

MySQL Tutorial 1: Overview, Tables, Queries

Getting to know the MySQL database engine

— [P. Lutus](#) — [Message Page](#) —

Copyright © 2012, [P. Lutus](#)

[Introduction](#) | [First Steps](#) | [Advanced Database Queries 1: People](#)
[Advanced Database Queries 2: ZIP Codes](#) | [Summary](#) | [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 .

Introduction

MMySQL is a free, open-source database engine available for all major platforms . (Technically, MySQL is a [relational database management system \(RDBMS\)](#) .) MySQL represents an excellent introduction to modern database technology, as well as being a reliable mainstream database resource for high-volume applications.

A modern database is an efficient way to organize, and gain access to, large amounts of data. A [relational database](#) is able to create relationships between individual database elements, to organize data at a higher level than a simple table of records, avoid data redundancy and enforce relationships that define how the database functions.

These articles will familiarize the reader with basic database concepts, move on to MySQL, and show how to create a useful database, step by step.

First Steps

Basic Database Terminology

- MySQL creates, configures, and communicates with databases.
- A database is an organized collection of data.
- At its simplest, a database will consist of tables like this:

Name	Age	Favorite Color
Bruce Callow	13	I haven't decided yet.
Frank Wright	37	Red .. no, wait ...
Seymour Hawthorne	82	None of your business.

- Tables contain *records* (sometimes called *rows*), and records contain *fields* (sometimes called *columns*) :

	<i>Field 1</i>	<i>Field 2</i>	<i>Field 3</i>
<i>Field Names:</i>	Name	Age	Favorite Color
<i>Record 1</i>	Bruce Callow	13	I haven't decided yet.
<i>Record 2</i>	Frank Wright	37	Red .. no, wait ...
<i>Record 3</i>	Seymour Hawthorne	82	None of your business.

The above is the basic premise of a simple database table, even a real one having millions of records. More exotic databases link tables together relationally, so that specific values that might appear in hundreds of tables only need to be defined in one (a topic to be covered later).

I've always thought that direct experience is a good way to inspire confidence in one's abilities, so we'll start out by creating a database and a table at the keyboard, by direct interaction with MySQL.

(Because the procedure for installing MySQL is different on each platform, and because the installation process is

nearly automatic in modern times, I've decided not to cover installation issues. At this point I assume the reader has successfully installed MySQL and has logon authority in the system.)

Start the MySQL command-line application

I'll make this one concession to the differences between platforms:

- Windows: Start -> All Programs -> MySQL -> MySQL Server (version) -> MySQL (version) Command Prompt
- Linux, OSX: Open a shell session and type "mysql --user=(username) --password=(password)"

For the above steps, the outcome should be the MySQL command-line prompt, awaiting your entries:

```
mysql> |
```

Make a Database

First, let's make sure we're not creating a database that already exists — we'll do this by listing the names of existing databases (user entries are in green):

```
mysql> show databases;
```

Type the **green text** above and press Enter. If your MySQL installation is new, the list should be short:

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| performance_schema |
| test      |
+-----+
```

Now enter this:

```
mysql> create database tutorial;
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| performance_schema |
| test      |
| tutorial   |
+-----+
```

So we've succeeded in creating a database to hold the content of this tutorial. Now let's try generating an error — what happens if we try to create a database that already exists? Type this:

```
mysql> create database tutorial;
ERROR 1007 (HY000): Can't create database 'tutorial'; database exists
```

I should tell you that's one of the more informative MySQL error messages — most messages require the user to figure out what went wrong.

Now that we have created our database, let's tell MySQL to use it for subsequent entries:

```
mysql> use tutorial;
```

NOTE: There's an alternative to telling MySQL which database to use — one simply prepends the database name to each table reference, like this: "database_name.table_name". When dealing with multiple databases, this is often more convenient.

Make a Table

In serious database work, one spends a fair amount of time in advance of table creation deciding what fields each table record should have, and what data types the fields should contain. But for this first effort, we'll limit ourselves to a simple table. *In the entry below I have included a common error, to make the reader aware of an issue having to do with spaces.* Watch what happens:

```
mysql> create table trivia (Name text, Age integer, Favorite Color text);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Color text)' at line 1
```

Apart from showing the first of the dreaded and uninformative MySQL error messages, what's wrong? Well, think about this — can MySQL figure out which parts of our entry are field names, and which are data types? The first field name is "Name", and it has the data type "text". The field name and data type are separated by a space. But the field name "Favorite Color" has a space within it, and MySQL can't decide whether "Color" is part of the name or a data type, and there's no data type called "Color". So — error message, and no table.

Here is the solution — enclose problem field names in backticks (` `):

```
mysql> create table trivia (Name text, Age integer, `Favorite Color` text);
mysql> describe trivia;
```

Field	Type	Null	Key	Default	Extra
Name	text	YES		NULL	
Age	int(11)	YES		NULL	
Favorite Color	text	YES		NULL	

The result produced by "describe (tablename)" has these fields:

- "Field": field name
- "Type": the data type supported by this field
- "Null": whether this field may be NULL, or undefined.
- "Key": whether this field represents a table key (see below)
- "Default": what value this field should have by default
- "Extra": special properties appear here

Okay, on examining this description, it seems obvious that our table shouldn't allow blank fields, so let's try again, this time requiring specific values for each field:

```
mysql> drop table trivia;
mysql> create table trivia (Name text not null, Age integer not null, `Favorite Color` text not null);
mysql> describe trivia;
```

Field	Type	Null	Key	Default	Extra
Name	text	NO		NULL	
Age	int(11)	NO		NULL	
Favorite Color	text	NO		NULL	

NOTE: Do be careful with commands like "drop (tablename)". This table happens to be empty, but the same command works with a table having a million records — it just goes away with no warning.

Now we have a table ready to accept records describing people by name, age and favorite color. It's time to insert some real data. *Again, I have deliberately introduced an error:*

```
mysql> insert into trivia values(Bruce Callow,13,I haven't decided yet.);
```

The above example is truly pathological in its errors, and I won't try to even list the possible outcomes, depending on what the reader types to try to recover. But here's the correct entry:

```
mysql> insert into trivia values('Bruce Callow','13','I haven't decided yet.);
```

```
mysql> select * from trivia;
+-----+-----+-----+
| Name          | Age | Favorite Color |
+-----+-----+-----+
| Bruce Callow  | 13  | I haven't decided yet. |
+-----+-----+-----+
```

Let's take a closer look at the corrected values for this successful entry:

- The name "Bruce Callow" is a string, a sequence of characters, so it must be enclosed in quotes of one kind or another. Examples: 'Bruce Callow', "Bruce Callow", but not `Bruce Callow` (backticks are reserved for field, table and database names).
- The age "13" is a number, so it doesn't need to be quoted. But quoting a number is not an error, and in some cases, for consistency, all values might be enclosed in quotes .
- The string "I haven't decided yet" contains an apostrophe. Unfortunately the apostrophe is also sometimes used as a quotation mark, which will confuse MySQL about where the string ends. One solution is to use double quotation marks around this specific field: "I haven't decided yet". Another solution is to "escape" the apostrophe in one of these ways: 'haven't', 'haven\t' — either will work.

NOTE: To save time and typing, readers are encouraged to copy the green entries from the example boxes and paste them into your MySQL command-line session:

- In this display, drag your mouse cursor across the phrase you want to capture, highlighting it.
- Press Ctrl+C or select menu item "Edit ... Copy".
- In the MySQL command-line window, select "Paste" (how to do that unfortunately depends on the platform).
- Press Enter.

Okay, let's add two more records to our table:

```
mysql> insert into trivia values('Frank Wright',37,'Red .. no, wait ...');
mysql> insert into trivia values('Seymour Hawthorne',82,'None of your business. ');
mysql> select * from trivia;
+-----+-----+-----+
| Name          | Age | Favorite Color |
+-----+-----+-----+
| Bruce Callow  | 13  | I haven't decided yet. |
| Frank Wright  | 37  | Red .. no, wait ...   |
| Seymour Hawthorne | 82  | None of your business. |
+-----+-----+-----+
```

NOTE: If something goes wrong and you want to start over entering records into the table, you may want to delete all existing records first. Here's how:

```
mysql> delete from trivia;
mysql> select * from trivia;
Empty set (0.00 sec)
```

Again, as with "delete (tablename)", this is a risky command that requires caution. And please notice that "delete **from** trivia" (note the bold word) deletes the records and keeps the table, but "delete trivia" deletes the table and its data.

At this point, let's assume we have successfully entered three records into the table. Now we can perform our first meaningful query:

```
mysql> select * from trivia where Age = 13;
+-----+-----+-----+
| Name          | Age | Favorite Color |
+-----+-----+-----+
| Bruce Callow  | 13  | I haven't decided yet. |
+-----+-----+-----+
```

Now let's try this:

```
mysql> select * from trivia where Age < 40;
```

```

+-----+-----+-----+
| Name          | Age | Favorite Color |
+-----+-----+-----+
| Bruce Callow  | 13  | I haven't decided yet. |
| Frank Wright  | 37  | Red .. no, wait ... |
+-----+-----+-----+

```

Queries can be very specific (note the use of `backticks` around the field name):

```

mysql> select `Favorite Color` from trivia where Age < 40;

+-----+-----+
| Favorite Color |
+-----+-----+
| I haven't decided yet. |
| Red .. no, wait ... |
+-----+-----+

```

Now let's display our table sorted by name:

```

mysql> select * from trivia order by name;

+-----+-----+-----+
| Name          | Age | Favorite Color |
+-----+-----+-----+
| Bruce Callow  | 13  | I haven't decided yet. |
| Frank Wright  | 37  | Red .. no, wait ... |
| Seymour Hawthorne | 82  | None of your business. |
+-----+-----+-----+

```

That didn't work out so well — Frank Wright should have appeared last. The reason he didn't is because MySQL sorted the names just as we entered them — as a single string composed of first name, then last name. At this point it may occur to the reader that name data should always be entered into separate fields for first and last name. The underlying idea is that one can always combine a first and last name to create a full name, but one cannot easily do the reverse.

Split a Field in Two, Make a New Table

This is slightly more ambitious — we're going to create a new table with separate fields for first and last name, then we're going to transfer all the records from the original table into the new one. The point of this exercise is to show how to do this without having to re-enter the records. Here's the procedure:

As a first step, before creating a new table, let's see if we can successfully split the "Name" field into first and last names:

```

mysql> select substring_index(Name,' ',1) as FirstName, substring_index(Name,' ',-1) as LastName, Age,
`Favorite Color` from trivia order by LastName;

+-----+-----+-----+
| FirstName | LastName | Age | Favorite Color |
+-----+-----+-----+
| Bruce    | Callow   | 13  | I haven't decided yet. |
| Seymour  | Hawthorne | 82  | None of your business. |
| Frank    | Wright   | 37  | Red .. no, wait ... |
+-----+-----+-----+

```

It seems the "Name" field was split up correctly, and because we then specified "order by LastName", the name "Frank Wright" appeared last as it should. Now let's create a new table with the names split in the same way:

```

mysql> create table triviab (FirstName text not null, LastName text not null, Age integer not null,
`Favorite Color` text not null);
mysql> describe triviab;

+-----+-----+-----+
| Field          | Type      | Null | Key | Default | Extra |
+-----+-----+-----+
| FirstName      | text      | NO   |     | NULL    |       |
| LastName       | text      | NO   |     | NULL    |       |
| Age            | int(11)   | NO   |     | NULL    |       |
+-----+-----+-----+

```

Favorite Color	text	NO	NULL
----------------	------	----	------

All right — now let's transfer the records from "trivia" to "triviab", splitting the names as we go:

```
mysql> insert into triviab select substring_index(Name,' ',1) as FirstName, substring_index(Name,' ',-1)
as LastName, Age, `Favorite Color` from trivia;
mysql> select * from triviab;
```

FirstName	LastName	Age	Favorite Color
Bruce	Callow	13	I haven't decided yet.
Frank	Wright	37	Red .. no, wait ...
Seymour	Hawthorne	82	None of your business.

The above is an example of creating a new table and providing it with records from another table, but split up and arranged differently. This isn't always necessary — in many cases one can write a query for an existing table to get different results without having to create a new table each time. Examples:

```
mysql> select * from triviab order by LastName, FirstName;
```

FirstName	LastName	Age	Favorite Color
Bruce	Callow	13	I haven't decided yet.
Seymour	Hawthorne	82	None of your business.
Frank	Wright	37	Red .. no, wait ...

```
mysql> select * from triviab order by Age desc;
```

FirstName	LastName	Age	Favorite Color
Seymour	Hawthorne	82	None of your business.
Frank	Wright	37	Red .. no, wait ...
Bruce	Callow	13	I haven't decided yet.

The above keyword "desc" means "descending", in this context meaning sort the records in the reverse of the natural sort order.

Advanced Database Queries 1: People

For more advanced query practice, I decided a table with only three records wasn't adequate, so I've created a much larger table composed of records describing make-believe people. Readers have two ways to get this table into their tutorial database:

- Download and load into your MySQL database the MySQL-formatted table description file [people.sql](#).
- Download and run the original Python program [generate_people_table.py](#) I used to create the table, and run it locally.

The first of the two options above is simpler, and is preferred. Here is how to proceed:

- Right-click here: [people.sql](#) and select context menu option "Save link as ..." or "Save Target as ..." (this differs with your browser).
- Download the file to any convenient location.
- Make a copy of the file in a directory that's easily located by the MySQL command-line application.
- From the MySQL application, type this:

```
mysql> use tutorial;
mysql> source (path)/people.sql;
mysql> describe people;
```

Field	Type	Null	Key	Default	Extra
First Name	text	NO		NULL	

```

+-----+-----+-----+-----+-----+
| Last Name | text | NO | | NULL |
| Gender | enum('M','F') | NO | | NULL |
| Age | int(11) | NO | | NULL |
| Music | text | NO | | NULL |
| Art | text | NO | | NULL |
| pk | int(11) | NO | PRI | NULL | auto_increment
+-----+-----+-----+-----+-----+
mysql> select count(*) from people;
+-----+
| count(*) |
+-----+
| 1000 |
+-----+

```

The purpose of the last command "select count(*) from people" is to verify that all 1000 records from the "people" table were successfully read.

In this table's description, we see there's a field named "pk" with a new role we haven't seen before — as a "primary key". As tables become larger, the presence of a unique primary key becomes more important for a number of reasons. One, MySQL can read from and write to the table more efficiently. Two, it makes each record unique — without a unique key, two records with identical field data would not be distinct (MySQL would not be able to tell them apart).

There's another, even more important role for table keys in the design of relational databases, a topic we'll be getting into later. But back to our new table — with 1000 records, we can pose some interesting queries. Here are some examples.

- Let's say you're responsible for a company database, and your boss shows up and asks, "How many of our customers are (a) interested in rock music, and (b) age 20 and younger? Being fully versed in modern database technology, you create an answer for him in a matter of seconds:

```

mysql> select count(*) from people where Age <= 20 and Music = 'Rock';
+-----+
| count(*) |
+-----+
| 23 |
+-----+

```

NOTE: Because I generated the "people" table using a seeded pseudorandom number generator, the numbers you acquire from your invocation of the "people" table should agree with those here. This also allows you to compare your results with the original ones, as an accuracy check for your query entries.

- The boss thinks for a minute and asks, "Can you show me the breakdown by musical preference for that age group?" You reply this way:

```

mysql> select count(*),Music from people where Age <= 20 group by Music;
+-----+-----+
| count(*) | Music |
+-----+-----+
| 25 | Baroque |
| 23 | C & W |
| 28 | Classical |
| 33 | Jazz |
| 23 | Rock |
+-----+-----+

```

- The boss next asks, "Can you show that breakdown in terms of percentages of the age group as well?" Here's your reply:

```

mysql> set @total = (select count(*) from people where Age <= 20);
mysql> select count(*),count(*) * 100/@total as '%',Music from people where Age <= 20 group by Music;
+-----+-----+-----+
| count(*) | % | Music |
+-----+-----+-----+
| 25 | 18.9394 | Baroque |

```

23	17.4242	C & W
28	21.2121	Classical
33	25.0000	Jazz
23	17.4242	Rock

- The boss, difficult to impress, asks, "Okay, how about the same breakdown for the age range between 20 and 50?" You reply:

```
mysql> set @total = (select count(*) from people where Age > 20 and Age <= 50);
mysql> select count(*),count(*) * 100/@total as '%',Music from people where Age > 20 and Age <= 50 group by Music;
```

count(*)	%	Music
103	20.5589	Baroque
90	17.9641	C & W
117	23.3533	Classical
87	17.3653	Jazz
104	20.7585	Rock

- The boss looks away for an uncomfortably long time, then gets a funny look on his face and says, "Women, over 60, who listen to Country & Western Music — go!" You reply with:

```
mysql> select count(*) from people where Gender = 'F' and Age > 60 and Music = 'C & W';
```

count(*)
14

- It's now clear that the boss is on some kind of roll — he says, "Breakdown artistic taste for that selection — go!" You reply:

```
mysql> select count(*), Art from people where Gender = 'F' and Age > 60 and Music = 'C & W' group by Art;
```

count(*)	Art
2	Abstract
1	Impressionist
3	Modern
4	Realist
1	Romantic
3	Surrealist

- The boss now says, "List the full record for the impressionist fan in that group — go!" Your reply:

```
mysql> select * from people where Gender = 'F' and Age > 60 and Music = 'C & W' and Art = 'Impressionist';
```

First Name	Last Name	Gender	Age	Music	Art	pk
Patricia	Brown	F	67	C & W	Impressionist	116

- At this point your boss leans over conspiratorially and says, "Print that out, we never had this conversation, and you're getting a raise."

Advanced Database Queries 2: ZIP Codes

The above exercise used a fairly large database of 1000 records. But this example uses a much larger U.S. ZIP code

database having over 43,000 records. I include this query example because it shows that one can extract many interesting facts from a large table with many fields.

We will proceed as in the previous example, but with one possible change — this database is so large that users may elect to download a zipped version of the file:

- Download [zipcodes.zip](#), and uncompress the file using your platform's unzip program. This will yield a file named "zipcodes.sql".
- (Those with fast Internet connections may prefer to download the original, uncompressed archive [zipcodes.sql](#) .)
- As before, place the file in a location convenient to the MySQL command-line application.
- From the MySQL application, type this (replace "(path)" with the path to the SQL archive file):

```
mysql> use tutorial;
mysql> source (path)/zipcodes.sql;
mysql> describe `zipcodes`;

+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| zip   | char(5) | NO | PRI | NULL |  |
| city  | char(20) | NO |  | NULL |  |
| state | char(4) | NO |  | NULL |  |
| latitude | double | NO |  | NULL |  |
| longitude | double | NO |  | NULL |  |
| timezone | int(11) | NO |  | NULL |  |
| dst   | int(11) | NO |  | NULL |  |
+-----+-----+-----+-----+-----+
mysql> select count(*) from `zipcodes`;

+-----+
| count(*) |
+-----+
| 43191 |
+-----+
```

Now for some fun. Before I had access to databases, I would ponder questions I knew I would never be able to answer, like "Which U.S. city has the most ZIP codes?" But one quickly discovers that, armed with a ZIP code table, MySQL can answer this question with a single-line query:

```
mysql> select `city`,count(*) as `total` from `zipcodes` group by `city` having count(*) > 90 order by `total`;

+-----+-----+
| city | total |
+-----+-----+
| 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 |
+-----+-----+
```

Wow — Washington, D.C. has 306 ZIP codes? But this result doesn't actually mean what it seems. If there is more than one city named "Washington", this query won't reveal it, instead it groups all cities with the same name and disregards the fact that they might lie in different states (because that's what I told MySQL to do — as they say, *garbage in, garbage out*). Let's write a more specific query:

```
mysql> select `city`,`state`,count(*) as `total` from `zipcodes` group by `city`,`state` having count(*) > 80 order by `total`;

+-----+-----+-----+
| city | state | total |
+-----+-----+-----+
```

Kansas City	MO	81
San Diego	CA	81
Austin	TX	83
Philadelphia	PA	84
Chicago	IL	85
San Antonio	TX	90
Miami	FL	96
Los Angeles	CA	101
Sacramento	CA	105
Atlanta	GA	111
Dallas	TX	122
El Paso	TX	156
New York	NY	166
Houston	TX	182
Washington	DC	276

Okay, this time Washington's total is 276, not 306. That means there are 30 "Washington" ZIP codes elsewhere. Can we find them? Well, in a word, yes:

```
mysql> select `city`,`state`,count(*) as `total` from `zipcodes` where city = 'Washington' group by
`city`,`state` order by `total`;
```

city	state	total
Washington	AR	1
Washington	VA	1
Washington	NH	1
Washington	OK	1
Washington	IA	1
Washington	WV	1
Washington	ME	1
Washington	TX	1
Washington	IL	1
Washington	NC	1
Washington	VT	1
Washington	UT	1
Washington	GA	1
Washington	CT	1
Washington	CA	1
Washington	KS	1
Washington	MS	1
Washington	NJ	1
Washington	NE	1
Washington	KY	1
Washington	PA	1
Washington	LA	1
Washington	MI	2
Washington	MO	2
Washington	IN	4
Washington	DC	276

Okay, wow. "Washington" seems to be a very popular city/town name (remembering that the totals are not for distinct towns but ZIP codes). In fact, if we count total ZIP codes as a measure of popularity, it's the most popular town name in the U.S.

I once wanted to know how many other towns had my town name (which at the time was Ashland). Here's the result:

```
mysql> SELECT `city`,`state`,count(*) as `total` FROM `zipcodes` WHERE `city` = "ashland" group by
`state` order by `total`,`state`;
```

city	state	total
Ashland	AL	1
Ashland	IL	1
Ashland	KS	1
Ashland	LA	1
Ashland	MA	1

Ashland	MO	1
Ashland	MS	1
Ashland	NE	1
Ashland	NH	1
Ashland	NY	1
Ashland	OH	1
Ashland	OR	1
Ashland	PA	1
Ashland	VA	1
Ashland	WI	1
Ashland	ME	2
Ashland	MT	2
Ashland	KY	4

Again, these are totals of ZIP codes, not towns. I suspect that there aren't four towns named "Ashland" in Kentucky, but one town with four ZIP codes.

Let's try something else. Let's see if we can collect all of Colorado's ZIP codes, not by the obvious approach of querying on the state's name, but on something more esoteric — the state's geographical dimensions. Here are the locations of Colorado's four corners, in decimal degrees (which I just got from [Google Earth](#), a first-rate program):

- Colorado:

Name	Latitude	Longitude
NW corner	41.000497	-109.050149
NE corner	41.002380	-102.051881
SE corner	36.993237	-102.041959
SW corner	36.999037	-109.045220

Now, just to (a) do something I've always wanted to try, and (b) show a more complex query, here's one that uses the above geographical information to search within a rectangular region:

```
mysql> SELECT `city`,`state`,count(*) as `total` FROM `zipcodes` WHERE ( `latitude` >= 37 AND
`latitude` <= 41 ) AND ( `longitude` >= -109 AND `longitude` <= -102 ) group by `state` order by
`total`,`state`;
```

city	state	total
Kanorado	KS	2
Aurora	CO	679

"Kanorado?" All right — my acute intellectual powers tell me this town is located right on the border between Kansas and Colorado, and that rounding off the query's geographical coordinates to integers might have been a mistake. Let's see where Kanorado is located ... [Map](#) ... yes, it's right next to the border (the vertical line at the left on the linked map). Now let's see if we can eliminate this interloper and perfect our result:

```
mysql> SELECT `city`,`state`,count(*) as `total` FROM `zipcodes` WHERE ( `latitude` >= 37 AND
`latitude` <= 41 ) AND ( `longitude` >= -109 AND `longitude` <= -102.1 ) group by `state` order by
`total`,`state`;
```

city	state	total
Aurora	CO	679

Okay, adjusting Colorado's eastern border seems to have solved that problem. Now I want to make sure I haven't missed any ZIP codes that belong to Colorado:

```
mysql> SELECT `state`,count(*) as total from `zipcodes` where `state` = 'CO';
```

state	total
-------	-------

city	state	zip	latitude	longitude
Gateway	CO	680		

Wups. It seems that, with my chosen geographical coordinates, I managed to miss one of Colorado's 680 ZIP codes. Let's discover which one:

```
mysql> select `city`,`state`,`zip`,`latitude`,`longitude` from `zipcodes` where `state` = 'CO' and
`zip` not in (SELECT `zip` FROM `zipcodes` WHERE ( `latitude` >= 37 AND `latitude` <= 41 ) AND (
`longitude` >= -109 AND `longitude` <= -102.1 ));
```

city	state	zip	latitude	longitude
Gateway	CO	81522	38.715101	-109.01087

Ah — that explains it. The ZIP code database has an incorrect position for Gateway, CO, which is actually located at West longitude 108.97519, not 109.01087 ([map of Gateway](#)). What this means is that, as written, even with integer latitudes and longitudes (with the single "Kanorado" exception above), my query would have produced an exclusive list of Colorado ZIP codes. It also serves to prove the old adage "the map is not the territory."

But back to the topic of MySQL queries. Notice about the above query that I actually wrote two queries (one of them is called a "subquery"), and used one as a filter against the other. The above rather complex query in essence says:

Give me results that meet specification A but don't meet specification B.

Let's write one more query — let's turn the above query around and write it so it produces all ZIP codes *except* those from Colorado. First let's adjust the query to accommodate the wrong Gateway position and give 680 ZIP codes:

```
mysql> SELECT `state`,count(*) FROM `zipcodes` WHERE ( `latitude` > 37 AND `latitude` < 41 ) AND
( `longitude` > -109.1 AND `longitude` < -102.1 ) group by `state`;
```

state	count(*)
CO	680

Now we want to write the *logically opposite* query, one that will produce all ZIP codes *except* those from Colorado. Earlier we found that the ZIP code database has 43,191 records, so our result should produce 43,191 - 680 = 42,511 ZIP codes. Let's see:

```
mysql> SELECT count(*) FROM `zipcodes` WHERE ( `latitude` <= 37 OR `latitude` >= 41 ) OR (
`longitude` <= -109.1 OR `longitude` >= -102.1 );
```

count(*)
42511

Success. But notice how this query differs from the first. A comparison:

- To select only Colorado, we said, "choose all locations with latitudes (> 37 **and** < 41) **and** longitudes (> -109 **and** < -102)". Our query defined an *inclusive* rectangle.
- To select *everything except* Colorado, we said, "choose all locations with latitudes (<= 37 **or** >= 41) **or** longitudes (<= -109 **or** >= -102)". This time we defined an *exclusive* rectangle.
- In the descriptions above, pay particular attention to the use of the words "**and**" and "**or**", and how their use differs between the examples. Think about how the logic plays out. Notice also how parentheses are used to group logical tests and clarify the meaning of logical expressions.

Summary

Feel free to experiment with more query designs using the "people" and "zipcodes" tables. You can't hurt anything -- the tables can be recreated by simply recreating the MySQL content files using the MySQL command-line program as was done above.



At this point the reader should have acquired a basic grounding in database operations, from table design up through

relatively advanced query design. In the next page of this article we'll look a ways to store queries in views, and automate certain kinds of actions that can be triggered by table updates.

Topical Links

- [MySQL \(Wikipedia\)](#) — overview, history
- [MySQL Workbench \(Wikipedia\)](#) — a graphical database design tool and front end for MySQL
- [JDBCClient](#) — a free, powerful, Java-based database client program

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

[Home](#) | [MySQL](#) |   [Share This Page](#)