Favourite MySQL Queries

by Peter Lavin

Overview

Being a web developer requires that you have a variety of skills and perform a number of different jobs. There may be long periods when you are working with a variety of different databases and then extended stretches when you are not working with any database at all. Being a Jack-of-All-Trades means that SQL syntax may not always be at your fingertips. For this reason a reference of useful MySQL queries comes in handy.

Getting Started

The basic syntax of a SELECT query even with a WHERE clause a GROUP BY and any other permutation you might want to think of, is not too difficult to remember and does not vary significantly across different dialects of SQL. Likewise with INSERT and UPDATE queries. But other queries, especially those that you do not use on a regular basis, can easily slip from memory. This article will deal with three basic types of queries, data definition, date manipulation and some miscellaneous queries. Creating a table can be a fairly complex matter, date manipulation is not simple or intuitive in any programming language and miscellaneous queries are hard to remember precisely because they are miscellaneous.

Data Definition

Let's start where it all has to start – by looking at data definition and throwing in a few metadata queries along the way. Creating a database is fairly simple but the intricacies of table creation can easily slip your mind. Because the tables we create often have similarities with tables we have created in the past, one option would be to save copies of the data definition statements used when creating those tables; but why do what has already been done for you? USE the database where the table resides and execute the query, "SHOW CREATE TABLE tblname;". This statement will give the exact SQL necessary to create this table. Remember this one simple statement and you can immediately have access to the SQL needed to recreate any table. Here's an example of what might be returned:

```
CREATE TABLE `tblbooks` (
`inventorynumber` int(11) NOT NULL auto_increment,
`cat` char(3) NOT NULL default '',
`catalogue` varchar(50) default NULL,
`isbn` varchar(10) default NULL,
`title` varchar(150) NOT NULL default '',
`author` varchar(100) NOT NULL default '',
`condition` enum('Very Fine','Fine','Near Fine','Very
Good','Good','Fair','Poor','Ex-Library','Book Club','Binding
Copy') default 'Near Fine',
`binding` enum('Hardcover','Paper') default 'Hardcover',
`pagecount` int(11) default NULL,
`description` varchar(255) default NULL,
`signed` tinyint(4) NOT NULL default '0',
`firstedition` tinyint(4) NOT NULL default '0',
`jacketed` tinyint(4) NOT NULL default '0',
`jacketcondition` enum('Very Fine','Fine','Near Fine','Very
Good','Good','Fair','Poor') default NULL,
`deficiencies` varchar(255) default NULL,
`cost` float(6,2) NOT NULL default '0.00',
`publicationdate` varchar(4) default NULL,
`publisher` varchar(4) NOT NULL default '',
`image` varchar(20) default NULL,
`sold` tinyint(4) NOT NULL default '0',
PRIMARY KEY (`inventorynumber`),
KEY `authidx` (`author`),
KEY `titleidx` (`title`),
KEY `sold` (`sold`)
) TYPE=MyISAM;
```

Okay, it's not the simplest table structure but you can quickly see, for example, how to create a float field suitable for currency, the syntax for "enum", how to create indices and a number of other things.

Seeing how a table is created may also help you determine how it might be improved. There are a few things that might be changed in the above table and changing the structure of a table is another thing that is easy to forget. Resist the urge to use the keyword UPDATE. You can only UPDATE data not a table's structure. What you want to do is ALTER the table. In the table above the "jacketcondition" field really should have a default value. This can be changed in the following way:

```
ALTER TABLE tblbooks
  CHANGE jacketcondition jacketcondition
  ENUM('Very Fine','Fine','Near Fine','Very
Good','Good','Fair','Poor')
  DEFAULT 'Near Fine';
```

Sorry, but you do have to duplicate all that information about the field just to set a default value.

Some of us get things right the first time around but I often find that I need to add fields to a table after initially creating it. Our "tblbooks" really should have a field that shows the retail price. Here's how to add it:

```
ALTER TABLE tblbooks
  ADD COLUMN listprice float(6,2) NOT NULL default '0.00'
  AFTER cost;
```

You don't just want to add fields but you want to place them at the right position within the table. Without the AFTER clause the newly added field would be the last field whereas, its more natural position is after the "cost" field.

We don't always notice how our tables might be improved but fortunately there is a query to remedy that: (Note the British spelling of the function ANALYSE() unlike the spelling of the ANALYZE keyword.)

```
SELECT *
  FROM `tblbooks`
  PROCEDURE ANALYSE();
```

My editor will think I'm trying to pad the word count so I won't display the result set of this query, but it does show actual minimum and maximum values for data in the various fields and makes some useful suggestions about the best field size. The optimal field type is often inappropriately suggested as ENUM but, all in all, this is a useful query to have under your belt. Averages and standard deviations are also shown so this is a good way to get a quick overview of your data.

Given that versions of MySQL prior to 4 cannot use nested SQL statements knowing the syntax for creating temporary tables using a SELECT statement is also useful.

```
CREATE TEMPORARY TABLE  tbltemp
 SELECT
    CONCAT( fname, " " ,lname) AS Name,
    courses.name AS Course,
    CEIL(SUM(weight.weight * mark)-.5) AS Course_Average
    FROM marks, weight, students, courses
    WHERE marks.studentid = students.studentid
    AND courses.code = marks.courseid
    AND marks.courseid=weight.courseid
    AND marks.assessment = weight.assessment
    GROUP BY marks.studentid, marks.courseid;
```

After creating this table you can run the secondary query:

```
SELECT Name, AVG(Course_Average)
  FROM tbltemp
  GROUP BY Name;
```

in order to get an overall average mark.

If you want to know the number of records in each table in your database you don't need to use COUNT(*) with every table. Just :

```
SHOW TABLE STATUS;
```

This query gives a quick overview of the state of the tables in your database including the number of rows in each table.

Before we leave data definition we should say something about mysqldump. It's well documented in the MySQL manual but it won't hurt to repeat the basics here. Properly speaking mysqldump isn't a query but a utility. It's a very valuable one for the web developer because usually a local database is created for testing purposes and later uploaded to a web server. The best way to create a copy for upload is:

```
mysqldump --opt databasename -h hostname -u username -p >
databasename.sql
```

After executing a database dump, the file "databasename.sql" will contain all the SQL statements necessary to recreate your database structure and content. If you are using a programme such as phpMyAdmin you can upload it and execute it in an SQL query window. If not, then you can recreate your tables in the following way:

```
mysql databasename -u username -p password -h hostname <
databasename.sql
```

Don't forget to create your database first.

Manipulating Dates

No matter what programming language you are using manipulating date or time fields is always a challenge. Often we require portions of dates or we want a humanly useable version of the date rather than the seemingly arbitrary numbers stored in a database. To extract the month and the day of the month as numbers do the following (assuming of course that "whenadded" is a date type field):

```
SELECT item, MONTH(whenadded)AS mon,
  DAYOFMONTH(whenadded)AS dayofmon
  FROM tblbulletin
  WHERE year(whenadded)=YEAR(NOW())
  ORDER BY whenadded DESC LIMIT 8;
```

This query extracts a numerical representation of the month and day for items added to this table in the current year and is perhaps most useful when mathematical operations need to be performed. If the date is needed for display purposes only, the function date_format is, perhaps, the better one to use.

```
SELECT title, description, topic, username, contents,
  DATE_FORMAT(whenadded,'%M %e,%Y') AS
  formatted
  FROM tblitem WHERE id = 1;
```

Here the date will be returned in the format "February 20, 2005".

Often you will want to do date comparisons, as in the following query:

```
SELECT MONTHNAME(fldwhen) AS fldmonth,
  DAYOFMONTH(fldwhen) AS fldday, YEAR(fldwhen) AS fldyear,
  fldtitle, flddescription
  FROM tblconcerts
  WHERE TO_DAYS(fldwhen) >= TO_DAYS(NOW())
  Order BY fldwhen ASC
  LIMIT 0,4;
```

Using the TO_DAYS() function along with the operator ">=", allows you to show only upcoming concerts and the LIMIT clause ensures that only the four closest to the current date are shown.

Date subtraction can be used to select more recent items in a list. For example:

```
SELECT description, url, target,
  IF(whenadded > (SUBDATE(CURDATE(),
  INTERVAL '14' DAY)), 'hot', 'info') AS cssclass
  FROM tbllinks
  ORDER BY cssclass, UCASE(description)";
```

This query uses "if else" logic to separate out new links and also creates the opportunity to use the field ‘cssclass' as a CSS style name in order to format these new links in a distinctive way.

Times up so here's one last query to select users who have been active in an online chat room within the last 10 minutes:

```
SELECT username FROM tblonline
  WHERE fldwhen > (SUBDATE(CURTIME(),INTERVAL '10' MINUTE));
```

Miscellaneous

Because they aren't really related to anything else miscellaneous queries can be the hardest to remember. While updating a password is not difficult – what comes after can create problems.

```
UPDATE user
  SET password = PASSWORD("reddevils")
  WHERE user = "root";
```

Just don't forget to FLUSH afterwards:

```
FLUSH PRIVILEGES;
```

And don't forget to LOCK your tables if concurrency issues might arise:

```
LOCK TABLES tblbooks WRITE,
  tblorderitem WRITE;
```

It's not particularly difficult to remember how to alias tables but I always want to include an AS so I'll give an example here:

```
SELECT l.description, l.url, l.target, lt.description
  AS heading
  FROM links l, linktypes lt
  WHERE l.typeid = lt.id
  ORDER BY lt.id, UCASE(l.description)
```

The syntax for setting variables is a bit idiosyncratic and that makes it a good candidate for inclusion. Here's how to set one:

```
SET @invid :=1;
```

Referencing that variable is exactly what you would expect:

```
SELECT CONCAT(e.firstname, " ", e.lastname) AS name,
  w.description, (i.hours * w. rate) AS billing, c.companyname
  FROM  invoiceitem i, worktypes w, employees e, companies c,
  invoices inv, domains d
  WHERE i.worktype = w.id AND i.invoicenumber = @invid
  AND d.domainname = inv.domainname
  AND d.companyid = c.id;
```

Finally, a last query to show how to randomly select your quotation of the day:

```
SELECT * FROM quotations
  ORDER BY RAND()
  LIMIT 0, 1;
```

Memory

It often seems that for every new thing we learn something else is forgotten. Writing things down helps in remembering them but it sometimes happens that even "auld acquaintance" are forgot, so I know that I will be referring back to this document at some time. Here's hoping it proves as useful for you in remembering "auld lang syne".

About the Author

Peter Lavin runs a Web Design/Development firm in Toronto, Canada. For more information visit http://www.softcoded.com. He may be reached at peterlavin@sympatico.ca.