MySQL Tutorial 5: SQL Examples

An index of useful SQL commands

— P. Lutus — Message Page —

MySQL Command Summary | Query Techniques | Topical Links

(double-click any word to see its definition)

— Navigate this multi-page article with the arrows and drop-down lists at the top and bottom of each page —

Note: In this article, footnotes are marked with a light bulb over which one hovers .

MySQL Command Summary

In this section I provide an overview of some SQL commands, the Structured Query Language behind many modern databases and database engines like MySQL . Remember that the syntax in the examples below is MySQL-specific — unfortunately, different database engines use different commands to accomplish the same ends.

As explained above, a modern database consists of tables, a table consists of records, and a record consists of fields.

- A field: "John"
- Another field: "Doe"
- A record: "John","Doe","123 Elm Street","Switchback","West Virginia"

(And no, boys and girls, I didn't make up the town of Switchback, West Virginia.)

To interactively experiment with the following commands, you have a number of options, including:

- Running the "mysql" command-line application.
- Using DBCLient's MySQL terminal.

But remember that an ill-formed SQL command can wipe out tables and/or databases, so *be careful*. One way to be careful is to create a new database and table just to experiment — avoid databases containing information you need to keep.

One of the reasons for this section is to put a number of hard-to-remember SQL commands and procedures in one place, as well as to familiarize the reader with typical database operations. Let's get started — each of the examples that follows contains the SQL you would type to get the desired outcome.

Create a Database

```
CREATE DATABASE database_name
```

Unless the database already exists, it will be created.

Delete a Database

```
DROP DATABASE database_name
```

You almost never want to do this, at least not without some deep thought.

Create a table

```
CREATE TABLE if not exists database_name.table_name (fieldname1 type1, fieldname2, type2 ...)
```

You need to specify what fields the table has, and what kind of data they hold. Here's an example:

```
CREATE TABLE if not exists db.people (First text, Last text, Age int)
```

There's one more refinement when creating tables — including a primary key. An auto-incrementing

primary key assures that adding a new record to an existing table won't erase a prior record. Like this:

CREATE TABLE if not exists db.people (First text, Last text, Age int, pk int not null auto_increment primary key)

The field "pk" (any name will do) provides some important properties to the table — each record is now unique, and the key also allows the database engine to operate more efficiently.

When you insert data into a table that has a primary key, you don't normally specify the key in your entry. The database engine takes care of that — it automatically creates the key field and gives it a unique value.

The use of the phrase "if not exists" is meant to avoid trying to create a table that already exists, which would produce an error. To replace an existing table with a new design, first delete the prior table.

Delete a table

DROP TABLE database_name.table_name

Again, as before, this can be misused or applied accidentally.

Suppose you want to keep the table's structure (the field definitions) but erase the data in the table. Here's how:

Truncate a table

TRUNCATE TABLE database_name.table_name

All data is deleted, but the table's basic structure remains.

Drop (delete) a field (column) from a table

ALTER TABLE database_name.table_name DROP COLUMN fieldname

This deletes the field and associated data. (The term "column" is often used as a synonym for "field".)

Delete a record from a table

DELETE FROM database_name.table_name WHERE (record identifiers)

This deletes a specific record from a table. Be careful to specify which record unambiguously to avoid deleting more records than you intend. If each record has a unique primary key, use that. Otherwise, list the values of each field for the intended record, like this:

DELETE FROM database_name.table_name WHERE Name = 'John Doe' and Address = '123 Elm Street' and ... (so forth)

Retrieve specific records and fields from a table

SELECT * FROM db.people

The above means "provide all the records and fields in the table 'db.people'". But we might want to specify the retrieval of particular fields, and we might also want to specify which records to retrieve. Like this:

SELECT Last,Age FROM db.people WHERE AGE > 60

The field list "Last,Age" specifies which fields to include in the result, and the "WHERE AGE > 60" part specifies which records to retrieve.

Insert records into a table

INSERT INTO db.people (First,Last,Age) VALUES("John","Doe",31)

The above creates a new record in the table "db.people". Notice that a specifier for the primary key field isn't included. That's intentional — it lets the database engine manage the key.

Here's an INSERT example that reads from one table and writes to another:

```
INSERT INTO db_name.dest_table (First,Last,Age) SELECT First,Last,Age from db_name.source_table
```

The above method can be used to accumulate data of various kinds:

```
INSERT INTO db_name.geriatric_table (First,Last,Age) SELECT First,Last,Age from db_name.source_table
WHERE Age > 65

INSERT INTO db_name.teens_table (First,Last,Age) SELECT First,Last,Age from db_name.source_table
WHERE Age < 19
```

I emphasize that such selective queries can be as complex as one wishes:

```
INSERT INTO db_name.dest_table (First,Last,Age) SELECT First,Last,Age from db_name.source_table
WHERE (Age < 18 and IQ > 120) OR (AGE >= 22 AND HairColor = "UnforeseeableFuchsia")
```

Copy a table

```
CREATE TABLE database_name.copy_name LIKE database_name.original_name

INSERT database_name.copy_name SELECT * FROM database_name.original_name
```

There are a number of ways to copy a table, but most of them are incomplete. The first line above copies the original table's structure, and the second line copies the data, leaving nothing out. Other copying methods may not copy every detail.

Change a field's datatype

This operation may seem esoteric, but it's surprisingly useful. You've created a table and populated it with data, only then to realize that one or more of the fields has the wrong data type. This often involves using a "text" datatype for everything during massive data conversions, only later to discover the drawbacks of this choice.

```
ALTER TABLE db_name.ziptable CHANGE latitude latitude double
```

Notice that the field name is repeated — this is to allow a renaming of the field as the conversion takes place.

This operation should be used carefully. It's possible to lose data by, for example, specifying an integer data type for data that has floating-point digits, i.e. digits lying to the right of the decimal point.

Change a field's contents

```
UPDATE db_name.table_name SET fieldname = replace(fieldname,'old content','new content')
```

Here's an alternate form that I needed to replace a blank field with a default value (the above form didn't work for that case):

```
UPDATE db_name.table_name SET fieldname = 'new content' WHERE fieldname = 'old content'
```

In these examples, the same field name is used for the search and replace arguments, but that's not required. This command can selectively replace particular data with other data across an entire table.

Insert a new field into a table

```
ALTER TABLE db_name.table_name ADD COLUMN column_name datatype [FIRST] [AFTER
column_name]
```

Example:

```
ALTER TABLE mybase.mytable ADD COLUMN Extra text AFTER Ordinary
```

Remember that "column" is a synonym for "field". If neither a FIRST or AFTER (name) argument is included, the new column will be placed at the end of the row.

Create a "Last Modified" field

This example is somewhat exotic but quite useful — it creates a field that displays the most recent modification date and time of the record it belongs to.

```
ALTER TABLE db_name.table_name ADD LastModified TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

UPDATE db_name.table_name SET LastModified=CURRENT_TIMESTAMP
```

Once this field is created, it will display the most recent modification time for each record. This is a very useful field category that DBClient knows about and works with (meaning DBCLient won't overwrite the MySQL-updated "last modified" timestamp).

Once this field is in place, it's important to avoid recreating it, an action that would reset all the modified times to the present time. In that connection, it would be nice if we could say:

```
ALTER TABLE db_name.table_name ADD COLUMN IF NOT EXISTS LastModified ...
```

But this syntax doesn't exist at the time of writing.

Eliminate Duplicate Records

It's surprisingly common to have a database containing duplicate records. There are a number of ways to eliminate duplicates — here's one:

```
drop table if exists `temp`

CREATE TABLE `temp` LIKE `SourceTable`

INSERT `temp` SELECT * FROM `SourceTable` WHERE 1 GROUP BY `Field1`,`Field2`, (etc.)

DROP TABLE `SourceTable`

RENAME TABLE `temp` TO `SourceTable`
```

The idea of the INSERT line above is to group the duplicate records and choose only one of the duplicates (WHERE 1). Therefore the GROUP BY clause must be specific enough to identify only duplicate records, but must not include a unique key if one exists (which is why we cannot say GROUP BY *). If a unique key is included in the GROUP BY clause, the duplicate elimination won't work, because each record has a unique key, including duplicate records.

Create a view instead of a query

"Views" are a relatively recent addition to MySQL. They're based on the idea of storing what might be a complex query and accessing it by name. The advantage of a view is that, when the source tables change, so does the view — no need to regenerate a table derived from a query.

```
CREATE VIEW db_name.view_name AS SELECT (Field1,Field2,Field3) from db_name.source_table WHERE
Field1 LIKE "%cycle%" and Field2 > 340 and Field3 < Field2 Order by Field2 DESC
```

A view encapsulates a particular way of looking at a table's contents. Unlike a query that creates a new table, a view accesses its sources and recreates its result each time it's accessed. This means if its sources change, so does the view.

Change an existing table's character set

The first example won't change the encoding of records already in the table, but it will allow use of the new character set for new records. I needed this for a table that was inadventently created using the Latin-1 character set but needed to support UTF-8. As I added records, I eventually encountered a UTF-8 sequence, which was rejected. So I applied this remedy, which had no effect on the pre-existing records but allows UTF-8 encoding for new ones.

```
ALTER TABLE db_name.table_name CHARACTER SET utf8 COLLATE utf8_unicode_ci
```

This next example converts all records to the new character set. This should be applied only if there's no possibility of a conflict between the old and new character encodings.

```
ALTER TABLE db_name.table_name CONVERT TO CHARACTER SET utf8 COLLATE utf8_unicode_ci
```

Query Techniques

There are a number of ways to write queries to produce interesting results. The simplest involves creative use of the GROUP BY clause, as explained in prior sections of this article. Suppose we need to break a table's contents down by category and get a total of each. For example this ZIP code database has 43,191 records, but no one wants to simply read the database line by line to find things out.

If you don't already have the ZIP code database on your system:

- Download this database definition file.
- Put it somewhere convenient and memorable.
- Run the MySQL command-line application "mysql", as explained in earlier sections of this aticle.
- From that application, type this: "source (path)/zipcodes.sql", including the path to the ZIP codes database file on your system.
- If the MySQL application was able to locate the file, you should now have a ZIP code table.

Now we can answer questions by writing MySQL queries.

- Q: How many zipcodes in the database?

```
mysql> use tutorial;
mysql> select count(*) as Total from zipcodes;

+-------+
| Total |
+-------+
| 43191 |
+-------+
```

- Q: How many ZIP codes in each state?

```
mysql> select State, count(*) as Total from zipcodes group by state;

+-------+-------+
| State | Total |
+-------+-------+
| AK    |   271 |
| AL    |   863 |
| AR    |   737 |
| AS    |     1 |
| AZ    |   538 |
| CA    |  2716 |
    ( ... )
```

Okay, that was a long list — more than 50 entries (the ZIP code table includes U.S. possessions as well as states), and in no useful order. Let's rewrite the query to make the result easier to understand:

- Q: Sort the list from least to greatest number of ZIP codes.

```
mysql> select State, count(*) as Total from zipcodes group by state order by Total;

+-------+-------+
| State | Total |
+-------+-------+
| AS    |     1 |
| VI    |    16 |
| RI    |    94 |
| DE    |   102 |
    ( ... )
| IL    |  1627 |
| PA    |  2269 |
| NY    |  2281 |
| CA    |  2716 |
| TX    |  2743 |
```

```
+-------+-------+
```

1. The phrase "order by Total" means to sort the list by the total amounts, smallest to largest. To get the reverse order, use the keyword DESC: "order by Total desc".
2. The first two listed "state" abbreviations are for U.S. possessions, not states — they represent American Samoa and the Virgin Islands. The first listed state is Rhode Island, the Pluto of American states.

Let's say we want to make the list shorter by excluding some entries:

- Q: Only list "states" with total ZIP codes less than 150.

```
mysql> select State, count(*) as Total from zipcodes group by state having Total < 150 order by
Total;

+-------+-------+
| State | Total |
+-------+-------+
| AS    |     1 |
| VI    |    16 |
| RI    |    94 |
| DE    |   102 |
| HI    |   148 |
+-------+-------+
```

- Q: Only list "states" with totals greater than 100 and less than 300.

```
mysql> select State, count(*) as Total from zipcodes group by state having (Total > 100 and Total
< 300) order by Total;

+-------+-------+
| State | Total |
+-------+-------+
| DE    |   102 |
| HI    |   148 |
| WY    |   199 |
| PR    |   206 |
| NV    |   232 |
| AK    |   271 |
| DC    |   276 |
+-------+-------+
```

1. These last two examples were meant to show uses of HAVING, an obscure but useful MySQL operator used in conjuction with GROUP BY.
2. Note the use of parentheses around the second HAVING example: "having (Total > 100 and Total < 300)".

Here are more questions for the ZIP code database:

- Q: Which city has the most ZIP codes?

```
mysql> select City,count(*) as Total from zipcodes group by City order by Total,City;

+----------------------------+-------+
|            City            | Total |
+----------------------------+-------+
        (end of a long list ...)
| Austin                     |    90 |
| San Antonio                |    94 |
| Kansas City                |    96 |
| Los Angeles                |   101 |
| Miami                      |   104 |
| Sacramento                 |   108 |
| Springfield                |   111 |
| Atlanta                    |   121 |
| Dallas                     |   132 |
| El Paso                    |   159 |
| New York                   |   167 |
| Houston                    |   191 |
| Washington                 |   306 |
```

```
+----------------------------+-------+
```

Such a result can be misleading -- there is more than one city called "Washington," as explained here.

- Q: which cities have the longest names?

```
mysql> select City, State, length(City) as Length from zipcodes group by City order by Length;

+----------------------------+-------+--------+
|            City            | State | Length |
+----------------------------+-------+--------+
            (end of a long list ...)
| Aberdeen Proving Ground    | MD    |     23 |
| Holloman Air Force Base    | NM    |     23 |
| Whiteman Air Force Base    | MO    |     23 |
| Tripler Army Medical Ctr   | HI    |     24 |
| High Rolls Mountain Park   | NM    |     24 |
| Fairchild Air Force Base   | WA    |     24 |
| Kings Canyon National Pk   | CA    |     24 |
| Petrified Forest Natl Pk   | AZ    |     24 |
| Mesa Verde National Park   | CO    |     24 |
| South International Falls  | MN    |     25 |
| Hot Springs National Park  | AR    |     25 |
| White Sands Missile Range  | NM    |     25 |
| King And Queen Court Hous  | VA    |     25 |
| Yellowstone National Park  | WY    |     25 |
| Little Rock Air Force Base | AR    |     26 |
+----------------------------+-------+--------+
```

- Q: Shortest names?

```
mysql> select City, State, length(City) as Length from zipcodes group by City order by Length desc;

+----------------------------+-------+--------+
|            City            | State | Length |
+----------------------------+-------+--------+
            (end of a long list ...)
| Ono                        | PA    |      3 |
| Ulm                        | MT    |      3 |
| Gay                        | WV    |      3 |
| May                        | OK    |      3 |
| Day                        | FL    |      3 |
| Loa                        | UT    |      3 |
| Max                        | MN    |      3 |
| Art                        | TX    |      3 |
| Iva                        | SC    |      3 |
| Eva                        | AL    |      3 |
+----------------------------+-------+--------+
```

Note about the last example that the order of the sort was reversed using the keyword DESC.

Topical Links

- MySQL (Wikipedia) — overview, history
- MySQL Workbench (Wikipedia) — a graphical database design tool and front end for MySQL
- JDBClient — a Java-based database client

— Navigate this multi-page article with the arrows and drop-down lists at the top and bottom of each page —

Home | MySQL | MySQL Tutorial 5: SQL Examples Share This Page