UNIT III

OPEN SOURCE DATABASE

MySQL: Introduction – Setting up account – Starting, terminating and writing your Own SQL programs – Record selection Technology – Working with strings – Date and Time– Sorting Query Results.

MySQL:

Introduction:

MySQL is the most popular open source SQL database management system (DBMS), is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers.

A database is a collection of data that is organized so that its contents can be easily accessed, managed and updated. MySQL is a data storage area. In this storage area, there are small sections called Tables. A Relational Database Management System (RDBMS) may be a DBMS in which data is stored in the form of tables and the relationship among the data is also stored in the form of tables.

The data in MySQL is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows. The MySQL database has become the world's most popular open source database because of its consistent fast performance, high reliability and ease of use.

Advantages:

- *MySQL is Cross-Platform. It runs on more than 20 platforms including Linux, Windows, Mac OS, Solaris, HP-UX, IBM AIX, giving you the kind of flexibility that puts you in control.*
- *MySQL is fast. It is used by a variety of corporations that demand performance and stability.*
- *MySQL is free. It is Open Source software. As such you are free to examine the source code and make any changes you wish. As per its GPL license, you are free to redistribute those changes as long as your software is also Open Source.*
- Reliable and easy to use.
- Multi-Threaded multi-user and robust SQL Database server.

Disadvantages:

- Missing Sub-selects.
- MySQL doesn't yet support the Oracle SQL extension.
- Does not support Stored Procedures and Triggers.
- MySQL doesn't support views, but this is on the TODO.

Setting up Account:

In order to provide access the MySQL database you need to create an account to use for connecting to the MySQL server running on a given host.Use the GRANT statement to set up the MySQL user account. Then use that account's name and password to make connections to the server.User names, as used by MySQL for

authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix MySQL user names can be up to 16 characters long.

MySQL passwords have nothing to do with passwords for logging in to your operating system. Connecting to a MySQL server requires a username and password. You can also specify the name of the host where the server is running. If you don't specify connection parameters explicitly, *mysql* assumes default values. For example, if you specify no hostname, *mysql* typically assumes the server is running on the local host.

Adding User Accounts:

You can create MySQL accounts in two ways:

- By using statements intended for creating accounts, such as CREATE USER or GRANT. These statements cause the server to make appropriate modifications to the grant tables.
- By manipulating the MySQL grant tables directly with statements such as INSERT, UPDATE, or DELETE.

Account Management Statements:

CREATE USER:

The CREATE USER statement creates new MySQL accounts.

Syntax: CREATE USER user [IDENTIFIED BY [PASSWORD] 'password'] [, user [IDENTIFIED BY [PASSWORD]'password']] ...

Example:

CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';

DROPUP USER:

The DROP USER statement removes one or more MySQL accounts and their privileges.

Syntax: DROP USER user [,user]...

Example:

DROP USER 'jeffrey'@'localhost';

GRANT : The GRANT statement enables system administrators to grant privileges to MySQL user accounts.

Syntax : GRANT priv_type [(column_list)]
[, priv_type [(column_list)]] ... ON [object_type] priv_level
TO user [IDENTIFIED BY [PASSWORD] 'password']
[, user [IDENTIFIED BY [PASSWORD] 'password']] ... [REQUIRE {NONE | ssl_option [[AND] ssl_option] ...}] [WITH with_option ...]

Example : GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost' WITH GRANT OPTION;

RENAME USER:

The RENAME USERstatement renames existing MySQL accounts.

Syntax : RENAME USER old_user TO new_user

[, old_user TO new_user] ...

Example RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';

REVOKE:The REVOKE statement enables system administrators to revoke privileges from MySQL accounts.

Syntax: REVOKE priv_type [(column_list)]

[, priv_type [(column_list)]] ... ON [object_type] priv_level FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION

Example: RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';

SET PASSWORD :

The SET PASSWORD statement assigns a password to an existing MySQL user.

Syntax

SET PASSWORD [FOR user] =

{

Example:

PASSWORD('some password')

| OLD_PASSWORD('some password')

| 'encrypted password'

}

SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');

Starting, terminating and writing your Own SQL programs:

Starting MySQL:

o The MySQL server can be started manually from the command line.

o To start the mysqld server from the command line, you should start a console window (or "DOS window") and enter this command:

shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"

o The path to mysqld may vary depending on the install location of MySQL on your system.

Login:

- To start the *mysql* program, type *myql* at your command-line prompt.
- If *mysql*starts up correctly, you'll see a short message, followed by a mysql> prompt that indicates the program is ready to accept queries.
- To illustrate, here's what the welcome message looks like (to save space, I won't show it in any further examples):

```
% mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 18427 to server version: 3.23.51-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

- If *mysql* tries to start but exits immediately with an "access denied" message, you'll need to specify connection parameters.
- The most commonly needed parameters are the host to connect to (the host where the MySQL server runs), your MySQL username, and a password.
  - For example:

```
% mysql -h localhost -p -u cbuser
Enter password: cbpass
```

Stopping MySQL:

You can stop the MySQL server by executing this command:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" -u
root shutdown
```

Logout:

To terminate a *mysql* session, issue a QUIT statement:

```
mysql> QUIT
```

You can also terminate the session by issuing an EXIT statement or (under Unix) by typing Ctrl-D.

Writing your Own SQL programs:

When existing software doesn't do what you want, you can write your own programs. We are going to create a simple login system using PHP code on our pages, and a MySQL database to store our user's information.

Sample Coding

```
<?php
    $host="localhost"; // Host name
    $username=""; // Mysql username
    $password=""; // Mysql password
    $db_name="test"; // Database name
    $tbl_name="members"; // Table name
    // Connect to server and select databse.
    mysql_connect("$host", "$username", "$password")or die("cannot connect");
    mysql_select_db("$db_name")or die("cannot select DB");
    // username and password sent from form
```

```
$myusername=$_POST['myusername'];
$mypassword=$_POST['mypassword'];
$sql="SELECT * FROM $tbl_name WHERE username='$myusername' and
password='$mypassword'";
$result=mysql_query($sql);
// Mysql_num_row is counting table row$count=mysql_num_rows($result);
// If result matched $myusername and $mypassword, table row must be 1 row
if($count==1)
  {
// Register $myusername, $mypassword and redirect to file "login_success.php"
session_register("myusername");
session_register("mypassword");
header("location:login_success.php");
}
  else
echo "Wrong Username or Password";
}
    ?>
```

Record selection Technology:

The SELECT statement is used to select data from a database. The statement begins with the SELECT keyword. The basic SELECT statement has 3 clauses:

- SELECT
- FROM
- WHERE

The SELECT clause specifies the table columns that are retrieved. The FROM clause specifies the tables accessed. The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

For example,

3.5.1 SELECT name FROM emp WHERE city='Rome'

This query accesses rows from the table emp. It then filters those rows where the *city* column contains Rome. Finally, the query retrieves the *name* column from each filtered row. Using the example *s* table, this query produces:

```
+-----------------+
|      name       |
+-----------------+
| aravindhan      |
```

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

SELECT gives you control over several aspects of record retrieval:

- Which table to use
- Which columns to display from the table
- What names to give the columns
- Which rows to retrieve from the table
- How to sort the rows

Checking what version of the server you're running or the name of the current database:

```
mysql> SELECT VERSION( ), DATABASE( );
+------------+-----------+
| VERSION( ) | DATABASE( )|
+------------+-----------+
| 3.23.51-log | cookbook |
+------------+-----------+
```

Specifying Which Columns to Display

To display some or all of the columns from a table. Use * as a shortcut that selects all columns. Or name the columns you want to see explicitly.

For example,

```
mysql> SELECT * FROM emp;
+--------------------+---------+---------+
|       name         |  age    |  exp    |
+--------------------+---------+---------+
|    AnandKumar      |  45     |  12     |
+--------------------+---------+---------+
mysql> SELECT name,age FROM emp;
+--------------------+---------+
|       name         |  age    |
+--------------------+---------+
|    AnandKumar      |  45     |
+--------------------+---------+
```

Giving Names to Output Columns:

Whenever you retrieve a result set, MySQL gives every output column a name. (That's how the *mysql* program gets the names that you see displayed as the initial row of column headers in result set output.) MySQL assigns default names to output columns, but if the defaults are not suitable, you can use column aliases to specify your own names.

```
mysql> SELECT name,age FROM emp;
```

```
+-------------------+--------+
|       name        |  age   |
+-------------------+--------+

|     AnandKumar    |  45    |
+-------------------+--------+
```

mysql> SELECT Emp.Name as name, Emp.Age as age FROM emp;

```
+-------------------+--------+
|     Emp.Name      | Emp.Age |
+-------------------+--------+
|     AnandKumar    |  45    |
+-------------------+--------+
```

Specifying Which Rows to Select:

Add a WHERE clause to the query that indicates to the server which rows to return. Unless you qualify or restrict a SELECT query in some way, it retrieves every row in your table, which may be a lot more information than you really want to see.

mysql> SELECT name,age FROM emp WHERE name='aravind';

```
+-------------------+--------+
|       name        |  age   |
+-------------------+--------+
|       aravind     |  45    |
+-------------------+--------+
```

Removing Duplicate Rows:

Use DISTINCT a query eliminate duplicate values. Some queries produce results containing duplicate records.

For example,

mysql> SELECT name FROM emp;

```
+--------+
|  name  |
+--------+
|aravind |
|anand   |
|Naveen  |
|siva    |
|araivnd |
|anand   |
```

+---------+

But that result is heavily redundant. Adding DISTINCT to the query removes the duplicate records, producing a set of unique values:

mysql> **SELECT name FROM emp;**

```
+---------+
   name  |
+---------+
| aravind |
| anand   |
| Naveen  |
| siva    |
+---------+
```

Working with NULL Values:

To summarizing a set of values that may include NULL values and you need to know how to interpret the results. Most aggregate functions ignore NULL values.

mysql> **SELECT subject, test, score FROM expt ORDER BY subject, test;**

```
+---------+------+-------+
| subject | test | score |
+---------+------+-------+
| Arun    |   A  |   47  |
| Chitra  |   B  |   50  |
| David   |   C  |  NULL |
| Ram     |   D  |  NULL |
+---------+------+-------+
```


mysql> **SELECT subject name FROM expt WHERE score IS NULL GROUP BY subject;**

```
+---------+
| Name    |
+---------+
| David   |
| Ram     |
+---------+
```

Sorting a Result Set:

Add an ORDER BY clause to tell it exactly how you want things sorted. When you select rows, the MySQL server is free to return them in any order, unless you instruct it otherwise by saying how to sort the result. There are lots of ways to use sorting techniques.

mysql> **SELECT name,age FROM emp WHERE age > 24 ORDER BY name;**

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

```
| name  | age  |
+--------+------+
| Arun   |  35  |
| Ram    |  45  |
| David  |  34  |
| Chitra |  56  |
+--------+------+
```

mysql> **SELECT name,age FROM emp WHERE age > 24 ORDER BY name ASC;**

```
+--------+------+
| name   | age  |
+--------+------+
| Arun   |  35  |
| Chitra |  45  |
| David  |  34  |
| Ram    |  56  |
+--------+------+
```

Working with Strings:

A string can be binary or nonbinary. Binary strings are used for raw data such as images, music files, or encrypted values. Nonbinary strings are used for character data such as text and are associated with a character set and collation (sorting order). A character set determines which characters are legal in a string. Collations can be chosen according to whether you need comparisons to be case-sensitive or case-insensitive, or to use the rules of a particular language.

Data types for binary strings are BINARY, VARBINARY, and BLOB. Data types for nonbinary strings are CHAR, VARCHAR, and TEXT, each of which allows CHARACTER SETand COLLATEattributes. See Recipe 5.2 for information about choosing data types for string columns.

You can convert a binary string to a nonbinary string and vice versa, or convert a nonbinary string from one character set or collation to another.

You can use a string in its entirety or extract substrings from it. Strings can be combined with other strings.

You can apply pattern-matching operations to strings. FULLTEXTsearching is available for efficient queries on large collections of text.

Strings properties:

- Strings can be case sensitive (or not), which can affect the outcome of string operations.
- You can compare entire strings or just parts of them by extracting substrings.
- You can apply pattern-matching operations to look for strings that have a certain structure.

Types of Strings

Binary *data* may contain bytes that lie outside the usual range of printable ASCII characters.A binary *string* in MySQL is one that MySQL treats as case sensitive in comparisons.For binary strings, the characters A and a are considered different. For non-binary strings, they're considered the same.

A binary column type is one that contains binary strings. Some of MySQL's column types are binary (case sensitive) and others are not, as illustrated here:

Column type	Binary/case sensitive
CHAR, VARCHAR	No
CHAR BINARY, VARCHAR BINARY	Yes
TEXT	No
BLOB	Yes
ENUM,SET	No

A characteristic of nonbinary strings is that they have a character set. To see which character sets are available, use this statement:

mysql> **SHOW CHARACTER SET;**

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
...			

The default character set in MySQL is **latin1.**

The following example illustrates how collation affects sort order. Suppose that a table contains a latin1 string column and has the following rows:

```
mysql> CREATE TABLE t (c CHAR(3) CHARACTER SET latin1);
mysql> INSERT INTO t (c) VALUES('AAA'),('bbb'),('aaa'),('BBB');
mysql> SELECT c FROM t;
  +------+
  | C |
  +-------+
```

| AAA |

| bbb |

| aaa |

| BBB |

+-------+

String Data Types:

- CHAR
- VARCHAR
- TINYTEXT
- TEXT
- BLOB
- MEDIUMTEXT
- LONGTEXT
- BINARY
- VARBINARY
- ENUM
- SET

CHAR():

It is a fixed length string and is mainly used when the data is not going to vary much in it's length. It ranges from 0 to 255 characters long. While storing CHAR values they are right padded with spaces to the specified length. When retrieving the CHAR values, trailing spaces are removed.

VARCHAR() :

It is a variable length string and is mainly used when the data may vary in length. It ranges from 0 to 255 characters long. VARCHAR values are not padded when they are stored.

Note: If CHAR and VARCHAR options are used in the same table, then MySQL will automatically change the CHAR into VARCHAR for compatability reasons. The ( ) bracket allows to enter a maximum number of characters that will be used in the column.

TINYTEXT, TINYBLOB :

A string with a maximum length of 255 characters.

TEXT :

TEXT columns are treated as character strings(non-binary strings). It contains a maximum length of 65535 characters.

BLOB :

BLOB stands for Binary Large OBject. It can hold a variable amount of data. BLOB columns are treated as byte strings(binary strings). It contains a maximum length of 65535 characters.

MEDIUMTEXT, MEDIUMBLOB:

It has a maximum length of 16777215 characters.

LONGTEXT, LONGBLOB :

It has a maximum length of 4294967295 characters.

BINARY :

The BINARY is similar to the CHAR type. It stores the value as binary byte strings instead of non- binary character strings.

VARBINARY :

The VARBINARY is similar to the VARCHAR type. It stores the value as binary byte strings instead of non-binary character strings.

ENUM() :

An enumeration. Each column may have one of a specified possible values. It can store only one of the values that are declared in the specified list contained in the ( ) brackets. The ENUM list ranges up to 65535 values.

SET() :

A set. Each column may have more than one of the specified possible values. It contains up to 64 list items and can store more than one choice. SET values are represented internally as integers.

Choosing a String Data Type

Choose the data type according to the characteristics of the information to be stored and how you need to use it. Consider questions such as these:

- Are the strings binary or nonbinary?
- Does case sensitivity matter?
- What is the maximum string length?
- Do you want to store fixed- or variable-length values?
- Do you need to retain trailing spaces?
- Is there a fixed set of allowable values?

MySQL provides several binary and nonbinary string data types. These types come in pairs as shown in the following table.

Binary data type	Nonbinary data type	Maximum length
BINARY	CHAR	255
VARBINARY	VARCHAR	65,535
TINYBLOB	TINYTEXT	255
BLOB	TEXT	65,535
MEDIUMBLOB	MEDIUMTEXT	16,777,215

LONGBLOB	LONGTEXT	4,294,967,295

Date and Time Data Types:

MySQL supports a number of date and time column formats. For all the date and time columns, we can also assign the values using either string or numbers.

- o DATE
- o TIME
- o DATETIME
- o TIMESTAMP
- o YEAR

DATE:

A Date. The range is 1000-01-01 to 9999-12-31. The date values are displayed in YYYY-MM-DD format.

TIME:

A Time. The range is -838:59:59 to 838:59:59. The time values are displayed in HH:MM:SS format.

DATETIME:

A Date and Time combination. The range is 1000-01-01 00:00:00 to 9999-12-31 23:59:59. The date time values are displayed in YYYY-MM-DD HH:MM:SS format.

TIMESTAMP:

A Timestamp. The range is 1970-01-01 00:00:01 UTC to partway through the year 2037. A TIMESTAMP column is useful for recording the date and time of an INSERT or UPDATE operation.

YEAR:

A Year. The year values are displayed either in two-digit or four-digit format. The range of values for a four-digit is 1901 to 2155. For two-digit, the range is 70 to 69, representing years from 1970 to 2069.

For Example

```
mysql> SELECT t1, t2 FROM time_val;
+----------+----------+
|    t1    |    t2    |
+----------+----------+
| 15:00:00 | 15:00:00 |
| 05:01:30 | 02:30:20 |
| 12:30:20 | 17:30:45 |
+----------+----------+
mysql> SELECT d FROM date_val;
+------------+
|     d      |
+------------+
```

| 1864-02-28 |
| 1900-01-15 |
| 1987-03-05 |
| 1999-12-31 |
| 2000-06-04 |
+------------+

Date and Time Formats:

It is most common to store dates using a dash (-) as the delimiter and a colon (:) as the time delimiter it is in fact possible to use any character, or no character between the date and time segments. For example, the following formats all achieve the same result:

2008-10-23 10:37:22

20081023103722

2008/10/23 10.37.22

2008*10*23*10*37*22

Date and Time Functions:

In addition to providing mechanisms for storing dates and times, MySQL also provides a wide range of functions that can be used to manipulate dates and times. The following table provides a list of the more common functions available for working with times and dates in MySQL:

Function	Description
ADDDATE()	Add dates
ADDTIME()	Add time
CONVERT_TZ()	Convert from one timezone to another
CURDATE()	Returns the current date
CURTIME()	Returns the current system time
DATE_ADD()	Add two dates
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract two dates
DATE()	Extract the date part of a date or datetime expression
DATEDIFF()	Subtract two dates
DAYNAME()	Returns the name of the weekday
DAYOFMONTH()	Returns the day of the month (1-31)
DAYOFWEEK()	Returns the weekday index of the argument
DAYOFYEAR()	Returns the day of the year (1-366)

EXTRACT	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format date as a UNIX timestamp
GET_FORMAT()	Returns a date format string
HOUR()	Extract the hour
LAST_DAY	Returns the last day of the month for the argument
MAKEDATE()	Create a date from the year and day of year
MAKETIME	MAKETIME()
MICROSECOND()	Returns the microseconds from argument
MINUTE()	Returns the minute from the argument
MONTH()	Returns the month from the date passed
MONTHNAME()	Returns the name of the month
NOW()	Returns the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Returns the number of months between two periods
QUARTER()	Returns the quarter from a date passed as an argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Returns the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBTIME()	Subtract times
SYSDATE()	Returns the time at which the function executes
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Returns the argument converted to seconds
TIME()	Extract the time portion of the expression passed as an
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the Date or the Datetime expression. With two arguments, the
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Returns the date argument converted to days

UNIX_TIMESTAMP()	Returns a UNIX timestamp to a format acceptable to
TC_DATE()	Returns the current Universal Time (UTC) date
UTC_TIME()	Returns the current Universal Time (UTC time
UTC_TIMESTAMP()	Returns the current Universal Time (UTC) date and
WEEK()	Returns the week number
WEEKDAY()	Returns the weekday index
WEEKOFYEAR()	Returns the calendar week of the date (1-53)
YEAR()	Returns the year
YEARWEEK()	Returns the year and week

Display Dates or Times:

Use the DATE_FORMAT( ) or TIME_FORMAT( ) functions to rewrite the DATE_FORMAT( ) function, which takes two arguments: a DATE, DATETIME, or TIMESTAMP value, and a string describing how to display the value.

Within the formatting string, you indicate what to display using special sequences of the form %*c*, where *c* specifies which part of the date to display.

For example, %Y, %M, and %d signify the four-digit year, the month name, and the two-digit day of the month.

The following query shows the values in the date_val table, both as MySQL displays them by default and as reformatted with DATE_FORMAT( ):

```
mysql> SELECT d, DATE_FORMAT(d,'%M %d, %Y') FROM date_val;
+------------+-------------------------+
|    d       |DATE_FORMAT(d,'%M %d, %Y')|
+------------+-------------------------+
| 1864-02-28 |    February 28, 1864    |
| 1900-01-15 |    January 15, 1900     |
| 1987-03-05 |    March 05, 1987       |
| 1999-12-31 |    December 31, 1999    |
| 2000-06-04 |    June 04, 2000        |
+------------+-------------------------+
```


Sorting Query Results:

SQL **SELECT** command to fetch data from MySQL table. When you select rows, the MySQL server is free to return them in any order, unless you instruct it otherwise by saying how to sort the result. But a query doesn't come out in the order you want.

Using ORDER BY to Sort Query Results:

Add an ORDER BY clause. ORDER BY will tell the MySQL server to sort the rows by a column. Define in which direction to sort, as the order of the returned rows may not yet be meaningful. Rows can be returned in ascending or descending order.

Use ASC or DESC. Using ASC will sort the data so that you see the smallest number first. Using DESC will sort the data so that you see the largest number first. In this query, you are looking for customers with the largest negative balance first. ORDER BY will return the arrays with the greatest negative number (the smallest number) at the top.

Syntax:

Here is generic SQL syntax of SELECT command along with ORDER BY clause to sort data from MySQL table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
ORDER BY field1, [field2...] [ASC [DESC]]
```

- You can sort returned result on any field provided that filed is being listed out.
- You can sort result on more than one field.
- You can use keyword ASC or DESC to get result in ascending or descending order. By default its ascending order.
- You can use WHERE...LIKE clause in usual way to put condition.

Example:

```
mysql> SELECT * from tutorials_tbl ORDER BY tutorial_author ASC
+-------------+---------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+---------------+-----------------+-----------------+
|           2 | Learn MySQL   | Abdul S         | 2007-05-24      |
|           1 | Learn PHP     | John Poul       | 2007-05-24      |
|           3 | JAVA Tutorial | Sanjay          | 2007-05-06      |
+-------------+---------------+-----------------+-----------------+
```


Sorting Subsets of a Table

You don't want to sort an entire table, just part of it. Add a WHERE clause that selects only the records you want to see.

ORDER BY doesn't care how many rows there are; it sorts whatever rows the query returns. If you don't want to sort an entire table, add a WHERE clause to indicate which rows to select. For example, to sort the records for just one of the drivers, do something like this:

```
mysql> SELECT trav_date, miles FROM driver_log WHERE name = 'Henry'
```

ORDER BY trav_date;

```
+------------+-------+
| trav_date | miles |
+------------+-------+
| 2001-11-26 | 115 |
| 2001-11-27 | 96 |
| 2001-11-29 | 300 |
| 2001-11-30 | 203 |
| 2001-12-01 | 197 |

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

Displaying One Set of Values While Sorting by Another

To sort a result set using values that you're not selecting. You can use columns in the ORDER BY clause that doesn't appear in the column output list. ORDER BY is not limited to sorting only those columns named in the column output list. It can sort using values that are "hidden".

Sorting and NULL Values

To sort columns that may contain NULL values. If NULL values don't come out in the desired position within the sort order, False them into appearing where you want. When a sorted column contains NULL values, MySQL puts them all together in the sort order.

mysql> **SELECT NULL = NULL;**

```
+-------------+
| NULL = NULL |
+-------------+
| NULL |
+-------------+
```


Normally, sorting puts the NULL values at the beginning:

mysql> **SELECT val FROM t ORDER BY val;**

```
+------+
| val |
+------+
| NULL |
| NULL |
| 3 |
| 9 |
| 100 |
+------+
```

Date-Based Sorting

To sort in temporal order. Sort using a date or time column type, ignoring parts of the values that are irrelevant if necessary. Many types of information include date or time information and it's very often

necessary to sort results in temporal order. MySQL knows how to sort temporal column types, so there's no special trick to ordering values in DATE, DATETIME, TIME, or TIMESTAMP columns. Begin with a table that contains values for each of those types:

```
mysql> SELECT * FROM temporal_val;
+------------+---------------------+----------+---------------+
| d          | dt                  | t        | ts            |
+------------+---------------------+----------+---------------+
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 19800101020000 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 20210101030000 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 19750101040000 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 19850101050000 |
+------------+---------------------+----------+---------------+
```

Sorting by Calendar Day

To sort by day of the calendar year. Sort using the month and day of a date, ignoring the year. Sorting in calendar order differs from sorting by date. You ignore the year part of the dates and sort using only the month and day to order records in terms of where they fall during the calendar year. Suppose you have an event table that looks like this when values are ordered by actual date of occurrence:

```
mysql> SELECT date, description FROM event ORDER BY date;
+------------+-----------------------------------+
| date       | description                       |
+------------+-----------------------------------+
| 1215-06-15 | Signing of the Magna Carta        |
| 1732-02-22 | George Washington's birthday      |
| 1776-07-14 | Bastille Day                      |
| 1789-07-04 | US Independence Day               |
| 1989-11-09 | Opening of the Berlin Wall        |
+------------+-----------------------------------+
```