tbl_values` :

Ich würde	Name	Gruppe	Wert
1	Pete	EIN	10
2	Pete	B	20
3	John	EIN	10

Anfrage: Erstellen Sie eine Abfrage, die die Summe des `Value` für jeden `Name` anzeigt. Die `Group` muss eine Spaltenüberschrift sein und `Name` muss die Zeilenüberschrift sein.

```
-- 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;
```

Ergebnis:

Name	EIN	B
John	10	NULL
Pete	10	20

Wichtig: Deaktivieren Sie die vorbereitete Anweisung, sobald sie nicht mehr benötigt wird:

```
deallocate prepare stmt;
```

Beispiel zur SQL-Geige

Pivot-Abfragen online lesen: https://riptutorial.com/de/mysql/topic/3074/pivot-abfragen

Kapitel 45: PREPARE-Anweisungen

Syntax

- PREPARE stmt_name FROM preparable_stmt
- EXECUTE stmt_name [USING @var_name [, @var_name] ...]
- {ENALLOCATE | DROP} PREPARE stmt_name

Examples

PREPARE-Anweisungen, EXECUTE- und DEALLOCATE-Anweisungen

PREPARE bereitet eine Anweisung zur Ausführung vor

EXECUTE führt eine vorbereitete Anweisung aus

DEALLOCATE PREPARE gibt eine vorbereitete Anweisung frei

```
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;
```

Ergebnis:

```
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
```

Endlich,

```
DEALLOCATE PREPARE stmt2;
```

Anmerkungen:

- Sie müssen @ -Variablen verwenden, nicht DECLAREd-Variablen für `FROM @s`
- Eine Hauptanwendung für Prepare usw. ist das Erstellen einer Abfrage für Situationen, in denen die Bindung nicht funktioniert, z.

Konstruieren und ausführen

(Dies ist eine Anfrage für ein gutes Beispiel , das zeigt , wie ein *konstruieren* `SELECT` mit `CONCAT` , dann bereiten + ausführen Bitte betonen die Verwendung von @variables gegen deklarierten Variablen -. Es macht einen großen Unterschied, und es ist etwas , die Anfänger (mich selbst)

stolpern vorbei.)

Ändern Sie die Tabelle mit Spalte hinzufügen

```
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;
```

PREPARE-Anweisungen online lesen: https://riptutorial.com/de/mysql/topic/2603/prepare-anweisungen

Kapitel 46: Protokolldateien

Examples

Eine Liste

- Allgemeines Protokoll - alle Abfragen - siehe VARIABLE general_log
- Langsame Log-Abfragen langsamer als long_query_time - slow_query_log_file
- Binlog - für die Replikation und Sicherung - log_bin_basename
- Relay Log - auch für die Replikation
- Allgemeine Fehler - mysqld.err
- start / stop - mysql.log (nicht sehr interessant) - log_error
- InnoDB-Wiederherstellungsprotokoll - iblog *

In den Variablen `basedir` und `datadir` Standardspeicherort für viele Protokolle

Einige Protokolle werden von anderen VARIABLES ein- und ausgeschaltet. Einige werden entweder in eine Datei oder in eine Tabelle geschrieben.

(Hinweis für Rezensenten: Dies erfordert mehr Details und weitere Erläuterungen.)

Dokumentierer : Bitte geben Sie den Standardspeicherort und -namen für jeden Protokolltyp sowohl für Windows als auch für * nix an. (Oder zumindest so viel wie möglich.)

Langsames Abfrageprotokoll

Das langsame `long_query_time` besteht aus Protokollereignissen für Abfragen, deren Abschluss bis zu `long_query_time` dauert. Zum Beispiel bis zu 10 Sekunden. Geben Sie Folgendes ein, um den aktuell eingestellten Zeitgrenzwert anzuzeigen:

```
SELECT @@long_query_time;
+--------------------+
| @@long_query_time |
+--------------------+
|          10.000000 |
+--------------------+
```

Sie kann als GLOBAL-Variable in der Datei `my.cnf` oder `my.ini` werden. Oder es kann von der Verbindung eingestellt werden, obwohl dies ungewöhnlich ist. Der Wert kann zwischen 0 und 10 Sekunden eingestellt werden. Welchen Wert verwenden?

- 10 ist so hoch, dass es fast nutzlos ist;
- 2 ist ein Kompromiss;
- 0,5 und andere Fraktionen sind möglich;
- 0 erfasst alles; Dies könnte die Festplatte gefährlich schnell füllen, kann jedoch sehr nützlich sein.

Die Erfassung langsamer Abfragen ist entweder aktiviert oder deaktiviert. Die angemeldete Datei wird ebenfalls angegeben. Das Folgende fängt diese Konzepte ein:

```
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)
```

Weitere Informationen finden Sie auf der MySQL-Handbuchseite The Slow Query Log

Hinweis: Die obigen Informationen zum Ein- / Ausschalten des Slowlogs wurden in 5.6 (?) Geändert. ältere Version hatte einen anderen Mechanismus.

Der "beste" Weg, um zu sehen, was Ihr System verlangsamt:

```
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
```

Allgemeines Abfrageprotokoll

Das allgemeine Abfrageprotokoll enthält eine Liste allgemeiner Informationen zu Clientverbindungen, Verbindungsabbrüchen und Abfragen. Es ist von unschätzbarem Wert für das Debugging, stellt jedoch ein Leistungshindernis dar (Zitat?).

Eine Beispielansicht eines allgemeinen Abfrageprotokolls ist unten zu sehen:

```
36 Query insert questions_c23(qId,ownerId,title,votes,answers,isClosed,closeVotes,views,owne
        comments,answeredAccepted,askDate,closeDate,lastScanDate,ign,bn,pvtc,
        mainTagForImport,prepStatus,touches,status,status_bef_change,cv_bef_change,max_cv_r
        values(38666373, 1322183, 'How to post a numeric value in c#', 0, 1, 0, 0, 50, 1,
        0, 0, '2016-07-29 19:40:32', null, now(), 0, 0, 0,
        'c%23',0,1,'O','',0,0)
        on duplicate key update title='How to post a numeric value in c#', votes=0, answers
        answeredAccepted=0,lastScanDate=now(), touches=touches+1,status='O'
```

So stellen Sie fest, ob das allgemeine Protokoll derzeit erfasst wird:

```
SELECT @@general_log; -- 1 = Capture is active; 0 = It is not.
```

So bestimmen Sie den Dateinamen der Capture-Datei:

```
SELECT @@general_log_file; -- Full path to capture file
```

Wenn der vollständige Pfad zur Datei nicht angezeigt wird, ist die Datei im `datadir` .

Windows-Beispiel:

```
+-----------------------------------------------------------+
| @@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 |
+----------------------------------+
```

Wenn Änderungen an der gemacht werden `general_log_file` GLOBAL Variable, wird das neue Protokoll in der gespeicherten `datadir` . Der vollständige Pfad wird jedoch möglicherweise durch die Untersuchung der Variablen nicht mehr angezeigt.

`general_log_file` in der Konfigurationsdatei kein Eintrag für `general_log_file` enthalten ist, wird standardmäßig `@@hostname` .log im `datadir` .

Best Practices sind das Abschalten der Erfassung. Speichern Sie die Protokolldatei in einem Sicherungsverzeichnis mit einem Dateinamen, der die Start- / Endzeit der Erfassung angibt. Löschen der vorherigen Datei, wenn diese Datei nicht *verschoben wurde* . Legen Sie einen neuen Dateinamen für die Protokolldatei fest und schalten Sie die Erfassung ein (alle werden unten angezeigt). Best Practices beinhalten auch eine sorgfältige Bestimmung, wenn Sie im Moment sogar erfassen möchten. In der Regel ist die Erfassung nur zu Debugging-Zwecken aktiviert.

Ein typischer Dateiname des Dateisystems für ein gesichertes Protokoll ist:

```
/LogBackup/GeneralLog_20160802_1520_to_20160802_1815.log
```

Dabei sind Datum und Uhrzeit Teil des Dateinamens als Bereich.

Für Windows beachten Sie die folgende Reihenfolge mit den Einstellungsänderungen.

```
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 ist ähnlich. Diese würden dynamische Veränderungen darstellen. Bei einem Neustart des Servers werden die Einstellungen der Konfigurationsdatei übernommen.

Berücksichtigen Sie bei der Konfigurationsdatei die folgenden relevanten Variableneinstellungen:

```
[mysqld]
```

```
general_log_file = /path/to/currentquery.log
general_log      = 1
```

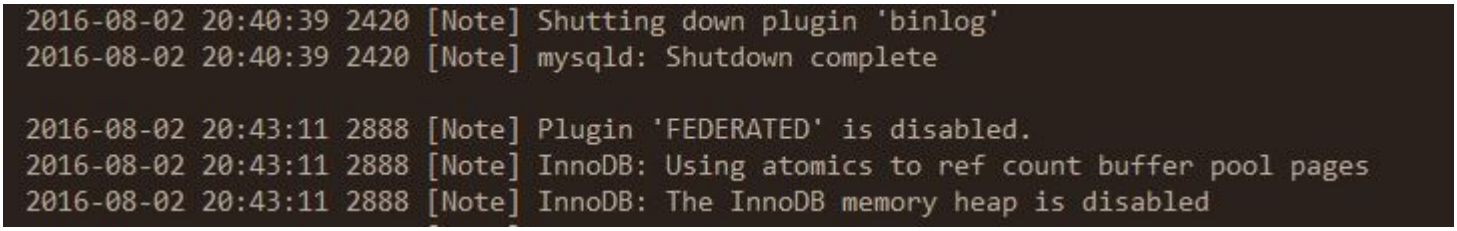
Außerdem kann die Variable `log_output` für die Ausgabe von `TABLE` konfiguriert werden, nicht nur für `FILE` . Siehe dazu Destinationen .

Siehe die MySQL-Handbuchseite Das allgemeine Abfrageprotokoll .

Fehlerprotokoll

Das Fehlerprotokoll enthält Start- und Stoppinformationen sowie kritische Ereignisse, die vom Server festgestellt wurden.

Das folgende ist ein Beispiel für den Inhalt:



Die Variable `log_error` enthält den Pfad zur Protokolldatei für die Fehlerprotokollierung.

`log_error` für `log_error` kein Konfigurationsdateieintrag `log_error` , werden die Werte des Systems standardmäßig auf `@@hostname` .err im `datadir` . Beachten Sie, dass `log_error` keine dynamische Variable ist. Daher werden Änderungen durch Änderungen an einer CNF- oder INI-Datei und durch einen Neustart des Servers vorgenommen (oder indem Sie unten im Link "Manual Page" auf der Seite "Manual Page" auf "Manuelles Löschen und Umbenennen der Fehlerprotokolldatei" klicken).

Die Protokollierung kann nicht für Fehler deaktiviert werden. Sie sind wichtig für die Systemintegrität bei der Fehlerbehebung. Außerdem sind Einträge im Vergleich zum allgemeinen Abfrageprotokoll selten.

Die GLOBAL-Variable `log_warnings` legt die Stufe für die Ausführlichkeit fest, die je nach Serverversion variiert. Der folgende Ausschnitt veranschaulicht:

```
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` wie oben gesehen eine dynamische Variable.

Konfigurationsdateiänderungen in `cnf` und `ini` Dateien können wie folgt aussehen.

```
[mysqld]
log_error        = /path/to/CurrentError.log
log_warnings     = 2
```

MySQL 5.7.2 erweiterte die Warnstufe auf 3 und fügte GLOBAL `log_error_verbosity` . Auch hier

wurde eingeführt in 5.7.2. Es kann dynamisch gesetzt und als Variable überprüft oder über die Konfigurationsdateien von `cnf` oder `ini` werden.

Ab MySQL 5.7.2:

```
[mysqld]
log_error           = /path/to/CurrentError.log
log_warnings        = 2
log_error_verbosity = 3
```

`log_warnings` `error_log_verbosity` Sie auf der MySQL-Handbuchseite mit dem Titel " Das Fehlerprotokoll", insbesondere zum Löschen und Umbenennen der *Fehlerprotokolldatei* , und *im* Abschnitt " *Fehlerprotokoll-Ausführlichkeit* " mit Versionen, die sich auf " `log_warnings` und " `error_log_verbosity` .

Protokolldateien online lesen: https://riptutorial.com/de/mysql/topic/5102/protokolldateien

Kapitel 47: Reguläre Ausdrücke

Einführung

Mit einem regulären Ausdruck können Sie ein Muster für eine komplexe Suche angeben.

Examples

REGEXP / RLIKE

Der `REGEXP` (oder sein Synonym, `RLIKE`) erlaubt das Abgleichen von Mustern basierend auf regulären Ausdrücken.

Betrachten Sie die folgende `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 |
+-------------+-------------+-------------+--------------+----------+
```

Muster ^

Wählen Sie alle Mitarbeiter aus, deren `FIRST_NAME` mit **N** beginnt.

Abfrage

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

Muster $ **

Wählen Sie alle Mitarbeiter aus, deren `PHONE_NUMBER` mit **4569** endet.

Abfrage

```
SELECT * FROM employees WHERE PHONE_NUMBER REGEXP '4569$'
```

```
-- Pattern end with----------------------------------^
```

NICHT REGEXP

Wählen Sie alle Mitarbeiter aus, deren `FIRST_NAME` *nicht* mit **N** beginnt.

Abfrage

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

Regex enthalten

Wählen Sie alle Mitarbeiter , deren `LAST_NAME` enthält und deren `FIRST_NAME` enthält `a` .

Abfrage

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

Ein beliebiges Zeichen zwischen []

Wählen Sie alle Mitarbeiter aus, deren `FIRST_NAME` mit **A** oder **B** oder **C** beginnt.

Abfrage

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

Muster oder |

Wählen Sie alle Mitarbeiter aus, deren `FIRST_NAME` mit **A** oder **B** oder **C** beginnt und mit **r** , **e** oder **i** endet.

Abfrage

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

Übereinstimmungen mit regulären Ausdrücken zählen

Betrachten Sie die folgende Abfrage:

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

`FIRST_NAME REGEXP '^N'` ist *1* oder *0,* abhängig davon, ob `FIRST_NAME` mit `^N` `FIRST_NAME` .

Um es besser zu visualisieren:

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

Zählen Sie schließlich die Gesamtzahl der übereinstimmenden und nicht übereinstimmenden Zeilen mit:

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

Reguläre Ausdrücke online lesen: https://riptutorial.com/de/mysql/topic/9444/regulare-ausdrucke

Kapitel 48: Replikation

Bemerkungen

Die Replikation wird verwendet, um [Backup] -Daten von einem MySQL-Datenbankserver auf einen oder mehrere MySQL-Datenbankserver zu kopieren.

Master - Der MySQL-Datenbankserver, der die zu kopierenden Daten bereitstellt

Slave - Der MySQL-Datenbankserver kopiert Daten, die vom Master bereitgestellt werden

Bei MySQL ist die Replikation standardmäßig asynchron. Dies bedeutet, dass Slaves nicht permanent verbunden sein müssen, um Updates vom Master zu erhalten. Wenn Ihr Slave beispielsweise ausgeschaltet oder nicht mit dem Master verbunden ist und Sie den Slave einschalten oder zu einem späteren Zeitpunkt mit dem Master verbunden sind, wird er automatisch mit dem Master synchronisiert.

Je nach Konfiguration können Sie alle Datenbanken, ausgewählte Datenbanken oder sogar ausgewählte Tabellen in einer Datenbank replizieren.

Replikationsformate

Es gibt zwei Kerntypen von Replikationsformaten

Statement Based Replication (SBR) - repliziert vollständige SQL-Anweisungen. Dabei schreibt der Master SQL-Anweisungen in das Binärlog. Die Replikation des Masters auf den Slave erfolgt durch Ausführen dieser SQL-Anweisungen auf dem Slave.

Row Based Replication (RBR) - *Diese Option* repliziert nur die geänderten Zeilen. Dabei schreibt der Master Ereignisse in das Binärlog, die angeben, wie einzelne Tabellenzeilen geändert werden. Die Replikation des Masters an den Slave erfolgt durch Kopieren der Ereignisse, die die Änderungen an den Tabellenzeilen darstellen, an den Slave.

Sie können auch eine dritte Variante, *Mixed Based Replication (MBR), verwenden* . Hierbei wird sowohl die anweisungsbasierte als auch die zeilenbasierte Protokollierung verwendet. Das Protokoll wird erstellt, je nachdem, welches für die Änderung am besten geeignet ist.

In MySQL-Versionen vor 5.7.7 war das Anweisungsformat der Standard. In MySQL 5.7.7 und höher ist das zeilenbasierte Format der Standard.

Examples

Master-Slave-Replikations-Setup

Betrachten Sie 2 MySQL-Server für die Replikationskonfiguration, einer ist ein Master und der andere ist ein Slave.

Wir werden den Master so konfigurieren, dass er ein Protokoll jeder durchgeführten Aktion speichert. Wir werden den Slave-Server so konfigurieren, dass er sich das Log im Master ansieht, und wenn sich Änderungen im Log auf dem Master ereignen, sollte er dasselbe tun.

Master-Konfiguration

Zunächst müssen wir einen Benutzer auf dem Master erstellen. Dieser Benutzer wird vom Slave verwendet, um eine Verbindung zum Master herzustellen.

```
CREATE USER 'user_name'@'%' IDENTIFIED BY 'user_password';
GRANT REPLICATION SLAVE ON *.* TO 'user_name'@'%';
FLUSH PRIVILEGES;
```

Ändern Sie `user_name` und `user_password` entsprechend Ihrem Benutzernamen und Passwort.

Jetzt sollte die Datei `my.inf` (my.cnf in Linux) bearbeitet werden. Fügen Sie die folgenden Zeilen in den Abschnitt [mysqld] ein.

```
server-id = 1
log-bin = mysql-bin.log
binlog-do-db = your_database
```

In der ersten Zeile wird diesem MySQL-Server eine ID zugewiesen.

In der zweiten Zeile wird MySQL angewiesen, ein Protokoll in die angegebene Protokolldatei zu schreiben. In Linux kann dies wie `log-bin = /home/mysql/logs/mysql-bin.log` . Wenn Sie die Replikation auf einem MySQL-Server starten, auf dem die Replikation bereits verwendet wurde, stellen Sie sicher, dass dieses Verzeichnis keine Replikationsprotokolle enthält.

In der dritten Zeile wird die Datenbank konfiguriert, für die das Protokoll geschrieben werden soll. Sie sollten ersetzen `your_database` mit Ihren Datenbanknamen.

Stellen Sie sicher, dass das `skip-networking` nicht aktiviert ist, und starten Sie den MySQL-Server (Master) erneut.

Slave-Konfiguration

`my.inf` Datei sollte auch im Slave bearbeitet werden. Fügen Sie die folgenden Zeilen in den Abschnitt [mysqld] ein.

```
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
```

In der ersten Zeile wird diesem MySQL-Server eine ID zugewiesen. Diese ID sollte eindeutig sein.

Die zweite Zeile ist die IP-Adresse des Masterservers. Ändern Sie dies entsprechend der IP-Adresse Ihres Mastersystems

In der dritten Zeile wird ein Wiederholungslimit in Sekunden festgelegt.

Die nächsten beiden Zeilen teilen dem Slave den Benutzernamen und das Passwort mit, mit denen er den Master verbindet.

In der nächsten Zeile wird die zu replizierende Datenbank festgelegt.

Die letzten beiden Zeilen, die zum Zuweisen von `relay-log` und `relay-log-index` Dateinamen verwendet werden.

Stellen Sie sicher, dass das `skip-networking` nicht aktiviert ist, und starten Sie den MySQL-Server (Slave) erneut.

Daten zum Slave kopieren

Wenn ständig Daten zum Master hinzugefügt werden, müssen wir den gesamten Datenbankzugriff auf den Master verhindern, damit nichts hinzugefügt werden kann. Dies kann durch Ausführen der folgenden Anweisung in Master erreicht werden.

```
FLUSH TABLES WITH READ LOCK;
```

Wenn dem Server keine Daten hinzugefügt werden, können Sie den obigen Schritt überspringen.

Wir werden eine Datensicherung des Masters mit `mysqldump`

```
mysqldump your_database –u root –p > D://Backup/backup.sql;
```

Ändern `your_database` Ihre `your_database` und Ihr Sicherungsverzeichnis entsprechend Ihrem Setup. Sie haben jetzt eine Datei namens `backup.sql` am angegebenen Ort.

Wenn Ihre Datenbank nicht in Ihrem Slave vorhanden ist, erstellen Sie diese, indem Sie Folgendes ausführen

```
CREATE DATABASE `your_database`;
```

Jetzt müssen wir ein Backup in den Slave MySQL Server importieren.

```
mysql –u root –p your_database  <D://Backup/backup.sql
--->Change `your_database` and backup directory according to your setup
```

Starten Sie die Replikation

Um die Replikation zu starten, müssen wir den Namen der Protokolldatei und die Protokollposition im Master suchen. Also führe folgendes in Master aus

```
SHOW MASTER STATUS;
```

Dadurch erhalten Sie eine Ausgabe wie unten

```
+--------------------+----------+------------------------------+------------------+
| File               | Position | Binlog_Do_DB                 | Binlog_Ignore_DB |
+--------------------+----------+------------------------------+------------------+
| mysql-bin.000001   | 130      | your_database                |                  |
+--------------------+----------+------------------------------+------------------+
```

Dann führe folgendes in Slave aus

```
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;
```

Zuerst stoppen wir den Sklaven. Dann sagen wir genau, wo in der Master-Protokolldatei nachgesehen werden soll. `MASTER_LOG_FILE` Namen `MASTER_LOG_FILE` und `MASTER_LOG_POS` die Werte, die wir durch Ausführen des Befehls `SHOW MASTER STATUS` auf dem Master erhalten haben.

Sie sollten die IP des Masters in `MASTER_HOST` ändern und den Benutzer und das Kennwort entsprechend ändern.

Der Slave wartet jetzt. Der Status des Slaves kann durch Ausführen des folgenden angezeigt werden

```
SHOW SLAVE STATUS;
```

Wenn Sie zuvor `FLUSH TABLES WITH READ LOCK` im Master ausgeführt haben, geben Sie die Tabellen für die Sperre frei, indem Sie Folgendes ausführen

```
UNLOCK TABLES;
```

Jetzt führt der Master für jede durchgeführte Aktion ein Protokoll, und der Slave-Server sieht das Protokoll im Master an. Wenn sich Änderungen am Log auf dem Master ereignen, replizieren Sie den Slave.

Replikationsfehler

Wenn beim Ausführen einer Abfrage auf dem Slave ein Fehler auftritt, stoppt MySQL automatisch die Replikation, um das Problem zu identifizieren und es zu beheben. Dies ist hauptsächlich darauf zurückzuführen, dass ein Ereignis einen doppelten Schlüssel verursachte oder eine Zeile nicht gefunden wurde und es nicht aktualisiert oder gelöscht werden kann. Sie können solche Fehler überspringen, auch wenn dies nicht empfohlen wird

Verwenden Sie die folgende Syntax, um nur eine Abfrage zu überspringen, bei der der Slave hängt

```
SET GLOBAL sql_slave_skip_counter = N;
```

Diese Anweisung überspringt die nächsten N Ereignisse vom Master. Diese Anweisung ist nur gültig, wenn die Slave-Threads nicht ausgeführt werden. Andernfalls wird ein Fehler ausgegeben.

```
STOP SLAVE;
SET GLOBAL sql_slave_skip_counter=1;
START SLAVE;
```

In manchen Fällen ist das in Ordnung. Wenn die Anweisung jedoch Teil einer Transaktion mit mehreren Anweisungen ist, wird sie komplexer, da das Überspringen der fehlererzeugenden Anweisung die gesamte Transaktion überspringt.

Wenn Sie mehr Abfragen überspringen möchten, die denselben Fehlercode erzeugen, und wenn Sie sicher sind, dass das Überspringen dieser Fehler Ihren Slave nicht inkonsistent macht und Sie sie alle überspringen möchten, fügen Sie eine Zeile zum Überspringen des Fehlercodes in Ihrer `my.cnf` .

Beispielsweise möchten Sie möglicherweise alle doppelten Fehler überspringen, die Sie möglicherweise erhalten

```
1062 | Error 'Duplicate entry 'xyz' for key 1' on query
```

Dann fügen Sie Ihrer `my.cnf` folgendes `my.cnf`

```
slave-skip-errors = 1062
```

Sie können auch andere Fehlerarten oder alle Fehlercodes überspringen. Stellen Sie jedoch sicher, dass das Überspringen dieser Fehler Ihren Slave nicht inkonsistent macht. Nachfolgend finden Sie die Syntax und Beispiele

```
slave-skip-errors=[err_code1,err_code2,...|all]

slave-skip-errors=1062,1053
slave-skip-errors=all
slave-skip-errors=ddl_exist_errors
```

Replikation online lesen: https://riptutorial.com/de/mysql/topic/7218/replikation

Kapitel 49: Reservierte Wörter

Einführung

MySQL hat einige spezielle Namen, die als *reservierte Wörter bezeichnet werden* . Ein reserviertes Wort kann nur dann als Bezeichner für eine Tabelle, eine Spalte usw. verwendet werden, wenn es in Backticks (`) eingeschlossen ist. Andernfalls wird ein Fehler ausgegeben.

Um solche Fehler zu vermeiden, verwenden Sie entweder keine reservierten Wörter als Bezeichner oder wickeln Sie den betreffenden Bezeichner in Backticks ein.

Bemerkungen

Nachfolgend sind alle reservierten Wörter (aus der offiziellen Dokumentation) aufgeführt:

- ZUGÄNGLICH
- HINZUFÜGEN
- ALLES
- ÄNDERN
- ANALYSIEREN
- UND
- WIE
- ASC
- ASENSITIVE
- VOR
- ZWISCHEN
- BIGINT
- BINÄR
- KLECKS
- BEIDE
- DURCH
- ANRUF
- KASKADE
- FALL
- VERÄNDERUNG
- VERKOHLEN
- CHARAKTER
- PRÜFEN
- COLLATE
- SÄULE
- BEDINGUNG
- ZWANG
- FORTSETZEN
- KONVERTIEREN
- ERSTELLEN

- KREUZ
- AKTUELLES DATUM
- AKTUELLE UHRZEIT
- AKTUELLER ZEITSTEMPEL
- CURRENT_USER
- MAUSZEIGER
- DATENBANK
- DATENBANKEN
- DAY_HOUR
- DAY_MICROSECOND
- DAY_MINUTE
- DAY_SECOND
- DEC
- DEZIMAL
- ERKLÄREN
- STANDARD
- VERSPÄTET
- LÖSCHEN
- DESC
- BESCHREIBEN
- DETERMINISTISCH
- DISTINCT
- UNTERSCHEIDUNG
- DIV
- DOPPELT
- FALLEN
- DUAL
- JEDER
- SONST
- ELSEIF
- BEIGEFÜGT
- ENTKAM
- EXISTS
- AUSFAHRT
- ERKLÄREN
- FALSCH
- HOLEN
- SCHWEBEN
- FLOAT4
- FLOAT8
- ZUM
- MACHT
- Fremde
- VON
- VOLLER TEXT
- GENERIERT

- ERHALTEN
- GEWÄHREN
- GRUPPE
- HABEN
- HOHE PRIORITÄT
- HOUR_MICROSECOND
- HOUR_MINUTE
- HOUR_SECOND
- OB
- IGNORIEREN
- IM
- INDEX
- IM ORDNER
- INNERE
- INOUT
- UNEMPFINDLICH
- EINFÜGEN
- INT
- INT1
- INT2
- INT3
- INT4
- INT8
- GANZE ZAHL
- INTERVALL
- IN
- IO_AFTER_GTIDS
- IO_BEFORE_GTIDS
- IS
- ITERATE
- BEITRETEN
- SCHLÜSSEL
- SCHLÜSSEL
- TÖTEN
- FÜHREN
- VERLASSEN
- LINKS
- MÖGEN
- GRENZE
- LINEAR
- LINIEN
- BELASTUNG
- ORTSZEIT
- LOCALTIMESTAMP
- SPERREN
- LANGE

- LONGBLOB
- LONGTEXT
- SCHLEIFE
- NIEDRIGE PRIORITÄT
- MASTER_BIND
- MASTER_SSL_VERIFY_SERVER_CERT
- SPIEL
- MAXVALUE
- MEDIUMBLOB
- MEDIUMINT
- MITTELTEXT
- MIDDLEINT
- MINUTE_MICROSECOND
- MINUTE_SECOND
- MOD
- ÄNDERUNGEN
- NATÜRLICH
- NICHT
- NO_WRITE_TO_BINLOG
- NULL
- NUMERISCH
- AUF
- OPTIMIEREN
- OPTIMIZER_COSTS
- MÖGLICHKEIT
- Wahlweise
- ODER
- AUFTRAG
- AUS
- ÄUSSERE
- OUTFILE
- TRENNWAND
- PRÄZISION
- PRIMARY
- VERFAHREN
- REINIGEN
- ANGEBOT
- LESEN
- LESEN
- LESEN SCHREIBEN
- ECHT
- VERWEISE
- REGEXP
- VERÖFFENTLICHUNG
- UMBENENNEN
- WIEDERHOLEN

- ERSETZEN
- BENÖTIGEN
- RESIGNAL
- BESCHRÄNKEN
- RÜCKKEHR
- WIDERRUFEN
- RECHT
- RLIKE
- SCHEMA
- SCHEMA
- SECOND_MICROSECOND
- WÄHLEN
- EMPFINDLICH
- SEPARATOR
- EINSTELLEN
- SHOW
- SIGNAL
- SMALLINT
- SPATIAL
- SPEZIFISCH
- SQL
- SQLEXCEPTION
- SQLSTATE
- SQLWARNING
- SQL_BIG_RESULT
- SQL_CALC_FOUND_ROWS
- SQL_SMALL_RESULT
- SSL
- BEGINNEND
- GELAGERT
- STRAIGHT_JOIN
- TABELLE
- BEENDET
- DANN
- TINYBLOB
- TINYINT
- TINYTEXT
- ZU
- TRAILING
- AUSLÖSEN
- WAHR
- RÜCKGÄNGIG MACHEN
- UNION
- EINZIGARTIG
- FREISCHALTEN
- OHNE VORZEICHEN

- AKTUALISIEREN
- VERWENDUNGSZWECK
- BENUTZEN
- VERWENDUNG
- UTC_DATE
- UTC_TIME
- UTC_TIMESTAMP
- WERTE
- VARBINARY
- VARCHAR
- VARCHARACTER
- UNTERSCHIEDLICH
- VIRTUAL
- WANN
- WOHER
- WÄHREND
- MIT
- SCHREIBEN
- XOR
- JAHR MONAT
- ZEROFILL
- GENERIERT
- OPTIMIZER_COSTS
- GELAGERT
- VIRTUAL

Examples

Fehler aufgrund reservierter Wörter

Wenn Sie versuchen, aus einer so genannten Tabelle eine Auswahl zu `order`

```
select * from order
```

Der Fehler steigt:

> Fehlercode: 1064. Sie haben einen Fehler in Ihrer SQL-Syntax. Überprüfen Sie das Handbuch, das Ihrer MySQL-Server-Version entspricht, auf die richtige Syntax, die in der Nähe von 'order' in Zeile 1 verwendet wird

Reservierte Schlüsselwörter in MySQL müssen mit Backticks (`) geschützt werden.

```
select * from `order`
```

zwischen einem Schlüsselwort und einem Tabellen- oder Spaltennamen unterscheiden.

Siehe auch: Syntaxfehler aufgrund der Verwendung eines reservierten Wortes als Tabellen- oder

Spaltenname in MySQL .

Reservierte Wörter online lesen: https://riptutorial.com/de/mysql/topic/1398/reservierte-worter

Kapitel 50: Schließt sich an

Syntax

- `INNER` und `OUTER` werden ignoriert.
- `FULL` ist nicht in MySQL implementiert.
- "commajoin" (`FROM a,b WHERE ax=by`) ist verpönt; Verwenden Sie stattdessen `FROM a JOIN b ON ax=by` .
- FROM a JOIN b ON ax = by fügt Zeilen ein, die in beiden Tabellen übereinstimmen.
- FROM a LEFT JOIN b ON ax = by schließt alle Zeilen von a `a` , plus übereinstimmende Daten von `b` oder `NULLs` wenn keine übereinstimmende Zeile vorhanden ist.

Examples

Beispiele verbinden

Abfrage zum Erstellen einer Tabelle auf 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;
```

Da wir InnoDB-Tabellen verwenden und wissen, dass user.course und course.id verwandt sind, können wir eine Fremdschlüsselbeziehung angeben:

```
ALTER TABLE `user`
ADD CONSTRAINT `FK_course`
FOREIGN KEY (`course`) REFERENCES `course` (`id`)
ON UPDATE CASCADE;
```

Query beitreten (Inner Join)

```
SELECT user.name, course.name
FROM `user`
INNER JOIN `course` on user.course = course.id;
```

JOIN mit Unterabfrage (Tabelle "Abgeleitete")

```
SELECT x, ...
    FROM ( SELECT y, ... FROM ... ) AS a
    JOIN tbl  ON tbl.x = a.y
    WHERE ...
```

Dadurch wird die Unterabfrage in eine temporäre Tabelle bewerten, dann `JOIN` dass `tbl` .

Vor 5.6 konnte es keinen Index für die temporäre Tabelle geben. Das war also möglicherweise sehr ineffizient:

```
SELECT ...
    FROM ( SELECT y, ... FROM ... ) AS a
    JOIN ( SELECT x, ... FROM ... ) AS b  ON b.x = a.y
    WHERE ...
```

Mit 5.6 ermittelt der Optimierer den besten Index und erstellt ihn im laufenden Betrieb. (Dies hat einen gewissen Aufwand, ist also immer noch nicht "perfekt".)

Ein weiteres allgemeines Paradigma besteht darin, eine Unterabfrage zu haben, um etwas zu initialisieren:

```
SELECT
        @n := @n + 1,
        ...
    FROM ( SELECT @n := 0 ) AS initialize
    JOIN the_real_table
    ORDER BY ...
```

(Hinweis: Dies ist technisch gesehen ein `CROSS JOIN` (kartesisches Produkt), was durch das Fehlen von `ON` angezeigt wird. Dies ist jedoch effizient, da die Unterabfrage nur eine Zeile zurückgibt, die mit den n Zeilen in der `the_real_table` .)

Kunden mit Bestellungen abrufen - Variationen eines Themas

Dies wird alle Bestellungen für alle Kunden erhalten:

```
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;
```

Dadurch wird die Anzahl der Bestellungen für jeden Kunden gezählt:

```
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;
```

Auch zählt, aber wahrscheinlich schneller:

```
SELECT  c.CustomerName,
        ( SELECT COUNT(*) FROM Orders WHERE CustomerID = c.CustomerID ) AS 'Order Count'
    FROM Customers AS c
    ORDER BY c.CustomerName;
```

Listen Sie nur den Kunden mit Bestellungen auf.

```
SELECT  c.CustomerName,
    FROM Customers AS c
    WHERE EXISTS ( SELECT * FROM Orders WHERE CustomerID = c.CustomerID )
    ORDER BY c.CustomerName;
```

Voller äußerer Join

MySQL unterstützt den `FULL OUTER JOIN` , es gibt jedoch Möglichkeiten, einen zu emulieren.

Einrichten der Daten

```
-- ----------------------------
-- 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);
```

Was wollen wir sehen?

Wir möchten eine Liste erhalten, in der wir sehen, wer welche Tools besitzt und welche Tools möglicherweise keinen Eigentümer haben.

Die Anfragen

Um dies zu erreichen, können wir zwei Abfragen mit `UNION` kombinieren. In dieser ersten Abfrage verbinden wir die Tools der Eigentümer mit einem `LEFT JOIN` . Dadurch werden alle unsere Besitzer zu unserem Resultset hinzugefügt. Dabei spielt es keine Rolle, ob sie tatsächlich Tools besitzen.

In der zweiten Abfrage verwenden wir eine `RIGHT JOIN` , um die Werkzeuge mit den Besitzern zu verbinden. Auf diese Weise erhalten wir alle Werkzeuge aus unserem Resultset. Wenn sie niemandem gehören, enthält ihre Eigentümerspalte einfach `NULL` . Durch das Hinzufügen einer `WHERE` -clause, die nach `owners.owner_id IS NULL` , definieren wir das Ergebnis als die Datasets, die nicht bereits von der ersten Abfrage zurückgegeben wurden, da wir nur nach den Daten in der rechts verbundenen Tabelle suchen.

Da wir `UNION ALL` das Resultset der zweiten Abfrage an das erste Resultset der Abfragen angehängt.

```
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)
```

Innenverbindung für 3 Tische

Angenommen, wir haben drei Tabellen, die für einfache Websites mit Tags verwendet werden können.

- Faust Tisch ist für Beiträge.
- Zweiter für Tags
- Dritter für Tags & Post-Relation

Faust Tisch "Videospiel"

Ich würde	Titel	reg_date	Inhalt
1	BioShock Infinite	2016-08-08	....

"Tags" -Tabelle

Ich würde	Name
1	Yennefer
2	Elisabeth

"tags_meta" -Tabelle

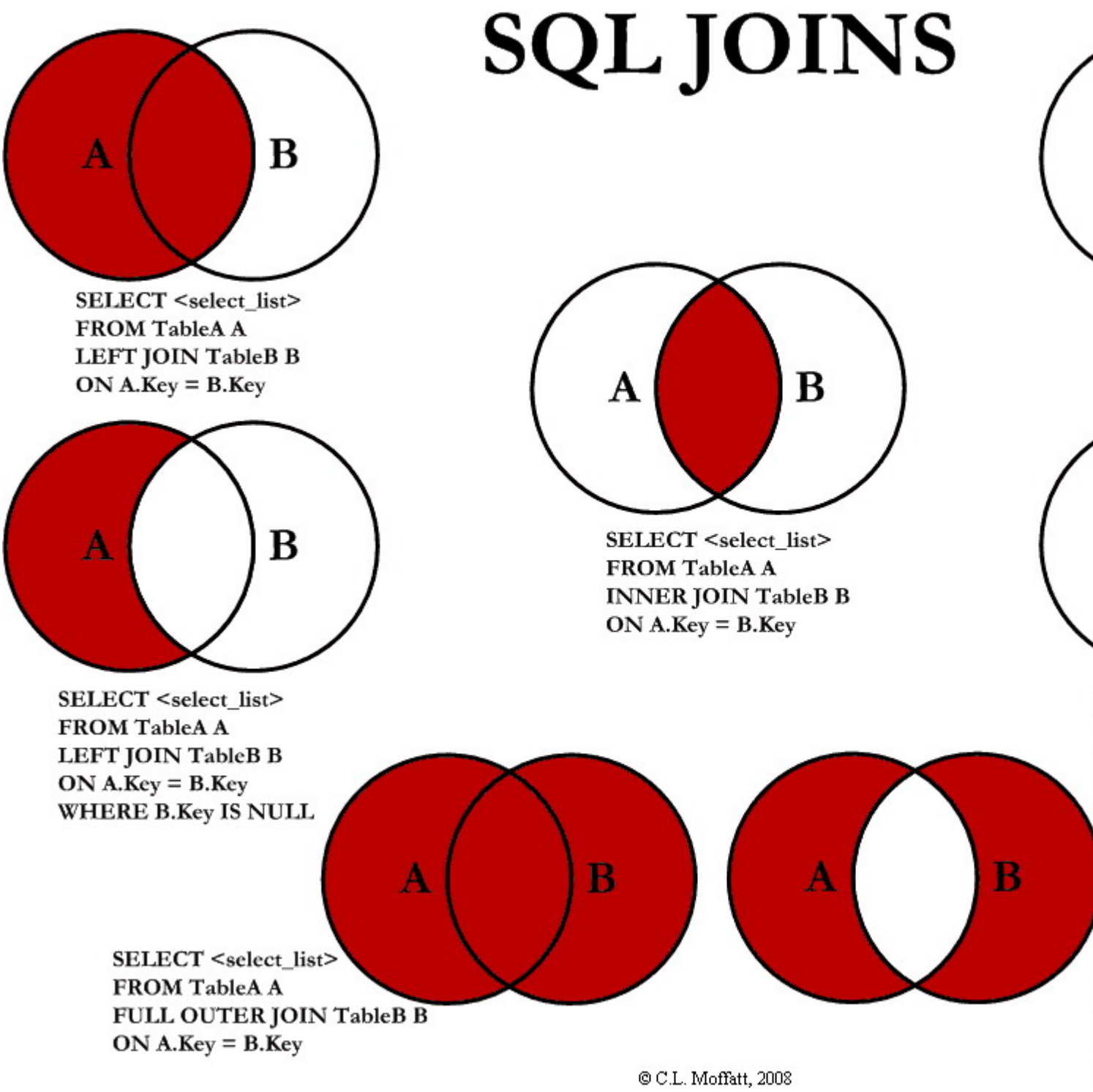
post_id	tag_id
1	2

```
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
```

Dieser Code kann alle Beiträge zurückgeben, die sich auf dieses Tag beziehen.

Joins visualisiert

Wenn Sie eine visuell orientierte Person sind, kann dieses Venn-Diagramm Ihnen helfen, die verschiedenen Arten von `JOIN` verstehen, die in MySQL existieren.



Schließt sich an online lesen: https://riptutorial.com/de/mysql/topic/2736/schlie-t-sich-an

Kapitel 51: Serverinformation

Parameter

Parameter	Erläuterung
GLOBAL	Zeigt die Variablen an, wie sie für den gesamten Server konfiguriert sind. Wahlweise.
SESSION	Zeigt die Variablen an, die nur für diese Sitzung konfiguriert sind. Wahlweise.

Examples

SHOW VARIABLES Beispiel

Um alle Servervariablen abzurufen, führen Sie diese Abfrage entweder im SQL-Fenster Ihrer bevorzugten Schnittstelle (PHPMyAdmin oder andere) oder in der MySQL-CLI-Schnittstelle aus

```
SHOW VARIABLES;
```

Sie können angeben, ob Sie die Sitzungsvariablen oder die globalen Variablen wie folgt verwenden möchten:

Sitzungsvariablen:

```
SHOW SESSION VARIABLES;
```

Globale Variablen:

```
SHOW GLOBAL VARIABLES;
```

Wie jeder andere SQL-Befehl können Sie Ihrer Abfrage Parameter hinzufügen, beispielsweise den LIKE-Befehl:

```
SHOW [GLOBAL | SESSION] VARIABLES LIKE 'max_join_size';
```

Oder mit Platzhaltern:

```
SHOW [GLOBAL | SESSION] VARIABLES LIKE '%size%';
```

Sie können die Ergebnisse der SHOW-Abfrage auch mit einem WHERE-Parameter wie folgt filtern:

```
SHOW [GLOBAL | SESSION] VARIABLES WHERE VALUE > 0;
```

SHOW STATUS-Beispiel

Um den Status des Datenbankservers zu erhalten, führen Sie diese Abfrage entweder im SQL-Fenster Ihrer bevorzugten Schnittstelle (PHPMyAdmin oder einer anderen) oder in der MySQL-CLI-Schnittstelle aus.

```
SHOW STATUS;
```

Sie können angeben, ob Sie den SESSION- oder GLOBAL-Status Ihres Servers erhalten möchten: Sitzungsstatus:

```
SHOW SESSION STATUS;
```

Globaler Status:

```
SHOW GLOBAL STATUS;
```

Wie jeder andere SQL-Befehl können Sie Ihrer Abfrage Parameter hinzufügen, beispielsweise den LIKE-Befehl:

```
SHOW [GLOBAL | SESSION] STATUS LIKE 'Key%';
```

Oder der Where-Befehl:

```
SHOW [GLOBAL | SESSION] STATUS WHERE VALUE > 0;
```

Der Hauptunterschied zwischen GLOBAL und SESSION besteht darin, dass mit dem GLOBAL-Modifikator der Befehl aggregierte Informationen zum Server und seinen gesamten Verbindungen anzeigt, während der SESSION-Modifikator nur die Werte für die aktuelle Verbindung anzeigt.

Serverinformation online lesen: https://riptutorial.com/de/mysql/topic/9924/serverinformation

Kapitel 52: Sicherheit über GRANTs

Examples

Beste Übung

Begrenzen Sie root (und alle anderen Benutzer mit SUPER-Berechtigungen) auf

```
GRANT ... TO root@localhost ...
```

Das verhindert den Zugriff von anderen Servern. Sie sollten SUPER nur sehr wenigen Menschen aushändigen, und sie sollten sich ihrer Verantwortung bewusst sein. Die Anwendung sollte nicht SUPER haben.

Beschränken Sie die Anwendungs-Logins auf die Datenbank, die sie verwendet:

```
GRANT ... ON dbname.* ...
```

Auf diese Weise kann jemand, der sich in den Anwendungscode stürzt, nicht an dbname vorbei gelangen. Dies kann durch eine der folgenden Optionen weiter verfeinert werden:

```
GRANT SELECT ON dname.* ...    -- "read only"
GRANT ... ON dname.tblname ... -- "just one table"
```

Das Readonly benötigt möglicherweise auch "sichere" Dinge wie

```
GRANT SELECT, CREATE TEMPORARY TABLE ON dname.* ...    -- "read only"
```

Wie Sie sagen, gibt es keine absolute Sicherheit. Mein Punkt hier ist, dass Sie ein paar Dinge tun können, um Hacker zu verlangsamen. (Gleiches gilt für ehrliche Leute, die vermasseln.)

In seltenen Fällen müssen Sie die Anwendung möglicherweise nur für root verwenden. Dies kann über eine "gespeicherte Prozedur" erfolgen, die über `SECURITY DEFINER` (und root definiert sie). Dadurch wird nur angezeigt, was der SP macht, was beispielsweise eine bestimmte Aktion für eine bestimmte Tabelle sein kann.

Host (von Benutzer @ Host)

Der "Host" kann entweder ein Hostname oder eine IP-Adresse sein. Es kann sich auch um Platzhalter handeln.

```
GRANT SELECT ON db.* TO sam@'my.domain.com' IDENTIFIED BY 'foo';
```

Beispiele: Hinweis: Diese müssen in der Regel zitiert werden

```
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.)
```

Die Verwendung von `localhost` hängt von der Sicherheit des Servers ab. Für eine optimale Vorgehensweise sollte `root` nur über localhost zugelassen werden. In einigen Fällen bedeuten diese das Gleiche: `0.0.0.1` und `::1` .

Sicherheit über GRANTs online lesen: https://riptutorial.com/de/mysql/topic/5131/sicherheit-uber-grants

Kapitel 53: Sichern Sie mit mysqldump

Syntax

- mysqldump -u [Benutzername] -p [Kennwort] [andere Optionen] Datenbankname> dumpFileName.sql /// Zum Sichern einer einzelnen Datenbank
- mysqldump -u [Benutzername] -p [Kennwort] [andere Optionen] Datenbankname [Tabellenname1 Tabellenname2 Tabellenname2 ...]> dumpFileName.sql /// So sichern Sie eine oder mehrere Tabellen
- mysqldump -u [Benutzername] -p [Kennwort] [Weitere Optionen] --databases Datenbankname1 Datenbankname2 Datenbankname3 ...> dumpFileName.sql /// So sichern Sie eine oder mehrere vollständige Datenbanken
- mysqldump -u [Benutzername] -p [Kennwort] [andere Optionen] --all -abases> dumpFileName.sql /// So sichern Sie den gesamten MySQL-Server

Parameter

Möglichkeit	Bewirken
-	# Server-Anmeldeoptionen
`-h` (`--host`)	Host (IP-Adresse oder Hostname), zu dem eine Verbindung hergestellt werden soll. Der Standardwert ist `localhost` (`127.0.0.1`). Beispiel: `-h localhost`
`-u` (`--user`)	MySQL-Benutzer
`-p` (`--password`)	MySQL-Passwort **Wichtig** : Bei der Verwendung von `-p` darf zwischen der Option und dem Kennwort kein Leerzeichen stehen. Beispiel: `-pMyPassword`
-	# Dump-Optionen
`--add-drop-database`	Fügen Sie vor jeder `CREATE DATABASE` Anweisung eine `DROP DATABASE CREATE DATABASE` Anweisung hinzu. Nützlich, wenn Sie Datenbanken auf dem Server ersetzen möchten.
`--add-drop-table`	Fügen Sie vor jeder `CREATE TABLE` Anweisung eine `DROP TABLE CREATE TABLE` Anweisung hinzu. Nützlich, wenn Sie Tabellen auf dem Server ersetzen möchten.
`--no-create-db`	Unterdrücken Sie die `CREATE DATABASE` Anweisungen im Speicherauszug. Dies ist nützlich, wenn Sie sicher sind, dass die Datenbank (en), die Sie sichern, bereits auf dem Server vorhanden sind, auf den Sie die Sicherung laden.
`-t` (`--no-create-info`)	Unterdrückt alle `CREATE TABLE` Anweisungen im Speicherauszug. Dies ist nützlich, wenn Sie nur die Daten aus den Tabellen sichern möchten und die

Möglichkeit	Bewirken
	Sicherungsdatei verwenden, um identische Tabellen in einer anderen Datenbank / einem anderen Server aufzufüllen.
`-d` (`--no-data`)	Schreiben Sie keine Tabelleninformationen. Dadurch werden nur die `CREATE TABLE` Anweisungen `CREATE TABLE` . Nützlich zum Erstellen von "Template" - Datenbanken
`-R` (`--routines`)	Fügen Sie gespeicherte Prozeduren / Funktionen in den Speicherauszug ein.
`-K` (`--disable-keys`)	Deaktivieren Sie die Schlüssel für jede Tabelle, bevor Sie die Daten einfügen, und aktivieren Sie die Schlüssel, nachdem die Daten eingefügt wurden. Dies beschleunigt das Einfügen nur in MyISAM-Tabellen mit nicht eindeutigen Indizes.

Bemerkungen

Die Ausgabe einer `mysqldump` ist eine leicht kommentierte Datei, die sequenzielle SQL-Anweisungen enthält, die mit der Version der MySQL-Dienstprogramme kompatibel sind, mit der sie erstellt wurde (wobei der Kompatibilität mit früheren Versionen Rechnung getragen wurde, jedoch keine Garantie für zukünftige Versionen). Daher umfasst die Wiederherstellung einer Datenbank `mysqldump` ed die Ausführung dieser Anweisungen. Im Allgemeinen diese Datei

- `DROP` die erste angegebene Tabelle oder Ansicht
- `CREATE` diese Tabelle oder Ansicht
- Für Tabellen, die mit Daten `--no-data` werden (dh ohne die Option `--no-data`)
    - `LOCK` der Tisch
    - `INSERT` alle Zeilen der ursprünglichen Tabelle in einer Anweisung
- `UNLOCK TABLES`
- Wiederholt das Obige für alle anderen Tabellen und Ansichten
- `DROP` die erste enthaltene Routine
- `CREATE` diese Routine
- Wiederholt das gleiche für alle anderen Routinen

Das Vorhandensein von `DROP` vor `CREATE` für jede Tabelle bedeutet, dass bei Vorhandensein des Schemas, unabhängig davon, ob es leer ist oder nicht, die Verwendung einer `mysqldump` Datei für die Wiederherstellung die darin enthaltenen Daten auffüllt oder überschreibt.

Examples

Sicherung einer Datenbank oder Tabelle erstellen

Erstellen Sie einen Schnappschuss einer gesamten Datenbank:

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

Erstellen Sie einen Schnappschuss mehrerer Datenbanken:

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

Erstellen Sie eine Momentaufnahme einer oder mehrerer Tabellen:

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

Erstellen Sie einen Schnappschuss *ohne* eine oder mehrere Tabellen:

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

Die Dateierweiterung `.sql` ist eine Frage des Stils. Jede Erweiterung würde funktionieren.

Angabe von Benutzername und Passwort

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

Wenn Sie das Kennwort in der Befehlszeile angeben müssen (z. B. in einem Skript), können Sie es nach der Option `-p` *ohne* Leerzeichen hinzufügen:

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

Wenn Ihr Kennwort Leerzeichen oder Sonderzeichen enthält, denken Sie daran, abhängig von Ihrer Shell / Ihrem System die Escape-Funktion zu verwenden.

Optional ist das erweiterte Formular:

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

(Die Angabe des Kennworts in der Befehlszeile ist aus Sicherheitsgründen nicht empfohlen.)

Wiederherstellen einer Sicherung einer Datenbank oder Tabelle

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

Beachten Sie, dass:

- `db_name` muss eine vorhandene Datenbank sein.
- Ihr authentifizierter Benutzer verfügt über ausreichende Berechtigungen, um alle Befehle in `filename.sql` auszuführen.
- Die Dateierweiterung `.sql` ist eine Frage des Stils. Jede Erweiterung würde funktionieren.
- Sie können keinen Tabellennamen angeben, in den geladen werden soll, obwohl Sie einen Namen zum Speichern angeben könnten. Dies muss in `filename.sql` geschehen.

Alternativ können Sie im **MySQL-Befehlszeilentool** mithilfe des Befehls source den Befehl wiederherstellen (oder jedes andere Skript ausführen):

```
source filename.sql
```

oder

```
\. filename.sql
```

mysqldump von einem Remote-Server mit Komprimierung

`--compress` Option `--compress` an `mysqldump` , um die Komprimierung über das `--compress` für eine schnellere Übertragung zu verwenden. Beispiel:

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

Wichtig: Wenn Sie die *Quelldatenbank* nicht `--lock-tables=false` möchten, sollten Sie auch `--lock-tables=false` . Auf diese Weise erhalten Sie möglicherweise kein intern konsistentes db-Bild.

Um die Datei auch komprimiert zu speichern, können Sie an `gzip` .

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

Wiederherstellen einer gzippten mysqldump -Datei ohne Dekomprimierung

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

Hinweis: `-c` bedeutet, dass die Ausgabe in stdout geschrieben wird.

Backup direkt auf Amazon S3 mit Komprimierung

Wenn Sie eine vollständige Sicherung einer großen MySql-Installation erstellen möchten und nicht über ausreichend lokalen Speicher verfügen, können Sie die Sicherung direkt in einem Amazon S3-Bucket sichern und komprimieren. Es empfiehlt sich auch, dies ohne das DB-Kennwort als Teil des Befehls zu tun:

```
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
```

Sie werden aufgefordert, das Kennwort einzugeben. Danach beginnt die Sicherung.

Übertragen von Daten von einem MySQL-Server zu einem anderen

Wenn Sie eine Datenbank von einem Server auf einen anderen kopieren möchten, haben Sie zwei Möglichkeiten:

Option 1:

1. Speichern Sie die Sicherungsdatei auf dem Quellserver
2. Kopieren Sie die Dump-Datei auf Ihren Zielserver
3. Laden Sie die Dump-Datei in Ihren Zielserver

Auf dem Quellserver:

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

Kopieren Sie auf dem Zielserver die Sicherungsdatei und führen Sie Folgendes aus:

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

Option 2:

Wenn der Zielserver eine Verbindung zum Hostserver herstellen kann, können Sie die Datenbank mithilfe einer Pipeline von einem Server auf den anderen kopieren:

Auf dem Zielserver

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

In ähnlicher Weise kann das Skript auf dem Quellserver ausgeführt werden und an das Ziel weiterleiten. In beiden Fällen ist es wahrscheinlich wesentlich schneller als Option 1.

Sicherungsdatenbank mit gespeicherten Prozeduren und Funktionen

Standardmäßig gespeicherte Prozeduren und Funktionen oder nicht von `mysqldump` generiert, müssen Sie den Parameter `--routines` (oder `-R`) hinzufügen:

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

Bei der Verwendung von `--routines` die Erstellung und Änderung Zeitstempel werden nicht beibehalten, sondern sollten Sie den Inhalt entleeren und neu zu laden `mysql.proc` .

Sichern Sie mit mysqldump online lesen: https://riptutorial.com/de/mysql/topic/604/sichern-sie-mit-mysqldump

Kapitel 54: SORTIEREN NACH

Examples

Kontexte

Die Klauseln in einem `SELECT` haben eine bestimmte Reihenfolge:

```
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
```

Basic

BESTELLEN DURCH x

`x` kann ein beliebiger Datentyp sein.

- `NULLs` gehen vor NULLs vor.
- Der Standardwert ist `ASC` (niedrigste bis höchste)
- Zeichenfolgen (`VARCHAR` usw.) werden gemäß der `COLLATION` der Deklaration `COLLATION`
- `ENUMs` werden nach der Deklarationsreihenfolge ihrer Zeichenfolgen geordnet.

Aufsteigend absteigend

```
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`
- `NULLs` sogar bei `DESC` erster Stelle.
- In den obigen Beispielen können `INDEX(x)` , `INDEX(lastname, firstname)` , `INDEX(submit_date)` die Leistung erheblich verbessern.

Aber ... Beim Mischen von `ASC` und `DESC` , wie im letzten Beispiel, kann kein zusammengesetzter Index verwendet werden. `INDEX(submit_date DESC, id ASC)` hilft auch nicht - " `DESC` " wird in der `INDEX` Deklaration syntaktisch erkannt, aber ignoriert.

Einige Tricks

```
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.
```

Kundenspezifische Bestellung

```
SELECT * FROM some_table WHERE id IN (118, 17, 113, 23, 72)
ORDER BY FIELD(id, 118, 17, 113, 23, 72);
```

Gibt das Ergebnis in der angegebenen Reihenfolge der IDs zurück.

Ich würde	...
118	...
17	...
113	...
23	...
72	...

Nützlich, wenn die IDs bereits sortiert sind und Sie nur die Zeilen abrufen müssen.

SORTIEREN NACH online lesen: https://riptutorial.com/de/mysql/topic/5469/sortieren-nach

Kapitel 55: SSL-Verbindungsaufbau

Examples

Setup für Debian-basierte Systeme

(Dies setzt voraus, dass MySQL installiert wurde und `sudo` verwendet wird.)

CA und SSL-Schlüssel generieren

Stellen Sie sicher, dass OpenSSL und Bibliotheken installiert sind:

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

Als Nächstes erstellen und geben Sie ein Verzeichnis für die SSL-Dateien ein:

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

Erstellen Sie zum Generieren von Schlüsseln eine Zertifizierungsstelle (Certificate Authority, CA) zum Signieren der Schlüssel (selbstsigniert):

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

Die bei jeder Eingabeaufforderung eingegebenen Werte haben keinen Einfluss auf die Konfiguration. Als Nächstes erstellen Sie einen Schlüssel für den Server und signieren Sie mit der Zertifizierungsstelle von vor:

```
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
```

Dann erstellen Sie einen Schlüssel für einen Kunden:

```
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
```

Um sicherzustellen, dass alles korrekt eingerichtet wurde, überprüfen Sie die Schlüssel:

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

Hinzufügen der Schlüssel zu MySQL

Öffnen Sie die MySQL-Konfigurationsdatei . Zum Beispiel:

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

`[mysqld]` Abschnitt `[mysqld]` die folgenden Optionen hinzu:

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

Starten Sie MySQL neu. Zum Beispiel:

```
service mysql restart
```

Testen Sie die SSL-Verbindung

`ssl-ca` dieselbe Weise eine Verbindung her und übergeben Sie die zusätzlichen Optionen `ssl-ca` , `ssl-cert` und `ssl-key` mit dem generierten Client-Schlüssel. Angenommen, `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
```

Überprüfen Sie nach dem Anmelden, ob die Verbindung tatsächlich sicher ist:

```
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  |
+---------------+-----------------------------------------+
```

Sie können auch überprüfen:

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

SSL erzwingen

Dies erfolgt über `GRANT` mit `REQUIRE SSL` :

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

Nun `superman` *muss* über SSL verbinden.

Wenn Sie keine Clientschlüssel verwalten möchten, verwenden Sie den Clientschlüssel von früher und verwenden Sie ihn automatisch für alle Clients. Öffnen Sie die MySQL-Konfigurationsdatei , zum Beispiel:

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

Fügen Sie im Abschnitt `[client]` die folgenden Optionen hinzu:

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

Jetzt muss `superman` nur noch Folgendes eingeben, um sich über SSL anzumelden:

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

Das Herstellen einer Verbindung von einem anderen Programm, beispielsweise in Python, erfordert normalerweise nur einen zusätzlichen Parameter für die Verbindungsfunktion. Ein Python-Beispiel:

```
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)
```

Referenzen und weiterführende Literatur:

- 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

Setup für CentOS7 / RHEL7

In diesem Beispiel werden zwei Server vorausgesetzt:

1. dbserver (wo unsere Datenbank lebt)

2. appclient (wo unsere Anwendungen leben)

FWIW, beide Server erzwingen SELinux.

Melden Sie sich zuerst bei dbserver an

Erstellen Sie ein temporäres Verzeichnis zum Erstellen der Zertifikate.

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

Erstellen Sie die Serverzertifikate

```
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
```

Serverzertifikate nach / etc / pki / tls / certs / mysql / verschieben

Der Verzeichnispfad setzt CentOS oder RHEL voraus (passen Sie ihn ggf. für andere Distros an):

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

Stellen Sie sicher, dass Sie Berechtigungen für den Ordner und die Dateien festlegen. mysql benötigt vollen Besitz und Zugriff.

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

Konfigurieren Sie nun 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
```

Dann

```
systemctl restart mariadb
```

Vergessen Sie nicht, Ihre Firewall zu öffnen, um Verbindungen vom Client (mit IP 1.2.3.4) zuzulassen.

```
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.
```

Starten Sie nun die Firewall neu

```
service firewalld restart
```

Melden Sie sich als Nächstes beim MySQL-Server von dbserver an:

```
mysql -uroot -p
```

Geben Sie Folgendes ein, um einen Benutzer für den Client zu erstellen. note Hinweis SSL in GRANT-Anweisung angeben.

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

Sie sollten sich im ersten Schritt noch in / root / certs / mysql befinden. Wenn nicht, kehren Sie für einen der folgenden Befehle zurück.

Erstellen Sie die Client-Zertifikate

```
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
```

Hinweis : Ich habe sowohl für Server- als auch für Client-Zertifikate denselben allgemeinen Namen verwendet. YMMV.

Stellen Sie sicher, dass Sie für den nächsten Befehl noch / root / certs / mysql / sind

Server- und Client-CA-Zertifikat in einer einzigen Datei kombinieren:

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

Stellen Sie sicher, dass Sie zwei Zertifikate sehen:

```
cat ca.pem
```

ENDE DER SERVER-SEITENARBEIT FÜR JETZT.

Öffnen Sie ein anderes Terminal und

```
ssh appclient
```

Erstellen Sie wie zuvor eine permanente Heimat für die Clientzertifikate

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

Platzieren Sie nun die Clientzertifikate (erstellt auf dem Datenbankserver) im Anwendungsclient. Sie können die Dateien entweder scp oder die Dateien nacheinander kopieren und einfügen.

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

Stellen Sie erneut sicher, dass Sie Berechtigungen für den Ordner und die Dateien festlegen. mysql benötigt vollen Besitz und Zugriff.

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

Sie sollten drei Dateien haben, die jeweils dem Benutzer mysql gehören:

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

Bearbeiten Sie nun die MariaDB / MySQL-Konfiguration des Anwendungs- `[client]` Abschnitt `[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
```

Starten Sie den Mariadb-Dienst von appclient erneut:

```
systemctl restart mariadb
```

immer noch auf dem Client hier

Dies sollte zurückgeben: ssl TRUE

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

Melden Sie sich jetzt bei der mysql-Instanz von appclient an

```
mysql -uroot -p
```

Sollte für beide Variablen JA stehen

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

Anfangs habe ich gesehen

```
 have_openssl NO
```

Ein kurzer Blick in mariadb.log ergab:

> SSL-Fehler: Zertifikat kann nicht von '/etc/pki/tls/certs/mysql/client-cert.pem' abgerufen werden.

Das Problem war, dass root-client.pem und der dazugehörige Ordner gehörten. Die Lösung bestand darin, den Besitz von / etc / pki / tls / certs / mysql / auf mysql zu setzen.

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

Starten Sie mariadb neu, falls erforderlich

JETZT KÖNNEN WIR DIE SICHERE VERBINDUNG TESTEN

Wir sind immer noch hier

Versuchen Sie, mit dem oben erstellten Konto eine Verbindung zur mysql-Instanz von dbserver herzustellen.

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

Mit etwas Glück sollten Sie ohne Fehler eingeloggt sein.

Geben Sie den folgenden Befehl an der MariaDB / MySQL-Eingabeaufforderung aus, um zu bestätigen, dass Sie mit SSL verbunden sind:

```
\s
```

Das ist ein Backslash, auch bekannt als Status

Daraufhin wird der Status Ihrer Verbindung angezeigt, der ungefähr wie folgt aussehen sollte:

```
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
```

Wenn Sie bei Ihrem Verbindungsversuch auf Erlaubnis verweigerte Fehler erhalten, überprüfen Sie Ihre obige GRANT-Anweisung, um sicherzustellen, dass keine verstreuten Zeichen oder 'Markierungen vorhanden sind.

Wenn Sie SSL-Fehler haben, lesen Sie diese Anleitung erneut durch, um sicherzustellen, dass die Schritte ordnungsgemäß ausgeführt werden.

Dies funktionierte mit RHEL7 und wird wahrscheinlich auch mit CentOS7 funktionieren. Kann nicht bestätigen, ob diese genauen Schritte anderswo funktionieren.

Ich hoffe, das erspart jemandem etwas Zeit und Ärger.

SSL-Verbindungsaufbau online lesen: https://riptutorial.com/de/mysql/topic/7563/ssl-verbindungsaufbau

Kapitel 56: Stellen Sie das Standard-Root-Passwort für MySQL 5.7 und höher wieder her

Einführung

Nach MySQL 5.7 müssen wir bei der Installation von MySQL manchmal kein Root-Konto erstellen oder ein Root-Passwort angeben. Wenn wir den Server starten, wird das Standardkennwort standardmäßig in der Datei `mysqld.log` gespeichert. Wir müssen uns mit diesem Passwort am System anmelden und es ändern.

Bemerkungen

Das Wiederherstellen und Zurücksetzen des Standard-Root-Kennworts mit dieser Methode gilt nur für MySQL 5.7+

Examples

Was passiert beim ersten Start des Servers?

Angenommen, das Datenverzeichnis des Servers ist leer:

- Der Server wird initialisiert.
- SSL-Zertifikate und Schlüsseldateien werden im Datenverzeichnis generiert.
- Das validate_password-Plugin ist installiert und aktiviert.
- Das Superuser-Konto 'root' @ 'localhost' wird erstellt. Das Passwort für den Superuser wird festgelegt und in der Fehlerprotokolldatei gespeichert.

So ändern Sie das Root-Kennwort mithilfe des Standardkennworts

So zeigen Sie das Standardkennwort "root" an:

```
shell> sudo grep 'temporary password' /var/log/mysqld.log
```

Ändern Sie das root-Passwort so schnell wie möglich, indem Sie sich mit dem generierten temporären Passwort anmelden und ein benutzerdefiniertes Passwort für das Superuser-Konto festlegen:

```
shell> mysql -uroot -p

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass5!';
```

Hinweis: Das Plug-In für validate_password von MySQL wird standardmäßig installiert. Dies setzt

voraus, dass Kennwörter mindestens einen Großbuchstaben, einen Kleinbuchstaben, eine Ziffer und ein Sonderzeichen enthalten und dass die Gesamtlänge des Kennworts mindestens 8 Zeichen beträgt.

Root-Passwort zurücksetzen, wenn "/ var / run / mysqld 'für UNIX-Socket-Datei nicht vorhanden ist"

Wenn ich das Passwort vergessen habe, bekomme ich eine Fehlermeldung.

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

Passwort eingeben:

FEHLER 1045 (28000): Zugriff für Benutzer 'root' @ 'localhost' verweigert (mit Kennwort: YES)

Ich habe versucht, das Problem zu lösen, indem ich zuerst den Status wusste:

```
$ systemctl status mysql.service
```

mysql.service - MySQL Community Server geladen: geladen (/lib/systemd/system/mysql.service; aktiviert; Hersteller-Voreinstellung: de Aktiv: aktiv (läuft) seit Do. 2017-06-08 14:31:33 IST; vor 38s

Dann habe ich den Code `mysqld_safe --skip-grant-tables &` aber ich bekomme den Fehler:

> mysqld_safe Das Verzeichnis '/ var / run / mysqld' für die UNIX-Socketdatei ist nicht vorhanden.

```
$ systemctl stop  mysql.service
$ ps -eaf|grep mysql
$ mysqld_safe --skip-grant-tables &
```

> I löste:

```
$ mkdir -p /var/run/mysqld
$ chown mysql:mysql /var/run/mysqld
```

Jetzt benutze ich den gleichen Code wie `mysqld_safe --skip-grant-tables &` und bekomme

> mysqld_safe Starten des mysqld-Daemons mit Datenbanken aus / var / lib / mysql

Wenn ich `$ mysql -u root` bekomme ich:

Serverversion: 5.7.18-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle und / oder ihre verbundenen Unternehmen. Alle Rechte vorbehalten.

Oracle ist eine eingetragene Marke der Oracle Corporation und / oder ihrer verbundenen Unternehmen. Andere Namen sind möglicherweise Marken ihrer jeweiligen Eigentümer.

Geben Sie "help" ein. oder '\ h' um Hilfe. Geben Sie '\ c' ein, um die aktuelle Eingabeanweisung zu löschen.

mysql>

Nun ist es Zeit das Passwort zu ändern:

```
mysql> use mysql
mysql> describe user;
```

Lesen von Tabelleninformationen zur Vervollständigung von Tabellen- und Spaltennamen Sie können diese Funktion deaktivieren, um einen schnelleren Start mit -A zu ermöglichen

Datenbank geändert

```
mysql> FLUSH PRIVILEGES;
mysql> SET PASSWORD FOR root@'localhost' = PASSWORD('newpwd');
```

oder Wenn Sie ein MySQL-Root-Konto haben, das von überall aus eine Verbindung herstellen kann, sollten Sie auch Folgendes tun:

```
UPDATE mysql.user SET Password=PASSWORD('newpwd') WHERE User='root';
```

Alternative Methode:

```
   USE mysql
   UPDATE user SET Password = PASSWORD('newpwd')
   WHERE Host = 'localhost' AND User = 'root';
```

Und wenn Sie ein Root-Konto haben, auf das von überall aus zugegriffen werden kann:

```
 USE mysql
 UPDATE user SET Password = PASSWORD('newpwd')
 WHERE Host = '%' AND User = 'root';`enter code here
```

Jetzt müssen Sie von MySQL `quit` und stoppen / starten

```
FLUSH PRIVILEGES;
sudo /etc/init.d/mysql stop
sudo /etc/init.d/mysql start
```

nun wieder `mysql -u root -p 'und benutze das neue Passwort um zu bekommen

mysql>

Stellen Sie das Standard-Root-Passwort für MySQL 5.7 und höher wieder her online lesen: https://riptutorial.com/de/mysql/topic/9563/stellen-sie-das-standard-root-passwort-fur-mysql-5-7-und-hoher-wieder-her

Kapitel 57: Stellen Sie das verlorene Root-Passwort wieder her

Examples

Legen Sie das root-Passwort fest und aktivieren Sie den root-Benutzer für den Socket- und http-Zugriff

Behebt das Problem: Zugriff verweigert für Benutzer root mit dem Passwort YES Stop mySQL:

```
sudo systemctl stop mysql
```

Starten Sie mySQL neu und überspringen Sie die Grant-Tabellen:

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

Anmeldung:

```
mysql -u root
```

Prüfen Sie in der SQL-Shell, ob Benutzer vorhanden sind:

```
select User, password,plugin FROM mysql.user ;
```

Aktualisieren Sie die Benutzer (Plugin null aktiviert für alle Plugins):

```
update mysql.user set password=PASSWORD('mypassword'), plugin = NULL WHERE User = 'root';
exit;
```

Stoppen Sie in der Unix-Shell mySQL ohne Grant-Tabellen und starten Sie dann mit Grant-Tabellen erneut:

```
sudo service mysql stop
sudo service mysql start
```

Stellen Sie das verlorene Root-Passwort wieder her online lesen:
https://riptutorial.com/de/mysql/topic/9973/stellen-sie-das-verlorene-root-passwort-wieder-her

Kapitel 58: Tabelle ablegen

Syntax

- DROP TABLE Tabellenname;
- DROP TABLE IF EXISTS Tabellenname; - um lästige Fehler in automatisierten Skripts zu vermeiden
- DROP TABELLE t1, t2, t3; - DROP mehrere Tabellen
- TROPFÄHNIGE TABELLE t; - DROP eine Tabelle von CREATE TEMPORARY TABLE ...

Parameter

Parameter	Einzelheiten
TEMPORARY	Wahlweise. Es gibt an, dass nur temporäre Tabellen von der DROP TABLE-Anweisung gelöscht werden sollen.
WENN EXISTS	Wahlweise. Wenn angegeben, gibt die DROP TABLE-Anweisung keinen Fehler aus, wenn keine der Tabellen vorhanden ist.

Examples

Tabelle ablegen

Tabelle löschen wird zum Löschen der Tabelle aus der Datenbank verwendet.

Tabelle erstellen:

Erstellen einer Tabelle mit dem Namen tbl und Löschen der erstellten Tabelle

```
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)
);
```

Ablagetisch:

```
DROP TABLE tbl;
```

> **BITTE BEACHTEN SIE**
>
> Durch das Löschen der Tabelle werden die Tabelle und alle Informationen vollständig

aus der Datenbank gelöscht und nicht wiederhergestellt.

Tabellen aus der Datenbank löschen

DROP TABLE Datenbank.Tabellenname

Tabelle ablegen online lesen: https://riptutorial.com/de/mysql/topic/4123/tabelle-ablegen

Kapitel 59: Tabellenerstellung

Syntax

- CREATE TABLE Tabellenname (Spaltenname1 Datentyp (Größe), Spaltenname2 Datentyp (Größe), Spaltenname3 Datentyp (Größe), ...); // Grundlegende Tabellenerstellung
- CREATE TABLE Tabellenname [IF NOT EXISTS] (Spaltenname1 Datentyp (Größe), Spaltenname2 Datentyp (Größe), Spaltenname3 Datentyp (Größe), ...); // Überprüfung der Tabellenerstellung vorhanden
- CREATE [TEMPORARY] TABLE tabellenname [IF NOT EXISTS] (Spaltenname1 Datentyp (Größe), Spaltenname2 Datentyp (Größe), Spaltenname3 Datentyp (Größe) ... // Temporäre Tabellenerstellung
- CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl; // Tabellenerstellung aus SELECT

Bemerkungen

Die `CREATE TABLE` Anweisung sollte mit einer `ENGINE` Spezifikation enden:

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

Einige Optionen sind:

- `InnoDB` : (Standard seit Version 5.5.5) Es ist eine transationssichere (ACID-kompatible) Engine. Es verfügt über Transaktions-Commit und -Rollback, Crash-Recovery-Funktionen und Sperren auf Zeilenebene.
- `MyISAM` : (Standard vor Version 5.5.5) Es ist eine einfache Engine. Es unterstützt keine Transaktionen oder Fremdschlüssel, aber es ist nützlich für Data Warehousing.
- `Memory` : Speichert alle Daten im RAM für extrem schnelle Vorgänge, das Datum der Tabelle geht jedoch beim Neustart der Datenbank verloren.

Weitere Motoroptionen hier .

Examples

Grundlegende Tabellenerstellung

Die `CREATE TABLE` Anweisung wird zum Erstellen einer Tabelle in einer MySQL-Datenbank verwendet.

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

```
    `Address`       TEXT,
    `City`          VARCHAR(100)
) Engine=InnoDB;
```

Jede Felddefinition muss Folgendes aufweisen:

1. Feldname: Ein gültiger Feldname. Stellen Sie sicher, dass die Namen in `-chars eingebettet sind. Dadurch wird sichergestellt, dass Sie im Feldnamen zB Leerzeichen verwenden können.
2. Datentyp [Länge]: Wenn das Feld `CHAR` oder `VARCHAR` ist, muss eine Feldlänge angegeben werden.
3. Attribute `NULL` | `NOT NULL` : Wenn `NOT NULL` angegeben ist, schlägt der Versuch eines `NULL` Werts in diesem Feld fehl.
4. Weitere Informationen zu Datentypen und ihren Attributen finden Sie hier .

`Engine=...` ist ein optionaler Parameter zur Angabe der Speicher-Engine der Tabelle. Wenn keine Speicher-Engine angegeben ist, wird die Tabelle mit der standardmäßigen Tabellen-Speicher-Engine des Servers (normalerweise InnoDB oder MyISAM) erstellt.

Voreinstellungen festlegen

`DEFAULT` es sinnvoll ist, können Sie außerdem mit `DEFAULT` einen Standardwert für jedes Feld `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;
```

Wenn beim Einfügen keine `Street` angegeben wird, ist dieses Feld beim Abrufen `NULL` . Wenn beim Einfügen kein `Country` angegeben wird, wird standardmäßig "USA" verwendet.

Sie können Standardwerte für alle Spaltentypen, stellen mit Ausnahme für `BLOB` , `TEXT` , `GEOMETRY` und `JSON` Felder aus .

Tabellenerstellung mit Primärschlüssel

```
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)
);
```

Ein **Primärschlüssel** ist ein ein- oder mehrspaltiger `NOT NULL` -Bezeichner, der eine Zeile einer Tabelle eindeutig identifiziert. Ein Index wird erstellt, und wenn MySQL nicht explizit als `NOT NULL` deklariert wird, werden sie von MySQL so still und implizit deklariert.

Eine Tabelle kann nur einen `PRIMARY KEY` , und für jede Tabelle wird empfohlen, einen zu haben. InnoDB erstellt in seiner Abwesenheit automatisch eine (wie in der MySQL-Dokumentation zu sehen), obwohl dies weniger wünschenswert ist.

Häufig wird ein `AUTO_INCREMENT INT` auch als "Ersatzschlüssel" bezeichnet, für die Optimierung des dünnen Index und für Beziehungen zu anderen Tabellen verwendet. Dieser Wert wird (normalerweise) um 1 erhöht, wenn ein neuer Datensatz hinzugefügt wird. Der Standardwert ist 1.

Trotz des Namens ist es jedoch nicht das Ziel, zu gewährleisten, dass die Werte inkrementell sind, sondern lediglich, dass sie sequentiell und eindeutig sind.

Ein `INT` Wert für die automatische Erhöhung wird nicht auf den Standardstartwert zurückgesetzt, wenn alle Zeilen in der Tabelle gelöscht werden, es sei denn, die Tabelle wird mit der `TRUNCATE TABLE` abgeschnitten.

Definieren einer Spalte als Primärschlüssel (Inline-Definition)

Wenn der Primärschlüssel aus einer einzelnen Spalte besteht, kann die `PRIMARY KEY` Klausel mit der `PRIMARY KEY` inline platziert werden:

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

Diese Befehlsform ist kürzer und leichter lesbar.

Definieren eines mehrspaltigen Primärschlüssels

Es ist auch möglich, einen Primärschlüssel mit mehr als einer Spalte zu definieren. Dies kann beispielsweise in der untergeordneten Tabelle einer Fremdschlüsselbeziehung erfolgen. Ein mehrspaltiger Primärschlüssel wird durch Auflisten der beteiligten Spalten in einer separaten `PRIMARY KEY` Klausel definiert. Inline-Syntax ist hier nicht zulässig, da nur eine Spalte als `PRIMARY KEY` inline deklariert werden kann. Zum Beispiel:

```
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
);
```

Beachten Sie, dass die Spalten des Primärschlüssels *sollte* in logische Sortierreihenfolge festgelegt werden, die von der Reihenfolge verschieden sein *können* , in denen die Säulen definiert sind , wie in dem obigen Beispiel.

Größere Indizes erfordern mehr Speicherplatz, Speicher und E / A. Daher sollten Schlüssel möglichst klein sein (insbesondere bei zusammengesetzten Schlüsseln). In InnoDB enthält jeder "Sekundärindex" eine Kopie der Spalten des `PRIMARY KEY` .

Tabellenerstellung mit Fremdschlüssel

```
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;
```

Fremdschlüssel: Ein **Fremdschlüssel** (`FK`) ist entweder eine einzelne Spalte oder eine mehrspaltige Zusammenstellung von Spalten in einer *Verweistabelle* . Es wurde bestätigt, dass diese `FK` in der *referenzierten* Tabelle vorhanden ist. Es wird dringend empfohlen, dass der *referenzierte* Tabellenschlüssel, der den `FK` bestätigt, ein Primärschlüssel ist, der jedoch nicht erzwungen wird. Es wird als schnelle Suche in *Bezug* auf die *referenzierten* Objekte verwendet, wo es nicht eindeutig sein muss. In der Tat kann es sich um einen Index ganz links handeln.

Fremdschlüsselbeziehungen umfassen eine übergeordnete Tabelle, die die zentralen Datenwerte enthält, und eine untergeordnete Tabelle mit identischen Werten, die auf ihr übergeordnetes Element verweisen. Die FOREIGN KEY-Klausel wird in der untergeordneten Tabelle angegeben. Die übergeordneten und untergeordneten Tabellen müssen dieselbe Speicher-Engine verwenden. Sie dürfen keine temporären Tabellen sein.

Entsprechende Spalten im Fremdschlüssel und der referenzierte Schlüssel müssen ähnliche Datentypen haben. Die Größe und das Vorzeichen von Integer-Typen müssen gleich sein. Die Länge der Zeichenfolgentypen muss nicht gleich sein. Für nicht-binäre (Zeichen-) Zeichenfolgenspalten müssen der Zeichensatz und die Sortierung identisch sein.

Hinweis: Fremdschlüsseleinschränkungen werden von der InnoDB-Speicher-Engine unterstützt (nicht MyISAM oder MEMORY). DB-Setups, die andere Engines verwenden, akzeptieren diese `CREATE TABLE` Anweisung, `CREATE TABLE` jedoch keine Fremdschlüsseleinschränkungen. (Obwohl neuere MySQL-Versionen standardmäßig `InnoDB` , ist es jedoch `InnoDB` , explizit zu sein.)

Eine vorhandene Tabelle klonen

Eine Tabelle kann wie folgt repliziert werden:

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

Die neue Tabelle hat genau dieselbe Struktur wie die ursprüngliche Tabelle, einschließlich Indizes und Spaltenattributen.

Neben dem manuellen Erstellen einer Tabelle können Sie auch eine Tabelle erstellen, indem Sie Daten aus einer anderen Tabelle auswählen:

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

Sie können die normalen Funktionen einer `SELECT` Anweisung verwenden, um die Daten während des `SELECT` zu ändern:

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

Primärschlüssel und Indizes bleiben beim Erstellen von Tabellen aus `SELECT` nicht erhalten. Sie müssen sie erneut deklarieren:

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

CREATE TABLE VON SELECT

Sie können eine Tabelle aus einer anderen erstellen, indem Sie am Ende der `CREATE TABLE` Anweisung eine `SELECT` Anweisung hinzufügen:

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

Erstellen Sie eine Tabelle in derselben Datenbank:

```
-- 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;
```

Erstellen Sie Tabellen aus verschiedenen Datenbanken:

```
-- 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

Um eine Tabelle einer anderen Tabelle zu erstellen, die in einer anderen Datenbank vorhanden ist, müssen Sie den Namen der Datenbank folgendermaßen angeben:

```
FROM NAME_DATABASE.name_table
```

Tabellenstruktur anzeigen

Wenn Sie die Schemainformationen Ihrer Tabelle anzeigen möchten, können Sie eine der folgenden Möglichkeiten verwenden:

```
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;
```

Wenn es vom mysql-Befehlszeilentool verwendet wird, ist dies weniger ausführlich:

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

Eine weniger beschreibende Art, die Tabellenstruktur darzustellen:

```
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    |       |
+-------+-------------+------+-----+---------+-------+
```

Sowohl **DESCRIBE** als auch **DESC** liefern das gleiche Ergebnis.

`DESCRIBE` wie `DESCRIBE` für alle Tabellen in einer Datenbank gleichzeitig ausgeführt wird, finden Sie in diesem Beispiel .

Tabelle mit Zeitstempelspalte erstellen, um die letzte Aktualisierung

anzuzeigen

Die TIMESTAMP-Spalte zeigt an, wann die Zeile zuletzt aktualisiert wurde.

```
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'
;
```

Tabellenerstellung online lesen: https://riptutorial.com/de/mysql/topic/795/tabellenerstellung

Kapitel 60: Temporäre Tabellen

Examples

Temporäre Tabelle erstellen

Temporäre Tabellen können sehr nützlich sein, um temporäre Daten zu speichern. Die Option für temporäre Tabellen ist in MySQL-Version 3.23 und höher verfügbar.

Die temporäre Tabelle wird automatisch gelöscht, wenn die Sitzung endet oder die Verbindung geschlossen wird. Der Benutzer kann auch eine temporäre Tabelle löschen.

Der Name der temporären Tabelle kann in vielen Verbindungen gleichzeitig verwendet werden, da die temporäre Tabelle nur für den Client verfügbar ist, der diese Tabelle erstellt.

Die temporäre Tabelle kann in den folgenden Typen erstellt werden

```
--->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;
```

Sie können beim Erstellen der Tabelle Indizes hinzufügen:

```
CREATE TEMPORARY TABLE tempTable1
        ( PRIMARY KEY(ColumnName2) )
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

`IF NOT EXISTS` Schlüsselwort wie unten angegeben verwendet werden kann, um Fehler in der *Tabelle bereits vorhanden* zu vermeiden. In diesem Fall wird jedoch keine Tabelle erstellt, wenn der von Ihnen verwendete Tabellenname in Ihrer aktuellen Sitzung bereits vorhanden ist.

```
CREATE TEMPORARY TABLE IF NOT EXISTS tempTable1
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

Temporäre Tabelle löschen

Temporäre Tabelle löschen wird verwendet, um die temporäre Tabelle zu löschen, die Sie in Ihrer aktuellen Sitzung erstellt haben.

```
DROP TEMPORARY TABLE tempTable1
```

```
DROP TEMPORARY TABLE IF EXISTS tempTable1
```

Verwenden Sie `IF EXISTS` , um zu verhindern, dass ein Fehler für Tabellen auftritt, die möglicherweise nicht vorhanden sind

Temporäre Tabellen online lesen: https://riptutorial.com/de/mysql/topic/5757/temporare-tabellen

Kapitel 61: Transaktion

Examples

Starten Sie die Transaktion

Eine Transaktion ist eine sequentielle Gruppe von SQL-Anweisungen wie select, insert, update oder delete, die als eine einzige Arbeitseinheit ausgeführt wird.

Mit anderen Worten, eine Transaktion ist niemals abgeschlossen, es sei denn, jede einzelne Operation in der Gruppe ist erfolgreich. Wenn eine Operation innerhalb der Transaktion fehlschlägt, schlägt die gesamte Transaktion fehl.

Die Banktransaktion ist das beste Beispiel, um dies zu erklären. Betrachten Sie eine Übertragung zwischen zwei Konten. Um dies zu erreichen, müssen Sie SQL-Anweisungen schreiben, die Folgendes tun

1. Überprüfen Sie die Verfügbarkeit des angeforderten Betrags im ersten Konto
2. Den angeforderten Betrag vom ersten Konto abziehen
3. Zahlen Sie es auf das zweite Konto ein

Wenn einer dieser Prozesse fehlschlägt, sollte das Ganze in den vorherigen Zustand zurückgesetzt werden.

ACID: Eigenschaften von Transaktionen

Transaktionen haben die folgenden vier Standardeigenschaften

- **Atomizität:** stellt sicher, dass alle Operationen innerhalb der Arbeitseinheit erfolgreich abgeschlossen werden; Andernfalls wird die Transaktion zum Zeitpunkt des Scheiterns abgebrochen, und vorherige Vorgänge werden in ihren früheren Zustand zurückgesetzt.
- **Konsistenz: Stellt** sicher, dass die Datenbank den Status bei einer erfolgreich abgeschlossenen Transaktion ordnungsgemäß ändert.
- **Isolation:** Ermöglicht Transaktionen unabhängig voneinander und transparent zu sein.
- **Dauerhaftigkeit: Stellt** sicher, dass das Ergebnis oder die Wirkung einer festgeschriebenen Transaktion bei einem Systemausfall erhalten bleibt.

Transaktionen beginnen mit der Anweisung `START TRANSACTION` oder `BEGIN WORK` und enden entweder mit einer `COMMIT` oder einer `ROLLBACK` Anweisung. Die SQL-Befehle zwischen der Anfangs- und der Endanweisung bilden den Großteil der Transaktion.

```
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;
```

Mit `START TRANSACTION` bleibt Autocommit deaktiviert, bis Sie die Transaktion mit `COMMIT` oder `ROLLBACK` beenden. Der Autocommit-Modus kehrt dann in den vorherigen Zustand zurück.

Das `FOR UPDATE` zeigt (und sperrt) die Zeile (n) für die Dauer der Transaktion.

Während die Transaktion nicht festgeschrieben ist, steht diese Transaktion für andere Benutzer nicht zur Verfügung.

Allgemeine an der Transaktion beteiligte Verfahren

- Beginnen Sie die Transaktion mit dem SQL-Befehl `BEGIN WORK` oder `START TRANSACTION` .
- Führen Sie alle Ihre SQL-Anweisungen aus.
- Prüfen Sie, ob alles Ihren Anforderungen entspricht.
- Wenn ja, geben Sie den Befehl `COMMIT` . Andernfalls geben Sie den Befehl ROLLBACK aus, um den vorherigen Zustand `ROLLBACK` .
- Prüfen Sie auch nach `COMMIT` auf Fehler, wenn Sie Galera / PXC verwenden oder möglicherweise verwenden.

COMMIT, ROLLBACK und AUTOCOMMIT

AUTOCOMMIT

MySQL schreibt automatisch Anweisungen ein, die nicht Teil einer Transaktion sind. Die Ergebnisse von `UPDATE` , `DELETE` oder `INSERT` Anweisungen, denen keine `BEGIN` oder `START TRANSACTION` Anweisung vorangestellt ist, sind sofort für alle Verbindungen sichtbar.

Die Variable `AUTOCOMMIT` ist standardmäßig auf *true* gesetzt. Dies kann auf folgende Weise geändert werden:

```
--->To make autcommit false
SET AUTOCOMMIT=false;
--or
SET AUTOCOMMIT=0;

--->To make autcommit true
SET AUTOCOMMIT=true;
--or
SET AUTOCOMMIT=1;
```

Zum Anzeigen des `AUTOCOMMIT` Status

```
SELECT @@autocommit;
```

VERPFLICHTEN

Wenn `AUTOCOMMIT` auf false gesetzt ist und die Transaktion nicht festgeschrieben ist, sind die Änderungen nur für die aktuelle Verbindung sichtbar.

Nachdem die `COMMIT` Anweisung die Änderungen an der Tabelle `COMMIT` , ist das Ergebnis für alle Verbindungen sichtbar.

Wir betrachten zwei Verbindungen, um dies zu erklären

Verbindung 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 |
+-----+
```

Verbindung 2

```
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
+-----+
---> Row inserted before autocommit=false only visible here
```

Verbindung 1

```
mysql> COMMIT;
--->Now COMMIT is executed in connection 1
mysql> SELECT * FROM testTable;
    +-----+
    | tId |
    +-----+
    |   1 |
    |   2 |
    |   3 |
    +-----+
```

Verbindung 2

```
mysql> SELECT * FROM testTable;
    +-----+
    | tId |
    +-----+
    |   1 |
    |   2 |
    |   3 |
    +-----+
--->Now all the three rows are visible here
```

ROLLBACK

Wenn bei der Abfrageausführung ein `ROLLBACK` aufgetreten ist, wurden die Änderungen mit `ROLLBACK` in rückgängig gemacht. Siehe die Erklärung unten

```
--->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 |
+-----+
```

Jetzt führen wir `ROLLBACK`

```
--->Rollback executed now
mysql> ROLLBACk;

mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
+-----+
--->Rollback removed all rows which all are not committed
```

Sobald `COMMIT` ausgeführt wird, verursacht `ROLLBACK` nichts

```
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
```

Wenn `AUTOCOMMIT` auf *true gesetzt ist* , sind `COMMIT` und `ROLLBACK` unbrauchbar

Transaktion mit JDBC-Treiber

Transaktion mit JDBC-Treiber wird verwendet, um zu steuern, wie und wann eine Transaktion festgeschrieben und zurückgesetzt werden soll. Die Verbindung zum MySQL-Server wird mithilfe des JDBC-Treibers hergestellt

Der JDBC-Treiber für MySQL kann hier heruntergeladen werden

Beginnen wir mit dem Herstellen einer Verbindung zur Datenbank mithilfe des JDBC-Treibers

```
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");
```

Zeichensätze : Gibt an, welchen Zeichensatz der Client zum Senden von SQL-Anweisungen an den Server verwendet. Es gibt außerdem den Zeichensatz an, den der Server zum Senden von Ergebnissen an den Client verwenden soll.

Dies sollte beim Herstellen der Verbindung zum Server erwähnt werden. Die Verbindungszeichenfolge sollte also lauten:

```
jdbc:mysql://localhost:3306/testDB?useUnicode=true&amp;characterEncoding=utf8
```

Weitere Informationen zu Zeichensätzen und Kollatierungen finden Sie hier

Wenn Sie offene Verbindung, die `AUTOCOMMIT` - Modus standardmäßig auf *true* gesetzt ist, werden das sollte *falsch* geändert Transaktion zu starten.

```
con.setAutoCommit(false);
```

Sie sollten die Methode `setAutoCommit()` immer direkt nach dem Öffnen einer Verbindung aufrufen.

Verwenden Sie andernfalls `START TRANSACTION` oder `BEGIN WORK` , um eine neue Transaktion zu starten. Wenn Sie `START TRANSACTION` oder `BEGIN WORK` , müssen Sie `AUTOCOMMIT` *false* nicht ändern. Das wird automatisch deaktiviert.

Jetzt können Sie mit der Transaktion beginnen. Nachfolgend finden Sie ein vollständiges Beispiel für eine JDBC-Transaktion.

```
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);
    }
}
```

Die JDBC-Transaktion stellt sicher, dass alle SQL-Anweisungen innerhalb eines Transaktionsblocks erfolgreich ausgeführt werden. Wenn eine der SQL-Anweisungen innerhalb des Transaktionsblocks fehlgeschlagen ist, wird der gesamte Transaktionsblock abgebrochen und zurückgesetzt.

Transaktion online lesen: https://riptutorial.com/de/mysql/topic/5771/transaktion

Kapitel 62: Umgang mit spärlichen oder fehlenden Daten

Examples

Mit Spalten arbeiten, die NULL-Werte enthalten

In MySQL und anderen SQL-Dialekten haben `NULL` Werte besondere Eigenschaften.

Betrachten Sie die folgende Tabelle mit den Bewerbern, den Unternehmen, für die sie gearbeitet haben, und dem Datum, an dem sie das Unternehmen verlassen haben. `NULL` bedeutet, dass ein Bewerber noch im Unternehmen arbeitet:

```
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 |
+--------------+-----------------+------------+
```

Ihre Aufgabe ist es, eine Abfrage zu `2016-01-01` , in der alle Zeilen nach dem `2016-01-01` , einschließlich aller Mitarbeiter, die noch in einem Unternehmen arbeiten (Mitarbeiter mit `NULL` Enddatum). Diese Select-Anweisung:

```
SELECT * FROM example WHERE end_date > '2016-01-01';
```

schlägt fehl, Zeilen mit `NULL` Werten aufzunehmen:

```
+--------------+-----------------+------------+
| applicant_id | company_name    | end_date   |
+--------------+-----------------+------------+
|            2 | Woodworking.com | 2016-08-25 |
+--------------+-----------------+------------+
```

Gemäß der MySQL-Dokumentation geben Vergleiche mit den arithmetischen Operatoren <,>, = und <> selbst `NULL` anstelle eines booleschen `TRUE` oder `FALSE` . Eine Zeile mit einem `NULL` end_date ist daher weder größer als 2016-01-01 oder kleiner als 2016-01-01.

Dies kann mit den Schlüsselwörtern IS NULL gelöst werden:

```
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 |
+--------------+-----------------+------------+
```

Das Arbeiten mit NULL-Werten wird komplexer, wenn die Aufgabe Aggregationsfunktionen wie `MAX()` und eine `GROUP BY` Klausel beinhaltet. Wenn Ihre Aufgabe darin bestand, das letzte verwendete Datum für jede Bewerber-ID auszuwählen, erscheint die folgende Abfrage als logischer erster Versuch:

```
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    |
+--------------+---------------+
```

In dem Wissen, dass `NULL` bedeutet, dass ein Bewerber noch in einem Unternehmen beschäftigt ist, ist die erste Zeile des Ergebnisses ungenau. `CASE WHEN` bietet eine Problemumgehung für das `NULL` Problem:

```
 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 |
+--------------+------------+
```

Dieses Ergebnis kann mit der ursprünglichen `example` werden, um die Firma zu bestimmen, bei der ein Bewerber zuletzt gearbeitet hat:

```
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 |
+--------------+-----------------+------------+
```

Dies sind nur einige Beispiele für die Arbeit mit `NULL` Werten in MySQL.

Umgang mit spärlichen oder fehlenden Daten online lesen:
https://riptutorial.com/de/mysql/topic/5866/umgang-mit-sparlichen-oder-fehlenden-daten

Kapitel 63: Umgang mit Zeitzonen

Bemerkungen

Wenn Sie Zeitinformationen für eine weltweite Benutzerbasis in MySQL verarbeiten müssen, verwenden Sie den TIMESTAMP-Datentyp in Ihren Tabellen.

Speichern Sie für jeden Benutzer eine Timezone-Spalte mit Benutzereinstellungen. VARCHAR (64) ist ein guter Datentyp für diese Spalte. Wenn sich ein Benutzer registriert, um Ihr System zu verwenden, fragen Sie nach dem Zeitzonenwert. Mein ist Atlantic Time, `America/Edmonton` . Ihr könnte `Asia/Kolkata` oder `Australia/NSW` sein oder nicht. Für eine Benutzeroberfläche für diese Benutzereinstellung bietet die WordPress.org-Software ein gutes Beispiel.

Wann immer Sie eine Verbindung von Ihrem Host-Programm (Java, PHP, was auch immer) zu Ihrem DBMS für einen Benutzer herstellen, geben Sie den SQL-Befehl aus

```
SET SESSION time_zone='(whatever tz string the user gave you)'
```

bevor Sie mit Benutzerdaten umgehen, die zeitaufwendig sind. Dann werden alle `TIMESTAMP` Zeiten, die Sie installiert haben, in der Ortszeit des Benutzers `TIMESTAMP` .

Dies führt dazu, dass alle Zeitangaben in Ihren Tabellen in UTC konvertiert werden und alle Zeitangaben in lokal übersetzt werden. Es funktioniert einwandfrei für JETZT () und CURDATE (). Auch hier müssen Sie die Datentypen TIMESTAMP und nicht DATETIME oder DATE verwenden.

Stellen Sie sicher, dass das Server-Betriebssystem und die Standard-MySQL-Zeitzonen auf UTC eingestellt sind. Wenn Sie dies nicht tun, bevor Sie mit dem Laden von Informationen in Ihre Datenbank beginnen, ist es fast unmöglich, das Problem zu beheben. Wenn Sie einen Anbieter zum Ausführen von MySQL verwenden, bestehen Sie darauf, dass diese das Recht haben.

Examples

Abrufen des aktuellen Datums und der Uhrzeit in einer bestimmten Zeitzone.

Hiermit wird der Wert von `NOW()` in der Ortszeit, in der indischen Standardzeit und dann erneut in UTC abgerufen.

```
SELECT NOW();
SET time_zone='Asia/Kolkata';
SELECT NOW();
SET time_zone='UTC';
SELECT NOW();
```

Konvertieren Sie einen gespeicherten Wert für 'DATE' oder 'DATETIME' in eine andere Zeitzone.

Wenn Sie ein `DATE` oder `DATETIME` (in einer Spalte irgendwo) gespeichert haben, wurde es in Bezug auf eine Zeitzone gespeichert, aber in MySQL wird die Zeitzone *nicht* mit dem Wert gespeichert. Wenn Sie es in eine andere Zeitzone konvertieren möchten, können Sie dies tun, aber Sie müssen die ursprüngliche Zeitzone kennen. Mit `CONVERT_TZ()` wird die Konvertierung durchgeführt. Dieses Beispiel zeigt Zeilen, die in Kalifornien in Ortszeit verkauft wurden.

```
SELECT CONVERT_TZ(date_sold,'UTC','America/Los_Angeles') date_sold_local
  FROM sales
 WHERE state_sold = 'CA'
```

Abrufen von gespeicherten TIMESTAMP-Werten in einer bestimmten Zeitzone

Das ist sehr einfach. Alle `TIMESTAMP` Werte werden in Universalzeit gespeichert und bei jeder `TIMESTAMP` immer in die `time_zone` Einstellung `time_zone` konvertiert.

```
SET SESSION time_zone='America/Los_Angeles';
SELECT timestamp_sold
  FROM sales
 WHERE state_sold = 'CA'
```

Warum ist das? `TIMESTAMP` Werte basieren auf dem ehrwürdigen UNIX `time_t` Datentyp . Diese UNIX-Zeitmarken werden seit `1970-01-01 00:00:00` UTC in Sekunden `1970-01-01 00:00:00` .

Hinweis `TIMESTAMP` Werte werden in `TIMESTAMP` gespeichert. `DATE` und `DATETIME` Werte werden gespeichert in welcher Ortszeit in Kraft war , als sie gespeichert wurden.

Wie ist die lokale Zeitzoneneinstellung meines Servers?

Jeder Server verfügt über eine globale Einstellung von time_zone, die vom Eigentümer der Server-Maschine konfiguriert wird. Sie können die aktuelle Zeitzoneneinstellung auf folgende Weise ermitteln:

```
SELECT @@time_zone
```

Leider ergibt dies normalerweise den Wert `SYSTEM` , dh die MySQL-Zeit wird durch die Zeitzoneneinstellung des Server-Betriebssystems bestimmt.

Diese Abfolge von Abfragen (ja, es ist ein Hack) gibt Ihnen den Versatz in Minuten zwischen der Zeitzoneneinstellung des Servers und UTC zurück.

```
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;
```

Wie funktioniert das? Die zwei Spalten in der temporären Tabelle mit unterschiedlichen Datentypen sind der Hinweis. `DATETIME` -Datentypen werden immer in Ortszeit in Tabellen und

`TIMESTAMP` in UTC gespeichert. Die `INSERT` Anweisung, die ausgeführt wird, wenn time_zone auf UTC gesetzt ist, speichert also zwei identische Datums- / Uhrzeitwerte.

Dann wird die SELECT-Anweisung ausgeführt, wenn time_zone auf Serverzeit eingestellt ist. `TIMESTAMP` werden in SELECT-Anweisungen immer aus ihrer gespeicherten UTC-Form in die Ortszeit übersetzt. `DATETIME` sind nicht. Die `TIMESTAMPDIFF(MINUTE...)` berechnet also die Differenz zwischen Ortszeit und `TIMESTAMPDIFF(MINUTE...)` .

Welche time_zone-Werte sind auf meinem Server verfügbar?

Verwenden Sie diesen Befehl, um eine Liste der möglichen time_zone-Werte in Ihrer MySQL-Serverinstanz abzurufen.

```
SELECT mysql.time_zone_name.name
```

Normalerweise wird hier die ZoneInfo-Liste der Zeitzonen angezeigt, die von Paul Eggert bei der Internet Assigned Numbers Authority verwaltet wird . Weltweit gibt es ca. 600 Zeitzonen.

Unix-ähnliche Betriebssysteme (z. B. Linux-Distributionen, BSD-Distributionen und moderne Mac OS-Distributionen) erhalten Routineupdates. Durch die Installation dieser Updates auf einem Betriebssystem können die dort ausgeführten MySQL-Instanzen die Änderungen der Zeitzonen sowie der Umstellung auf Tageslicht / Standardzeit nachverfolgen.

Wenn Sie eine viel kürzere Liste von Zeitzonennamen erhalten, ist Ihr Server entweder unvollständig konfiguriert oder läuft unter Windows. Hier finden Sie Anweisungen für Ihren Serveradministrator zum Installieren und Verwalten der ZoneInfo-Liste.

Umgang mit Zeitzonen online lesen: https://riptutorial.com/de/mysql/topic/7849/umgang-mit-zeitzonen

Kapitel 64: UNION

Syntax

- UNION DISTINCT - Dedups nach dem Kombinieren der SELECTs
- UNION ALL - nicht deduziert (schneller)
- UNION - Die Standardeinstellung ist DISTINCT
- SELECT ... UNION SELECT ... - ist OK, aber bei `ORDER BY` mehrdeutig
- (SELECT ...) UNION (SELECT ...) ORDER BY ... - löst die Mehrdeutigkeit auf

Bemerkungen

UNION verwendet nicht mehrere CPUs.

UNION * beinhaltet immer eine temporäre Tabelle, um die Ergebnisse zu sammeln. * Ab 5.7.3 / MariaDB 10.1 liefern einige Formen von UNION die Ergebnisse ohne Verwendung einer tmp-Tabelle (daher schneller).

Examples

SELECT-Anweisungen mit UNION kombinieren

Sie können die Ergebnisse von zwei identisch strukturierten Abfragen mit dem Schlüsselwort `UNION` kombinieren.

Wenn Sie beispielsweise eine Liste aller Kontaktinformationen aus zwei separaten Tabellen, `authors` und `editors` möchten, können Sie das `UNION` Schlüsselwort wie folgt verwenden:

```
select name, email, phone_number
from authors

union

select name, email, phone_number
from editors
```

Durch die Verwendung von `union` werden Duplikate entfernt. Wenn Sie in Ihrer Abfrage Duplikate beibehalten müssen, können Sie das `ALL` Schlüsselwort wie `UNION ALL` : `UNION ALL` .

SORTIEREN NACH

Wenn Sie die Ergebnisse einer UNION sortieren müssen, verwenden Sie dieses Muster:

```
( SELECT ... )
UNION
( SELECT ... )
```

```
ORDER BY
```

Ohne die Klammern würde das letzte ORDER BY zum letzten SELECT gehören.

Paginierung über OFFSET

Beim Hinzufügen eines LIMIT zu einer UNION ist dies das zu verwendende Muster:

```
( SELECT ... ORDER BY x  LIMIT 10 )
UNION
( SELECT ... ORDER BY x  LIMIT 10 )
ORDER BY x  LIMIT 10
```

Da Sie nicht vorhersagen können, aus welchen SELECT (s) die "10" kommt, müssen Sie jeweils 10 davon erhalten und dann die Liste weiter unten durchlaufen, wobei sowohl `ORDER BY` als auch `LIMIT` wiederholt werden.

Für die 4. Seite von 10 Elementen wird dieses Muster benötigt:

```
( SELECT ... ORDER BY x  LIMIT 40 )
UNION
( SELECT ... ORDER BY x  LIMIT 40 )
ORDER BY x  LIMIT 30, 10
```

Das heißt, sammeln Sie in jeder `SELECT` 4 Seitenwerte und führen Sie dann `OFFSET` in der `UNION` .

Daten mit unterschiedlichen Spalten kombinieren

```
SELECT name, caption as title, year, pages FROM books
UNION
SELECT name, title, year, 0 as pages FROM movies
```

Wenn Sie zwei Datensätze mit unterschiedlichen Spalten kombinieren, dann emulieren Sie die fehlenden mit Standardwerten.

UNION ALL und UNION

SELECT 1,22,44 UNION SELECT 2,33,55

信息 结果1 概况 状态

1	22	44
1	22	44
2	33	55

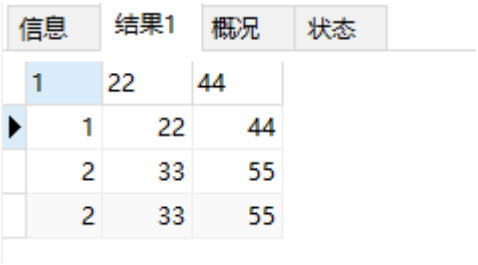
SELECT 1,22,44 UNION SELECT 2,33,55 UNION SELECT 2,33,55

Das Ergebnis ist das gleiche wie oben.

Verwenden Sie UNION ALL

wann

SELECT 1,22,44 UNION SELECT 2,33,55 UNION ALL SELECT 2,33,55



1	22	44
1	22	44
2	33	55
2	33	55

Kombinieren und Kombinieren von Daten in verschiedenen MySQL-Tabellen mit denselben Spalten in eindeutigen Zeilen und Ausführen der Abfrage

Dieses ***UNION ALL-System*** kombiniert Daten aus mehreren Tabellen und dient als Tabellennamensalias für Ihre Abfragen:

```
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);
```

UNION online lesen: https://riptutorial.com/de/mysql/topic/3847/union

Kapitel 65: Veranstaltungen

Examples

Erstellen Sie ein Ereignis

Mysql verfügt über die EVENT-Funktion zum Vermeiden komplizierter Cron-Interaktionen, wenn ein Großteil Ihrer Zeitplanung auf SQL und weniger auf Dateien bezogen ist. Siehe die Handbuchseite hier . Stellen Sie sich Ereignisse als gespeicherte Prozeduren vor, deren Ausführung in regelmäßigen Abständen geplant ist.

Um beim Debuggen von ereignisbezogenen Problemen Zeit zu sparen, müssen Sie berücksichtigen, dass der globale Ereignishandler aktiviert sein muss, um Ereignisse zu verarbeiten.

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

Bei AUS wird nichts ausgelöst. Also einschalten:

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

Schema zum Testen

```
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');
```

Die obigen Einsätze dienen als Ausgangspunkt. Beachten Sie, dass die unten erstellten 2 Ereignisse die Zeilen bereinigen.

Erstellen Sie 2 Ereignisse, 1. Läufe täglich, 2. Läufe alle 10 Minuten

Ignoriere, was sie tatsächlich tun (gegeneinander spielen). Der Punkt liegt auf dem INTERVAL und der Terminplanung.

```
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 ;
```

Ereignisstatus anzeigen (unterschiedliche Ansätze)

```
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)
```

Zufälliges Zeug zum Nachdenken

`DROP EVENT someEventName;` - Löscht das Ereignis und seinen Code

`ON COMPLETION PRESERVE` - Wenn das Ereignis abgeschlossen ist, behalten Sie es bei. Ansonsten wird es gelöscht.

Ereignisse sind wie Auslöser. Sie werden nicht von einem Benutzerprogramm aufgerufen. Sie sind vielmehr geplant. Als solche sind sie erfolgreich oder scheitern ohne Erfolg.

Der Link zur Manual Page zeigt einige Flexibilität mit Intervallauswahlmöglichkeiten (siehe unten):

> Intervall:
>
> ```
> quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
>           WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
>           DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
> ```

Ereignisse sind leistungsfähige Mechanismen, mit denen wiederkehrende und geplante Aufgaben für Ihr System ausgeführt werden. Sie können beliebig viele Anweisungen, DDL- und DML-Routinen sowie komplizierte Verknüpfungen enthalten. Weitere Informationen finden Sie auf der MySQL-Handbuchseite unter Einschränkungen für gespeicherte Programme .

Veranstaltungen online lesen: https://riptutorial.com/de/mysql/topic/4319/veranstaltungen

Kapitel 66: Verbindung mit UTF-8 unter Verwendung verschiedener Programmiersprachen.

Examples

Python

1. oder 2. Zeile im Quellcode (um Literale im Code utf8-codiert zu haben):

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

Verbindung:

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

Für Webseiten eine davon:

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

PHP

In php.ini (dies ist der Standard nach PHP 5.6):

```
default_charset UTF-8
```

Beim Erstellen einer Webseite:

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

Wenn Sie sich mit MySQL verbinden:

```
(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);
```

Verwenden Sie im Code keine Konvertierungsroutinen.

Für die Dateneingabe

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

Für JSON, um `\uxxxx` zu vermeiden:

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

Verbindung mit UTF-8 unter Verwendung verschiedener Programmiersprachen. online lesen: https://riptutorial.com/de/mysql/topic/7332/verbindung-mit-utf-8-unter-verwendung-verschiedener-programmiersprachen-

Kapitel 67: Verwenden von Variablen

Examples

Variablen einstellen

Hier sind einige Möglichkeiten, um Variablen zu setzen:

1. Sie können eine Variable mit SET auf eine bestimmte Zeichenfolge, eine bestimmte Zahl oder ein Datum setzen

   EX: SET @var_string = 'my_var'; SET @var_num = '2' SET @var_date = '2015-07-20';

2. Sie können eine Variable als Ergebnis einer select-Anweisung festlegen, indem Sie: = verwenden

   EX: Wählen Sie @var: = '123'; (Hinweis: Sie müssen: = verwenden, wenn Sie eine Variable zuweisen, die nicht die SET-Syntax verwendet, da in anderen Anweisungen (select, update ...) das "=" zum Vergleichen verwendet wird. Wenn Sie also einen Doppelpunkt vor dem "" = ", Sie sagen" Dies ist kein Vergleich, das ist ein SET ".)

3. Mit INTO können Sie eine Variable als Ergebnis einer select-Anweisung festlegen

   (Dies war besonders hilfreich, wenn ich dynamisch auswählen musste, aus welchen Partitionen eine Abfrage durchgeführt werden sollte.)

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;
```

Zeilennummer und Gruppierung nach Variablen in Select-Anweisung

Nehmen wir an, wir haben eine Tabelle `team_person` wie `team_person` :

```
+======+===========+
| 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;
```

Zum Auswählen der Tabelle `team_person` mit zusätzlicher `row_number` Spalte

```
SELECT @row_no := @row_no+1 AS row_number, team, person
FROM team_person, (SELECT @row_no := 0) t;
```

ODER

```
SET @row_no := 0;
SELECT  @row_no := @row_no + 1 AS row_number, team, person
FROM team_person;
```

wird das Ergebnis unten ausgeben:

```
+============+======+===========+
| row_number | team |    person |
+============+======+===========+
|          1 |   A  |      John |
+------------+------+-----------+
|          2 |   B  |     Smith |
+------------+------+-----------+
|          3 |   A  |    Walter |
+------------+------+-----------+
|          4 |   A  |     Louis |
```

```
+------------+------+-----------+
|          5 |   C  | Elizabeth |
+------------+------+-----------+
|          6 |   B  |     Wayne |
+------------+------+-----------+
```

Schließlich, wenn wir die `row_number` Gruppe nach Spalte `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 |
+------------+------+-----------+
```

Verwenden von Variablen online lesen: https://riptutorial.com/de/mysql/topic/5013/verwenden-von-variablen

Kapitel 68: Viele-zu-viele-Zuordnungstabelle

Bemerkungen

- Keine `AUTO_INCREMENT` ID für diese Tabelle vorhanden - Die angegebene PK ist die "natürliche" PK; Es gibt keinen guten Grund für einen Ersatz.
- `MEDIUMINT` - Dies ist eine Erinnerung daran, dass alle `INTs` so klein wie möglich gemacht werden sollten (kleiner ⇒ schneller). Natürlich muss die Deklaration hier mit der Definition in der Tabelle übereinstimmen, mit der verlinkt wird.
- `UNSIGNED` - Nahezu alle INTs können auch als nicht negativ deklariert werden
- `NOT NULL` - Das stimmt, nicht wahr?
- `InnoDB` - Effizienter als MyISAM aufgrund der Art und Weise, wie der `PRIMARY KEY` mit den Daten in InnoDB gebündelt wird.
- `INDEX(y_id, x_id)` - Der `PRIMARY KEY` macht es effizient, in eine Richtung zu gehen; das macht die andere Richtung effizient. Keine Notwendigkeit, `UNIQUE` zu sagen; das wäre ein zusätzlicher Aufwand für `INSERTs` .
- Im Sekundärindex würde nur die `INDEX(y_id)` funktionieren, da dies implizit die `x_id` von `x_id` . Ich möchte jedoch eher deutlich machen, dass ich auf einen "abdeckenden" Index hoffe.

Sie *können* der Tabelle weitere Spalten hinzufügen. das ist selten. Die zusätzlichen Spalten können Informationen über die *Beziehung enthalten* , die die Tabelle darstellt.

Möglicherweise möchten Sie `FOREIGN KEY` Einschränkungen hinzufügen.

Examples

Typisches Schema

```
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;
```

(Siehe Anmerkungen unten, für Gründe.)

Viele-zu-viele-Zuordnungstabelle online lesen: https://riptutorial.com/de/mysql/topic/4857/viele-zu-viele-zuordnungstabelle

Kapitel 69: Volltextsuche

Einführung

MySQL bietet die Volltextsuche. Es durchsucht Tabellen mit Spalten mit Text nach den besten Übereinstimmungen für Wörter und Ausdrücke.

Bemerkungen

`FULLTEXT` funktioniert merkwürdig bei Tabellen, die eine geringe Anzahl von Zeilen enthalten. Wenn Sie also damit experimentieren, kann es hilfreich sein, online einen mittelgroßen Tisch zu erhalten. Hier ist eine Tabelle mit Buchartikeln , Titeln und Autoren. Sie können es herunterladen, entpacken und in MySQL laden.

`FULLTEXT` ist für die Verwendung mit menschlicher Unterstützung gedacht. Es wurde entwickelt, um mehr Übereinstimmungen zu erzielen als bei einer gewöhnlichen `WHERE column LIKE 'text%'` .

`FULLTEXT` Suche ist für `MyISAM` Tabellen verfügbar. Es ist auch für `InnoDB` Tabellen in MySQL Version 5.6.4 oder höher verfügbar.

Examples

Einfache FULLTEXT-Suche

```
   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;
```

Bei einer Tabelle mit dem Namen `book` mit den Spalten `ISBN` , 'Title' und 'Author' werden Bücher gefunden, die den Begriffen `'Database Programming'` . Es zeigt zuerst die besten Übereinstimmungen.

Damit dies funktioniert, muss ein Volltextindex für die `Title` verfügbar sein:

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_index (Title);
```

Einfache BOOLEAN-Suche

```
   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;
```

Bei einer Tabelle mit dem Namen " `book` mit den Spalten " `ISBN` , " `Title` und " `Author` wird die Suche nach Büchern mit den Wörtern `'Database'` und `'Programming'` im Titel durchgeführt, nicht jedoch nach dem Wort `'Java'` .

Damit dies funktioniert, muss ein Volltextindex für die Titelspalte verfügbar sein:

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_index (Title);
```

Mehrspaltige FULLTEXT-Suche

```
    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;
```

Bei einer Tabelle mit dem Namen "book" mit den Spalten " `ISBN` , " `Title` und " `Author` werden Bücher gefunden, die den Begriffen "Date Database Programming" entsprechen. Es zeigt zuerst die besten Übereinstimmungen. Zu den besten Übereinstimmungen zählen Bücher von Prof. CJ Date.

(Eines der besten Übereinstimmungen ist jedoch auch *der Date Doctor's Guide für das Dating: Wie man vom ersten Date bis zum Perfect Mate kommt* . Dies zeigt eine Einschränkung der FULLTEXT-Suche: Es gibt nicht vor, solche Dinge als Teile der Sprache zu verstehen die Bedeutung der indizierten Wörter.)

Damit dies funktioniert, muss ein Volltextindex für die Spalten Titel und Autor verfügbar sein:

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_author_index (Title, Author);
```

Volltextsuche online lesen: https://riptutorial.com/de/mysql/topic/8759/volltextsuche

Kapitel 70: WÄHLEN

Einführung

`SELECT` wird verwendet, um aus einer oder mehreren Tabellen ausgewählte Zeilen abzurufen.

Syntax

- SELECT DISTINCT [Ausdrücke] FROM TableName [WHERE-Bedingungen]; /// Einfache Auswahl
- SELECT DISTINCT (a), b ... ist das gleiche wie SELECT DISTINCT a, b ...
- SELECT [ALL | DISTINCT | DISTINCTROW] [HIGH_PRIORITY] [STRAIGHT_JOIN] [SQL_SMALL_RESULT | SQL_BIG_RESULT] [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] Ausdrücke FROM Tabellen [WHERE-Bedingungen] [GROUP BY-Ausdrücke] [HAVING-Bedingung] [ORDER BY-Ausdruck [ASC | DESC]] [LIMIT [offset_value] number_rows | LIMIT number_rows OFFSET offset_value] [PROCEDURE Prozedurname] [INTO [Optionen für OUTFILE 'Dateiname' | DUMPFILE 'Dateiname' | @ variable1, @ variable2, ... @variable_n] [FOR UPDATE | LOCK IN SHARE-MODUS]; /// Full Select-Syntax

Bemerkungen

Weitere Informationen zur `SELECT` Anweisung von MySQL finden Sie in MySQL Docs .

Examples

SELECT nach Spaltennamen

```
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');
```

Abfrage

```
SELECT id FROM stack;
```

Ergebnis

```
+------+
| id   |
+------+
|    1 |
|    2 |
+------+
```

SELECT alle Spalten (*)

Abfrage

```
SELECT * FROM stack;
```

Ergebnis

```
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | admin    |
|    2 | stack    | stack    |
+------+----------+----------+
2 rows in set (0.00 sec)
```

Sie können alle Spalten aus einer Tabelle in einem Join auswählen, indem Sie Folgendes tun:

```
SELECT stack.* FROM stack JOIN Overflow ON stack.id = Overflow.id;
```

Bewährtes Verfahren Verwenden Sie `*` nicht, wenn Sie die Zeile (n) nicht in assoziativen Arrays debuggen oder abrufen. Andernfalls können Schemaänderungen (ADD / DROP / neu anordnen) zu bösen Anwendungsfehlern führen. Wenn Sie die Liste der Spalten angeben, die Sie in Ihrer Ergebnismenge benötigen, kann der Abfrageplaner von MySQL die Abfrage häufig optimieren.

Pros:

1. Wenn Sie Spalten hinzufügen oder entfernen, müssen Sie keine Änderungen an den `SELECT *` vornehmen, an denen Sie `SELECT *`
2. Es ist kürzer zu schreiben
3. Sie sehen auch die Antworten, kann `SELECT *` -Anwendung jemals gerechtfertigt sein?

Nachteile:

1. Sie geben mehr Daten zurück, als Sie benötigen. Angenommen, Sie fügen eine VARBINARY-Spalte hinzu, die 200.000 Zeilen pro Zeile enthält. Sie benötigen diese Daten nur an einem Ort für einen einzelnen Datensatz. Mit `SELECT *` Sie 2 MB pro 10 Zeilen zurückgeben, die Sie nicht benötigen
2. Explizit darüber, welche Daten verwendet werden
3. Wenn Sie Spalten angeben, erhalten Sie einen Fehler, wenn eine Spalte entfernt wird
4. Der Abfrageprozessor muss etwas mehr Arbeit erledigen - herausfinden, welche Spalten in der Tabelle vorhanden sind (danke @vinodadhikary)
5. Sie können leichter finden, wo eine Spalte verwendet wird

6. Sie erhalten alle Spalten in Joins, wenn Sie SELECT * verwenden.
7. Sie können die Ordinalreferenzierung nicht sicher verwenden (obwohl die Verwendung von Ordinalreferenzen für Spalten an sich eine schlechte Praxis ist)
8. Bei komplexen Abfragen mit `TEXT` Feldern kann die Abfrage durch die weniger optimale Verarbeitung der temporären Tabelle verlangsamt werden

Wähle mit WO

Abfrage

```
SELECT * FROM stack WHERE username = "admin" AND password = "admin";
```

Ergebnis

```
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | admin    |
+------+----------+----------+
1 row in set (0.00 sec)
```

Abfrage mit einem verschachtelten SELECT in der WHERE-Klausel

Die `WHERE` Klausel kann eine beliebige gültige `SELECT` Anweisung enthalten, um komplexere Abfragen zu schreiben. Dies ist eine "verschachtelte" Abfrage

Abfrage

Verschachtelte Abfragen werden normalerweise verwendet, um einzelne atomare Werte aus Abfragen für Vergleiche zurückzugeben.

```
SELECT title FROM books WHERE author_id = (SELECT id FROM authors WHERE last_name = 'Bar' AND
first_name = 'Foo');
```

Wählt alle Benutzernamen ohne E-Mail-Adresse aus

```
SELECT * FROM stack WHERE username IN (SELECT username FROM signups WHERE email IS NULL);
```

Haftungsausschluss: Erwägen Sie die Verwendung von Joins für Leistungsverbesserungen, wenn Sie eine gesamte Ergebnismenge vergleichen.

WÄHLEN mit 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" überall:

```
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   |
+----+-----------+
```

Beginnt mit "adm":

```
SELECT * FROM stack WHERE username LIKE "adm%";
+----+----------+
| id | username |
+----+----------+
|  1 | admin    |
|  3 | adm      |
|  6 | adm now  |
+----+----------+
```

Endet mit "adm":

```
SELECT * FROM stack WHERE username LIKE "%adm";
+----+----------+
| id | username |
+----+----------+
|  3 | adm      |
+----+----------+
```

So wie das Zeichen `%` in einer `LIKE` Klausel einer beliebigen Anzahl von Zeichen entspricht, entspricht das Zeichen _ nur einem Zeichen. Zum Beispiel,

```
SELECT * FROM stack WHERE username LIKE "adm_n";
+----+----------+
| id | username |
+----+----------+
|  1 | admin    |
+----+----------+
```

Leistungshinweise Wenn es einen Index für den `username` , dann

- `LIKE 'adm'` führt das gleiche wie `= 'adm' durch
- `LIKE 'adm%` ist ein "Bereich", ähnlich wie bei `BETWEEN..AND..` Es kann ein Index für die Spalte

verwendet werden.

- `LIKE '%adm'` (oder eine Variante mit einem *führenden* Platzhalter) kann keinen Index verwenden. Deshalb wird es langsam sein. Bei Tabellen mit vielen Zeilen ist es wahrscheinlich so langsam, dass es unbrauchbar ist.
- `RLIKE` (`REGEXP`) ist tendenziell langsamer als `LIKE` , verfügt jedoch über mehr Funktionen.
- Während MySQL die `FULLTEXT` Indizierung für viele Arten von Tabellen und Spalten `FULLTEXT` , werden diese `FULLTEXT` Indizes *nicht* zur Ausführung von Abfragen mit `LIKE` .

SELECT mit Alias (AS)

SQL-Aliase werden verwendet, um eine Tabelle oder eine Spalte vorübergehend umzubenennen. Sie werden im Allgemeinen zur Verbesserung der Lesbarkeit verwendet.

Abfrage

```
SELECT username AS val FROM stack;
SELECT username val FROM stack;
```

(Hinweis: `AS` ist syntaktisch optional.)

Ergebnis

```
+-------+
| val   |
+-------+
| admin |
| stack |
+-------+
2 rows in set (0.00 sec)
```

SELECT mit einer LIMIT-Klausel

Abfrage:

```
SELECT *
  FROM Customers
 ORDER BY CustomerID
 LIMIT 3;
```

Ergebnis:

Kundennummer	Kundenname	Kontaktname	Adresse	Stadt	Postleitzahl	Land
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Deutschlan
2	Ana Trujillo Emparedados und Helados	Ana Trujillo	Avda. de la Constitución 2222	Mexiko df	05021	Mexiko

3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	Mexiko df	05023	Mexiko

Best Practice Verwenden Sie immer `ORDER BY` wenn Sie `LIMIT` . Andernfalls sind die Zeilen, die Sie erhalten, unvorhersehbar.

Abfrage:

```
SELECT *
  FROM Customers
 ORDER BY CustomerID
 LIMIT 2,1;
```

Erläuterung:

Wenn eine `LIMIT` Klausel zwei Zahlen enthält, wird sie als `LIMIT offset,count` interpretiert `LIMIT offset,count` . In diesem Beispiel überspringt die Abfrage also zwei Datensätze und gibt einen zurück.

Ergebnis:

Kundennummer	Kundenname	Kontaktname	Adresse	Stadt	Postleitzahl	Land
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	Mexiko df	05023	Mexiko

Hinweis:

Die Werte in `LIMIT` Klauseln müssen Konstanten sein. Sie dürfen keine Spaltenwerte sein.

Wähle mit DISTINCT

Die `DISTINCT` Klausel nach `SELECT` doppelte Zeilen aus der Ergebnismenge.

```
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` arbeitet für alle Spalten, um die Ergebnisse zu liefern, nicht für einzelne Spalten. Letzteres ist oft ein Missverständnis neuer SQL-Entwickler. Kurz gesagt, es kommt auf die Unterscheidbarkeit auf Zeilenebene der Ergebnismenge an, nicht auf Unterscheidbarkeit auf Spaltenebene. Um dies zu visualisieren, schauen Sie sich "Audi A1" in der obigen Ergebnisliste an.

Für spätere Versionen von MySQL hat `DISTINCT` neben `ORDER BY` Auswirkungen auf seine Verwendung. Die Einstellung für `ONLY_FULL_GROUP_BY` kommt ins Spiel, wie auf der folgenden MySQL-Handbuchseite mit dem Titel MySQL-Handling von GROUP BY dargestellt .

WÄHLEN mit LIKE (_)

Ein _ Zeichen in einem `LIKE` Klauselmuster entspricht einem einzelnen Zeichen.

Abfrage

```
SELECT username FROM users WHERE users LIKE 'admin_';
```

Ergebnis

```
+----------+
| username |
+----------+
| admin1   |
| admin2   |
| admin-   |
| adminA   |
+----------+
```

SELECT mit CASE oder IF

Abfrage

```
SELECT st.name,
       st.percentage,
       CASE WHEN st.percentage >= 35 THEN 'Pass' ELSE 'Fail' END AS `Remark`
FROM student AS st ;
```

Ergebnis

```
+--------------------------------+
|   name   | percentage | Remark |
+--------------------------------+
|   Isha   |     67     |  Pass  |
|   Rucha  |     28     |  Fail  |
|   Het    |     35     |  Pass  |
|   Ansh   |     92     |  Pass  |
```

```
+--------------------------------+
```

Oder mit 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')
```

Das bedeutet: Wenn st.percentage> = 35 **TRUE ist,** dann `'Pass'` ELSE `'Fail'`

Wähle mit ZWISCHEN

Sie können die Klausel BETWEEN verwenden, um eine Kombination der Bedingungen "größer als gleich UND kleiner als gleich" zu ersetzen.

Daten

```
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  2 | root      |
|  3 | toor      |
|  4 | mysql     |
|  5 | thanks    |
|  6 | java      |
+----+-----------+
```

Abfrage mit Operatoren

```
SELECT * FROM stack WHERE id >= 2 and id <= 5;
```

Ähnliche Abfrage mit BETWEEN

```
SELECT * FROM stack WHERE id BETWEEN 2 and 5;
```

Ergebnis

```
+----+-----------+
| id | username  |
+----+-----------+
|  2 | root      |
|  3 | toor      |
|  4 | mysql     |
|  5 | thanks    |
+----+-----------+
```

```
4 rows in set (0.00 sec)
```

Hinweis

> **BETWEEN verwendet `>=` und `<=` , nicht `>` und `<` .**

Verwenden Sie NICHT ZWISCHEN

Wenn Sie das Negativ verwenden möchten, können Sie `NOT` . Zum Beispiel :

```
SELECT * FROM stack WHERE id NOT BETWEEN 2 and 5;
```

Ergebnis

```
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  6 | java      |
+----+-----------+
2 rows in set (0.00 sec)
```

Hinweis

> **NOT BETWEEN verwendet `>` und `<` und nicht `>=` und `<=`** Das heißt, `WHERE id NOT BETWEEN 2 and 5` ist identisch mit `WHERE (id < 2 OR id > 5)` .

Wenn Sie einen Index für eine Spalte haben, die Sie in einer `BETWEEN` Suche verwenden, kann MySQL diesen Index für eine Bereichsüberprüfung verwenden.

SELECT mit Datumsbereich

```
SELECT ... WHERE dt >= '2017-02-01'
             AND dt  < '2017-02-01' + INTERVAL 1 MONTH
```

Sicher, das könnte man mit `BETWEEN` und Aufnahme von `23:59:59` . Das Muster hat jedoch folgende Vorteile:

- Das Enddatum ist nicht vorberechnet (dies ist oft eine exakte Länge von Anfang an)
- Sie schließen nicht beide Endpunkte ein (wie bei `BETWEEN`), oder geben Sie '23: 59: 59 'ein, um dies zu vermeiden.
- Es funktioniert für `DATE` , `TIMESTAMP` , `DATETIME` und sogar für die im Mikrosekundenbereich enthaltene `DATETIME(6)` .
- Es kümmert sich um Schalttage, Jahresende usw.
- Es ist indexfreundlich (so ist es `BETWEEN`).

WÄHLEN online lesen: https://riptutorial.com/de/mysql/topic/3307/wahlen

Kapitel 71: Zeichenkettenoperationen

Parameter

Name	Beschreibung
ASCII()	Gibt den numerischen Wert des am weitesten links stehenden Zeichens zurück
BEHÄLTER()	Gibt eine Zeichenfolge zurück, die eine binäre Darstellung einer Zahl enthält
BIT_LENGTH ()	Gibt die Länge des Arguments in Bits zurück
VERKOHLEN()	Gibt das Zeichen für jede übergebene ganze Zahl zurück
CHAR_LENGTH ()	Anzahl der Zeichen im Argument zurückgeben
CHARACTER_LENGTH ()	Synonym für CHAR_LENGTH ()
CONCAT ()	Gibt eine verkettete Zeichenfolge zurück
CONCAT_WS ()	Rücklauf verketten mit Trennzeichen
ELT ()	Zeichenfolge an Indexnummer zurückgeben
EXPORT_SET ()	Geben Sie eine Zeichenfolge zurück, sodass für jedes in den Wertebits gesetzte Bit eine Ein-Zeichenfolge und für jedes nicht gesetzte Bit eine Aus-Zeichenfolge angezeigt wird
FELD()	Liefert den Index (Position) des ersten Arguments in den nachfolgenden Argumenten
FIND_IN_SET ()	Gibt die Indexposition des ersten Arguments innerhalb des zweiten Arguments zurück
FORMAT()	Gibt eine Zahl zurück, die auf die angegebene Anzahl von Dezimalstellen formatiert ist
FROM_BASE64 ()	Dekodieren Sie in eine Basis-64-Zeichenfolge und geben Sie das Ergebnis zurück
VERHEXEN()	Gibt eine hexadezimale Darstellung eines Dezimal- oder Zeichenfolgenwerts zurück
EINFÜGEN()	Fügen Sie an der angegebenen Position einen Teilstring bis zur angegebenen Zeichenanzahl ein

Name	Beschreibung
INSTR ()	Gibt den Index des ersten Vorkommens der Teilzeichenfolge zurück
LCASE ()	Synonym für LOWER ()
LINKS()	Gibt die äußerste linke Anzahl von Zeichen wie angegeben zurück
LÄNGE()	Gibt die Länge einer Zeichenfolge in Byte zurück
MÖGEN	Einfache Musteranpassung
LADE DATEI()	Laden Sie die genannte Datei
LOKALISIEREN()	Gibt die Position des ersten Vorkommens der Teilzeichenfolge zurück
NIEDRIGER()	Geben Sie das Argument in Kleinbuchstaben zurück
LPAD ()	Gibt das Zeichenfolgenargument zurück, das mit der angegebenen Zeichenfolge links aufgefüllt ist
LTRIM ()	Entfernen Sie führende Leerzeichen
MAKE_SET ()	Gibt einen Satz von durch Kommas getrennten Zeichenfolgen zurück, für die das entsprechende Bit in Bits gesetzt ist
SPIEL	Volltextsuche durchführen
MID ()	Gibt einen Teilstring an der angegebenen Position zurück
NICHT WIE	Negation des einfachen Mustervergleichs
NICHT REGEXP	Negation von REGEXP
ÜLG ()	Gibt eine Zeichenfolge zurück, die eine oktale Darstellung einer Zahl enthält
OCTET_LENGTH ()	Synonym für LENGTH ()
ORD ()	Gibt den Zeichencode für das linke Zeichen des Arguments zurück
POSITION()	Synonym für LOCATE ()
ZITAT()	Das Argument kann in einer SQL-Anweisung verwendet werden
REGEXP	Musterabgleich mit regulären Ausdrücken

Name	Beschreibung
WIEDERHOLEN()	Wiederholen Sie eine Zeichenfolge so oft wie angegeben
ERSETZEN()	Ersetzen Sie Vorkommen einer angegebenen Zeichenfolge
UMKEHREN()	Umkehren der Zeichen in einer Zeichenfolge
RECHT()	Gibt die angegebene Anzahl von Zeichen ganz rechts zurück
RLIKE	Synonym für REGEXP
RPAD ()	Zeichenfolge an die angegebene Anzahl von Malen anhängen
RTRIM ()	Entfernen Sie nachfolgende Leerzeichen
SOUNDEX ()	Gib einen Soundex-String zurück
HÖRT SICH AN WIE	Sounds vergleichen
PLATZ()	Gibt eine Zeichenfolge mit der angegebenen Anzahl von Leerzeichen zurück
STRCMP ()	Vergleichen Sie zwei Zeichenketten
SUBSTR ()	Gibt den Teilstring wie angegeben zurück
SUBSTRING ()	Gibt den Teilstring wie angegeben zurück
SUBSTRING_INDEX ()	Gibt eine Teilzeichenfolge aus einer Zeichenfolge vor der angegebenen Anzahl von Vorkommen des Trennzeichens zurück
TO_BASE64 ()	Gibt das in eine Basis-64-Zeichenfolge konvertierte Argument zurück
TRIMMEN()	Entfernen Sie führende und nachgestellte Leerzeichen
UCASE ()	Synonym für UPPER ()
UNHEX ()	Gibt eine Zeichenfolge zurück, die eine Hexendarstellung einer Zahl enthält
OBERER, HÖHER()	Konvertieren Sie in Großbuchstaben
WEIGHT_STRING ()	Gibt die Gewichtszeichenfolge für eine Zeichenfolge zurück

Examples

Findet das Element in einer durch Kommas getrennten Liste

```
SELECT FIND_IN_SET('b','a,b,c');
```

Rückgabewert:

2

```
SELECT FIND_IN_SET('d','a,b,c');
```

Rückgabewert:

0

STR_TO_DATE - Konvertiert den String in das Datum

Mit einer Spalte eines der Zeichenfolgentypen namens `my_date_field` mit einem Wert wie [die Zeichenfolge] `07/25/2016` zeigt die folgende Anweisung die Verwendung der Funktion `STR_TO_DATE` :

```
SELECT STR_TO_DATE(my_date_field, '%m/%d/%Y') FROM my_table;
```

Sie können diese Funktion auch als Teil der `WHERE` Klausel verwenden.

UNTER () / LCASE ()

Konvertieren Sie das String-Argument in Kleinbuchstaben

Syntax: LOWER (str)

```
LOWER('fOoBar') -- 'foobar'
LCASE('fOoBar') -- 'foobar'
```

ERSETZEN()

Konvertieren Sie das String-Argument in Kleinbuchstaben

Syntax: REPLACE (str, von_str, bis_str)

```
REPLACE('foobarbaz', 'bar', 'BAR') -- 'fooBARbaz'
REPLACE('foobarbaz', 'zzz', 'ZZZ') -- 'foobarbaz'
```

SUBSTRING ()

SUBSTRING (oder äquivalent: SUBSTR) gibt die Teilzeichenfolge ausgehend von der angegebenen Position und optional mit der angegebenen Länge zurück

Syntax: `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'
```

Syntax: `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 ()

Konvertieren Sie das String-Argument in Großbuchstaben

Syntax: UPPER (str)

```
UPPER('fOoBar') -- 'FOOBAR'
UCASE('fOoBar') -- 'FOOBAR'
```

LÄNGE()

Gibt die Länge der Zeichenfolge in Byte zurück. Da einige Zeichen mit mehr als einem Byte codiert werden können, siehe CHAR_LENGTH ().

Syntax: LÄNGE (str)

```
LENGTH('foobar') -- 6
LENGTH('fööbar') -- 8 -- contrast with CHAR_LENGTH(...) = 6
```

CHAR_LENGTH ()

Gibt die Anzahl der Zeichen in der Zeichenfolge zurück

Syntax: CHAR_LENGTH (str)

```
CHAR_LENGTH('foobar') -- 6
CHAR_LENGTH('fööbar') -- 6 -- contrast with LENGTH(...) = 8
```

HEX (str)

Konvertieren Sie das Argument in Hexadezimal. Dies wird für Strings verwendet.

```
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
```

Zeichenkettenoperationen online lesen:
https://riptutorial.com/de/mysql/topic/1399/zeichenkettenoperationen

Kapitel 72: Zeichensätze und Kollatierungen

Examples

Erklärung

```
CREATE TABLE foo ( ...
    name CHARACTER SET utf8mb4
    ... );
```

Verbindung

Für die Verwendung von Zeichensätzen ist es wichtig, dem MySQL-Server mitzuteilen, wie die Bytes des Clients codiert werden. Hier ist ein Weg:

```
SET NAMES utf8mb4;
```

Jede Sprache (PHP, Python, Java, ...) hat einen eigenen Weg, der normalerweise `SET NAMES` vorzuziehen ist.

Zum Beispiel: `SET NAMES utf8mb4` zusammen mit einer Spalte, die als `CHARACTER SET latin1` deklariert ist. Dies wird beim `INSERTing` von latin1 in utf8mb4 `INSERTing` und beim `SELECTing` zurück `SELECTing` .

Welches Zeichenset und welche Sammlung?

Es gibt Dutzende von Zeichensätzen mit Hunderten von Kollatierungen. (Eine gegebene Kollatierung gehört nur zu einem Zeichensatz.) Siehe Ausgabe von `SHOW COLLATION;` .

In der Regel sind nur 4 `CHARACTER SETs` Bedeutung:

```
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.
```

Alle enthalten englische Zeichen, die identisch codiert sind. utf8 ist eine Teilmenge von utf8mb4.

Beste Übung...

- Verwenden Sie utf8mb4 für jeden `TEXT` oder `VARCHAR` - Spalte , die eine Vielzahl von Sprachen in ihm haben können.
- Verwenden Sie ascii (latin1 ist in Ordnung) für Hex-Zeichenfolgen (UUID, MD5 usw.) und einfache Codes (Ländercode, Postcode usw.).

utf8mb4 gab es bis Version 5.5.3 nicht, daher war utf8 das beste, was davor verfügbar war.

Außerhalb von MySQL bedeutet "UTF8" dasselbe wie utf8mb4 von MySQL und nicht utf8 von

MySQL.

Kollatierungen beginnen mit dem Zeichensatznamen und enden normalerweise mit `_ci` für "Groß- und Kleinschreibung" und " `_bin` " für "Vergleichen Sie einfach die Bits."

Die "neueste" utf8mb4-Kollatierung ist `utf8mb4_unicode_520_ci` , basierend auf Unicode 5.20. Wenn Sie mit einer einzigen Sprache arbeiten, möchten Sie beispielsweise `utf8mb4_polish_ci` werden die Buchstaben basierend auf den polnischen Konventionen leicht neu angeordnet.

Zeichensätze für Tabellen und Felder festlegen

Sie können einen Zeichensatz sowohl pro Tabelle als auch für `CHARSET` einzelne Feld mit den Anweisungen `CHARACTER SET` und `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` und `Country` verwenden `UTF8` , da wir dies als Standardzeichensatz für die Tabelle festlegen. `Street` hingegen wird `ASCII` , da wir dies ausdrücklich gesagt haben.

Die Einstellung des richtigen Zeichensatzes hängt stark von Ihrem Datensatz ab, kann jedoch auch die Portabilität zwischen Systemen verbessern, die mit Ihren Daten arbeiten.

Zeichensätze und Kollatierungen online lesen:
https://riptutorial.com/de/mysql/topic/4569/zeichensatze-und-kollatierungen

Kapitel 73: Zeit mit Sekundengenauigkeit

Bemerkungen

Sie müssen auf MySQL Version 5.6.4 oder höher sein, um Spalten mit Sekundenbruchzeitdatentypen zu deklarieren.

Mit `DATETIME(3)` Sie beispielsweise eine Millisekundenauflösung in Ihren Zeitstempeln, und `TIMESTAMP(6)` eine Mikrosekundenauflösung mit einem * nix-artigen Zeitstempel.

Lesen Sie dies: http://dev.mysql.com/doc/refman/5.7/de/fractional-seconds.html

`NOW(3)` gibt Ihnen die aktuelle Uhrzeit des Betriebssystems Ihres MySQL-Servers in Millisekunden-Genauigkeit.

(Beachten Sie, dass die interne fraktale Arithmetik von MySQL wie * 0,001 immer als Gleitpunkt mit doppelter Genauigkeit nach IEEE754 behandelt wird. Es ist daher unwahrscheinlich, dass Sie an Präzision verlieren, bevor die Sonne zum weißen Zwergstern wird.)

Examples

Holen Sie sich die aktuelle Uhrzeit in Millisekundengenauigkeit

```
SELECT NOW(3)
```

macht den Trick

Holen Sie sich die aktuelle Uhrzeit in einem Formular, das wie ein Javascript-Zeitstempel aussieht.

Javascript Zeitstempel werden auf der Grundlage der ehrwürdige UNIX `time_t` Datentyp und zeigen die Anzahl der Millisekunden seit `1970-01-01 00:00:00` UTC.

Dieser Ausdruck ruft die aktuelle Uhrzeit als JavaScript-Zeitstempel-Ganzzahl ab. (Dies ist unabhängig von der aktuellen Einstellung von time_zone korrekt.)

```
ROUND(UNIX_TIMESTAMP(NOW(3)) * 1000.0, 0)
```

Wenn Sie in einer Spalte `TIMESTAMP` Werte gespeichert haben, können Sie diese mit der Funktion UNIX_TIMESTAMP () als ganzzahlige Javascript-Zeitstempel abrufen.

```
 SELECT ROUND(UNIX_TIMESTAMP(column) * 1000.0, 0)
```

Wenn Ihre Spalte `DATETIME` Spalten enthält und Sie diese als Javascript-Zeitstempel abrufen, werden diese Zeitstempel um den Zeitzonenversatz der Zeitzone versetzt, in der sie gespeichert

sind.

Erstellen Sie eine Tabelle mit Spalten, um die zweite Sekunde zu speichern.

```
CREATE TABLE times (
     dt DATETIME(3),
     ts TIMESTAMP(3)
 );
```

erstellt eine Tabelle mit Millisekunden-genauen Datums- / Uhrzeitfeldern.

```
 INSERT INTO times VALUES (NOW(3), NOW(3));
```

fügt eine Zeile mit `NOW()` Werten mit einer Millisekundengenauigkeit in die Tabelle ein.

```
 INSERT INTO times VALUES ('2015-01-01 16:34:00.123','2015-01-01 16:34:00.128');
```

fügt spezifische Millisekunden-Präzisionswerte ein.

Beachten Sie, dass Sie `NOW(3)` anstelle von `NOW()` wenn Sie diese Funktion verwenden, um hochgenaue Zeitwerte einzufügen.

Konvertieren Sie einen Datums- / Zeitwert in Millisekundengenauigkeit in Text.

`%f` ist der Formatbezeichner für gebrochene Genauigkeit für die Funktion DATE_FORMAT () .

```
SELECT DATE_FORMAT(NOW(3), '%Y-%m-%d %H:%i:%s.%f')
```

zeigt einen Wert wie `2016-11-19 09:52:53.248000` mit gebrochenen Mikrosekunden an. Da wir `NOW(3)` , sind die letzten drei Ziffern im Bruchteil 0.

Speichern Sie einen Javascript-Zeitstempel in einer TIMESTAMP-Spalte

Wenn Sie einen Javascript-Zeitstempelwert haben, zum Beispiel `1478960868932` , können Sie diesen Wert in einen MySQL-Bruchzeitwert wie `1478960868932` konvertieren:

```
FROM_UNIXTIME(1478960868932 * 0.001)
```

Es ist einfach, diese Art von Ausdruck zu verwenden, um Ihren Javascript-Zeitstempel in einer MySQL-Tabelle zu speichern. Mach das:

```
INSERT INTO table (col) VALUES (FROM_UNIXTIME(1478960868932 * 0.001))
```

(Natürlich möchten Sie andere Spalten einfügen.)

Zeit mit Sekundengenauigkeit online lesen: https://riptutorial.com/de/mysql/topic/7850/zeit-mit-sekundengenauigkeit

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit 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	AKTUALISIEREN	4thfloorstudios, Chris, Drew, Khurram, Ponnarasu, Rick James, Sevle
3	ALTER TABELLE	e4c5, JohnLBevan, kolunar, LiuYan , Matas Vaitkevicius, mayojava, Rick James, Steve Chambers, Thuta Aung, WAF, YCF_L
4	Ändere das Passwort	e4c5, Hardik Kanjariya ツ, Rick James, Viktor, ydaetskcoR
5	Arithmetik	Barranka, Dinidu, Drew, JonMark Perry, O. Jones, RamenChef, Richard Hamilton, Rick James
6	AUSSICHT	Abhishek Aggrawal, Divya, e4c5, Marina K., Nikita Kurtin, Ponnarasu, R.K123, ratchet, Rick James, WAF, Yury Fedorov, Илья Плотников
7	Backticks	Drew, SuperDJ
8	Clustering	Drew, Rick James
9	Datenbanken erstellen	Daniel Käfer, Drew, Ponnarasu, R.K123, Rick James, still_learning
10	Datentypen	Batsu, dakab, Drew, Dylan Vander Berg, e4c5, juergen d, MohaMad, Richard Hamilton, Rick James
11	Datum und Uhrzeit	Abhishek Aggrawal, Drew, Matt S, O. Jones, Rick James, Sumit Gupta
12	Dynamische Un-Pivot-Tabelle mit Prepared-Anweisung	rpd

13	EINFÜGEN	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
14	Eins zu vielen	falsefive
15	ENUM	Philipp, Rick James
16	Extrahieren Sie Werte aus dem JSON-Typ	MohaMad
17	Fehler 1055: ONLY_FULL_GROUP_BY: etwas ist nicht in der GROUP BY-Klausel enthalten ...	Damian Yerrick, O. Jones
18	Fehlercodes	Drew, e4c5, juergen d, Lucas Paolillo, O. Jones, Ponnarasu, Rick James, WAF, Wojciech Kazior
19	Gespeicherte Routinen (Prozeduren und Funktionen)	Abhishek Aggrawal, Abubakkar, Darwin von Corax, Dinidu , Drew, e4c5, juergen d, kolunar, llanato, Rick James, userlond
20	Gruppiere nach	Adam, Filipe Martins, Lijo, Rick James, Thuta Aung, WAF, whrrgarbl
21	Indizes und Schlüssel	Alex Recarey, Barranka, Ben Visness, Drew, kolunar, Rick James, Sanjeev kumar
22	Installieren Sie den Mysql-Container mit Docker-Compose	Marc Alff, molavec
23	JOINS: Join 3 Tabelle mit dem gleichen Namen von ID.	FMashiro
24	JSON	A. Raza, Ben, Drew, e4c5, Manatax, Mark Amery, MohaMad, phatfingers, Rick James, sunkuet02
25	Kommentar Mysql	Franck Dernoncourt, Rick James, WAF, YCF_L
26	Konfiguration und Abstimmung	ChintaMoney, CodeWarrior, Epodax, Eugene, jan_kiran, Rick James
27	Konvertierung von MyISAM nach InnoDB	Ponnarasu, Rick James, yukoff

28	LADEN DATEN INFILE	aries12, Asaph, bhrached, CGritton, e4c5, RamenChef, Rick James, WAF
29	Leistungsoptimierung	e4c5, RamenChef, Rick James
30	Limit und Offset	Alvaro Flaño Larrondo, Ani Menon, animuson, ChaoticTwist, Chris Rasys, CPHPython, Ian Gregory, Matt S, Rick James, Sumit Gupta, WAF
31	LÖSCHEN	Batsu, Drew, e4c5, ForguesR, gabe3886, Khurram, Parth Patel, Ponnarasu, Rick James, strangeqargo, WAF, whrrgarbl, ypercube, Илья Плотников
32	LÖST AUS	Blag, e4c5, Matas Vaitkevicius, ratchet, WAF, YCF_L
33	MyISAM Engine	Rick James
34	Mysql Performance-Tipps	arushi, RamenChef, Rick James, Rodrigo Darti da Costa
35	MySQL-Admin	Florian Genser, Matas Vaitkevicius, RationalDev, Rick James
36	MySQL-Client	Batsu, Nathaniel Ford, Rick James
37	MySQL-Gewerkschaften	Ani Menon, Rick James
38	mysqlimport	Batsu
39	MySQL-Sperrtabelle	Ponnarasu, Rick James, vijeeshin
40	Neuen Benutzer erstellen	Aminadav, Batsu, Hardik Kanjariya ツ, josinalvo, Rick James, WAF
41	NULL	Rick James, Sumit Gupta
42	Partitionierung	Majid, Rick James
43	Passen Sie PS1 an	Eugene, Wenzhong
44	Pivot-Abfragen	Barranka
45	PREPARE-Anweisungen	kolunar, Rick James, winter
46	Protokolldateien	Drew, Rick James
47	Reguläre Ausdrücke	user2314737, YCF_L
48	Replikation	Ponnarasu
49	Reservierte Wörter	juergen d, user2314737

50	Schließt sich an	Artisan72, Batsu, Benvorth, Bikash P, Drew, Matt, Philipp, Rick, Rick James, user3617558
51	Serverinformation	FMashiro
52	Sicherheit über GRANTs	Rick James
53	Sichern Sie mit mysqldump	agold, Asaph, Barranka, Batsu, KalenGi, Mark Amery, Matthew, mnoronha, Ponnarasu, RamenChef, Rick James, still_learning, strangeqargo, Sumit Gupta, Timothy, WAF
54	SORTIEREN NACH	Florian Genser, Rick James
55	SSL-Verbindungsaufbau	4444, a coder, Eugene
56	Stellen Sie das Standard-Root-Passwort für MySQL 5.7 und höher wieder her	Lahiru, ParthaSen
57	Stellen Sie das verlorene Root-Passwort wieder her	BacLuc, Jen R
58	Tabelle ablegen	Noah van der Aa, Parth Patel, Ponnarasu, R.K123, Rick James, trf, Tushar patel, YCF_L
59	Tabellenerstellung	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
60	Temporäre Tabellen	Ponnarasu, Rick James
61	Transaktion	Ponnarasu, Rick James
62	Umgang mit spärlichen oder fehlenden Daten	Batsu, Nate Vaughan
63	Umgang mit Zeitzonen	O. Jones
64	UNION	Mattew Whitt, Rick James, Riho, Tarik, wangengzheng
65	Veranstaltungen	Drew, rene
66	Verbindung mit UTF-8 unter Verwendung	Epodax, Rick James

	verschiedener Programmiersprachen.	
67	Verwenden von Variablen	kolunar, user6655061
68	Viele-zu-viele-Zuordnungstabelle	Rick James
69	Volltextsuche	O. Jones
70	WÄHLEN	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
71	Zeichenkettenoperationen	Abubakkar, Batsu, juergen d, kolunar, Rick James, uruloke, WAF
72	Zeichensätze und Kollatierungen	frlan, Rick, Rick James
73	Zeit mit Sekundengenauigkeit	O. Jones