Basic MySQL Tutorial

TABLE OF CONTENTS

Chapter 1

Managing Databases in MySQL

Summary: In this tutorial you will learn how to manage database in MySQL. You will learn how to create a new database, list all databases in the MySQL database server and remove databases from database catalog.

Let's start creating a new database in MySQL.

Creating Database

Before doing anything else with data, you need to create a database. A database is a container of data. It stores contact, vendor or customer or any kind of data you can think of. In MySQL, a database is a collection of objects that are used to store and manipulate data such as tables, stored procedures...etc. To create a database in MySQL, you use **CREATE DATABASE** statement as follows:

```
CREATE DATABASE [IF NOT EXISTS] database_name;
```

Let's examine the CREATE DATABASE statement in a greater details:

- Followed by CREATE DATABASE statement is database name you want MySQL to create. It is recommended that database name should be meaningful and descriptive.
- IF NOT EXISTS is an optional part of the statement. The IF NOT EXISTS statement prevents you from error if there is a database with the same name exists in the database catalog. Of course you cannot have 2 databases with the same name in a same database catalog.

For example, to create *classicmodels* database, you just need to execute the CREATE DATABASE statement above as follows:

```
CREATE DATABASE classicmodels;
```

After executing the statement, MySQL will returns you a message to notify the new database created successfully or not.

Showing Databases Statement

SHOW DATABASE statement shows all databases in your database server. You can use SHOW DATABASE statement to check the database you've created or to see all the databases on the database server before you create a new database, for example:

```
SHOW DATABASES;
```

In my database server, the output is :

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| classicmodels      |
| mysql              |
+--------------------+
```

```
8 rows in set (0.00 sec)
```

Selecting database to work with

Before working with database you must tell MySQL which database you want to work with. MySQL provides you USE statement to help you to do so.

```
USE database_name;
```

You can select our sample database by using the USE statement as follows:

```
USE classicmodels;
```

From now on all actions you are performing will affect current database such as querying data, create new table or stored procedure.

Removing Database

Removing database means you delete the database physically. All the data and related objects inside the database are permanently deleted and cannot be undone. So it is very important to execute this query with cares. To delete a database you can use DROP DATABASE statement, which is a standard SQL statement as well.

```
DROP DATABASE [IF EXISTS] database_name;
```

You need to specify database name after the DROP DATABASE statement. Like CREATE DATABASE statement, IF EXIST is an optional part to prevent you from removing database which does not exist in database catalog.
If you want to practice with DROP DATABASE statement, you can first create a new temporary database, make sure that it is created and remove it. The sequence of SQL query to execute is as follows:

```
CREATE DATABASE IF NOT EXISTS temp_database;
SHOW DATABASES;
DROP DATABASE IF EXISTS temp_database;
```

In this tutorial, you've learned how to create a new database, drop an existing database, select the database you want to work with and list all databases in database catalog.

MySQL Sample Database

We will use the *Classic Models* database, a retailer of scale models of classic cars, as the sample database to help you to work with MySQL quickly. The sample database contains typical business data such as customers, products,sale orders, sale order line items and etc. It is used in our MySQL tutorials to demonstrate many features of MySQL.
You can install MySQL in your local PC for practicing.

You can download the sample database in the following download section:

sampledatabase.zip

After unzip **sampledatabase.zip** file, you can load the sample database into MySQL database server.

Loading a sample Database

1. Copy the sample database onto the server at a randomly chosen path say /home

2. CREATE a database as following.

```
mysql> create database classicmodels;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| classicmodels      |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.01 sec)
```

2. Go to the selected pat where the database file is copied

```
[root@testrhel ~]# cd /home/
[root@testrhel home]# ll
total 224
drwxr-xr-x 4 root root   4096 Aug  7 15:03 hmp
drwx------ 2 root root  16384 Aug  5 13:57 lost+found
-rw-r--r-- 1 root root 199833 Aug  5  2008 sampledatabase.sql
drwxr-xr-x 2 root root   4096 Aug 15 19:58 share
```

3. Execute the below command to import the database file.

```
[root@testrhel home]# mysql -u root -p classicmodels<sampledatabase.sql
Enter password:
```

Sample Database Schema

The sample database schema consists of several table as below:

- Customers: Stores customers's data
- Products: Stores a list of scale model cars.
- ProductLines: Stores a list of product line category.
- Orders: Stores orders placed by customers.
- OrderDetails: Stores order line items in each order.
- Payments: Stores payments made by customers based on their account.
- Employees: Stores all employee information include organization unit structure such as who reports to whom.
- Offices: Stores sale office data.

Chapter 2

Understanding MySQL Table Types

Summary: *In this tutorial, you will learn various table types in MySQL. It is essential to know what kind of table types or storage engine you use for your database to maximize the performance of your application.*

MySQL supports various of table types or storage engines to allow you to optimize your database. The table types are available in MySQL are:

- ISAM
- MyISAM
- InnoDB
- BerkeleyDB (BDB)
- MERGE
- HEAP

The most important feature to make all the table types above distinction is transaction-safe or not. Only InnoDB and BDB tables are transaction safe and only MyISAM tables support full-text indexing and searching feature. MyISAM is also the default table type when you create table without declaring which storage engine to use. Here are some major features of each table types:

ISAM

ISAM had been deprecated and removed from version 5.x. All of it functionality entire replace by MyISAM. ISAM table has a hard size 4GB and is not portable.

MyISAM

MyISAM table type is default when you create table. MyISAM table work very fast but not transaction-safe. The size of MyISAM table depends on the operating system and the data file are portable from system to system. With MyISAM table type, you can have 64 keys per table and maximum key length of 1024 bytes.

InnoDB

Different from MyISAM table type, InnoDB table are transaction safe and supports row-level locking. Foreign keys are supported in InnoDB tables. The data file of InnoDB table can be stored in more than one file so the size of table depends on the disk space. Like the MyISAM table type, data file of InnoDB is portable from system to system. The disadvantage of InnoDB in comparison with MyISAM is it take more disk space.

BDB

BDB is similar to InnoDB in transaction safe. It supports page level locking but data file are not portable.

MERGE

Merge table type is added to treat multiple MyISAM tables as a single table so it remove the size limitation from MyISAM tables.

HEAP

Heap table is stored in memory so it is the fastest one. Because of storage mechanism, the data will be lost when the power failure and sometime it can cause the server run out of memory. Heap tables do not support columns with AUTO_INCREMENT, BLOB and TEXT characteristics.

Chapter 3

MySQL Data Types

Summary: *In this tutorial, you will learn various **MySQL data types** to use them effectively in database table design.*

Database table contains multiple columns with specific data types such as numeric or string.

MySQL provides you many more specific data types than just "numeric" or "string". Each data type in MySQL can be determined by the following characteristics:

- What kind of value it can represent.
- The space values take up and whether the values are fixed-length or variable-length.
- The values of a data type can be indexed.
- How MySQL compare values of that data types.

Numeric Data Types

You can find all SQL standard numeric types in MySQL including exact number data type and approximate numeric data types including integer, fixed-point and floating point. In addtion, MySQL also supports BIT data type for storing bit field values. Numeric types can be signed or unsigned except BIT type. The following table shows you the summary of numeric types in MySQL:

Numeric Types	Description
TINYINT	A very small integer
SMALLINT	A small integer
MEDIUMINT	A medium-sized integer
INT	A standard integer
BIGINT	A large integer
DECIMAL	A fixed-point number
FLOAT	A single-precision floating-point number
DOUBLE	A double-precision floating-point number
BIT	A bit field

String Data Types

In MySQL, string can hold anything from plain text to binary data such as images and files. String can be compared and searched based on pattern matching by using LIKE clause or regular expression. The table below shows you the string data types in MySQL:

String Types	Description
CHAR	A fixed-length non-binary (character) string
VARCHAR	A variable-length non-binary string
BINARY	A fixed-length binary string
VARBINARY	A variable-length binary string
TINYBLOB	A very small BLOB (binary large object)
BLOB	A small BLOB
MEDIUMBLOB	A medium-sized BLOB
LONGBLOB	A large BLOB
TINYTEXT	A very small non-binary string
TEXT	A small non-binary string
MEDIUMTEXT	A medium-sized non-binary string
LONGTEXT	A large non-binary string
ENUM	An enumeration; each column value may be assigned one enumeration member
SET	A set; each column value may be assigned zero or more set members

Date and Time Data Types

MySQL provides types for date and time and combination of date and time. In addition, MySQL also provide timestamp data type for tracking last change on a record. If you just want to store the year without date and month, you can use YEAR data type. Here is the table which showing MySQL date and type data types:

Date and Time Types	Description
DATE	A date value in 'CCYY-MM-DD' format
TIME	A time value in 'hh:mm:ss' format
DATETIME	A date and time value in 'CCYY-MM-DD hh:mm:ss' format
TIMESTAMP	A timestamp value in 'CCYY-MM-DD hh:mm:ss' format
YEAR	A year value in CCYY or YY format

Spatial Data Types

MySQL support many spatial data types as below table which contains various kind of geometrical and geographical values.

Spatial Data Types	Description
GEOMETRY	A spatial value of any type
POINT	A point (a pair of X Y coordinates)
LINESTRING	A curve (one or more POINT values)
POLYGON	A polygon
GEOMETRYCOLLECTION	A collection of GEOMETRY values
MULTILINESTRING	A collection of LINESTRING values
MULTIPOINT	A collection of POINT values
MULTIPOLYGON	A collection of POLYGON values

Chapter 4

Understanding MySQL TIMESTAMP

Summary: *The **MySQL TIMESTAMP** is a temporal data type that store combination of date and time values. In this tutorial you will learn how **MySQL TIMESTAMP** values are stored in the database and how to use automatic initialization and automatic update to create created on and last changed on column in a table.*

How MySQL TIMESTAMP stored in the database

The format of MySQL TIMESTAMP column is 'YYYY-MM-DD HH:MM:SS' which is fixed at 19 characters. The MySQL TIMESTAMP are stored as four bytes number of second since the first day of UNIX time which is '1970-01-01 00:00:01'. The upper end of range correspond to maximum four bytes value of UNIX time which is '2038-01-19 03:14:07'. Therefore the TIMESTAMP column has a range of values from '1970-01-01 00:00:01' to '2038-01-19 03:14:07'.

The values of the MySQL TIMESTAMP columns depend on connection's time zone. When insert values for MySQL TIMESTAMP columns, they are converted to Universal Coordinated Time (UTC) from connection's time zone. When you select the value, the server converts it back from UTC to the connection's time zone so you have the same value that you inserted. However if another client with different time zone connects to the server to select value from MySQL TIMESTAMP column, it will see the value adjusted to its time zone. MySQL allows you to change your own time zone when you connect to it so you can see this effect by using a single connection.

Let's try in our sample database to see this effect.

```
CREATE TABLE test_timestamp('t1'  TIMESTAMP);
SET time_zone='+00:00';
INSERT INTO test_timestamp VALUES('2008-01-01 00:00:01');
SELECT t1
FROM test_timestamp;
```

```
+---------------------+
| t1                  |
+---------------------+
| 2008-01-01 07:00:01 |
+---------------------+
1 row in set (0.00 sec)
```

The SQL commands can be explained as follows:

- First we created a table called test_TIMESTAMP.
- Next we set our time zone to UTC.
- Then we insert a TIMESTAMP value into the table test_timestamp.
- Finally we select it to see the value we inserted.

Now can set our time zone to a different time zone and see what value we get from database server:

```
SET time_zone =’+03:00’;

SELECT t1
FROM test_timestamp;

+---------------------+
| t1                  |
+---------------------+
| 2008-01-01 03:00:01 |
+---------------------+
1 row in set (0.00 sec)
```

As you see, we get adjusted value to our new time zone.

INSERT and UPDATE TIMESTAMP column

If you omit the MySQL TIMESTAMP column’s value in the INSERT statement or you set it to NULL, it will be automatically set to current TIMESTAMP. This characteristic of MySQL TIMESTAMP is known as automatic initialization.

In the table which has a MySQL TIMESTAMP column, if you change other columns’ value, the MySQL TIMESTAMP column will be automatic updated to the current TIMESTAMP. The change only accepted if you change its current value to different values in order to have TIMESTAMP column updated. This characteristic of TIMESTAMP called automatic update. Note that only one column can be designated as a TIMESTAMP column which has automatic update.

The MySQL TIMESTAMP columns are design in a table that you need to save the created date and last change date for the records. By applying automatic initialization and automatic update of MySQL TIMESTAMP column you can design the table as follows:

```
CREATE TABLE tbl_name(
            …
            created_on  TIMESTAMP DEFAULT 0
            changed_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      );
```

So when you insert a new record, just omit two TIMESTAMP columns or set them to NULL. The both *created_on* and *changed_on* column will be inserted as the current TIMESTAMP.

When you update, omit both TIMESTAMP columns, the *changed_on* column's value will be automatic updated if there is any change in other columns’ values.

In this tutorial, you've learned how MySQLTIMESTAMP data stored in the MySQL database table and how to use its characteristics to design the *created on* and *last changed on* columns.

Chapter 5

Working with Database Table - Part I

Summary: *In this tutorial you will learn how to create, show and describe tables in MySQL.*

Database table or table in short is one of the most important objects of relational database. How to create databases tables correctly is crucial when you work with any database system especially MySQL. In this tutorial, you will learn how to create a simple database table. You'll also learn how to list all database tables of a specific database.

Creating Tables

To create table we use the **CREATE TABLE** statement. The typical form of SQL CREATE TABLE statement is as follows:

```
CREATE TABLE [IF NOT EXISTS] table_name(
       column_list
       ) type=table_type
```

- MySQL supports IF NOT EXISTS after CREATE TABLE statement to prevent you from error of creating table which already exists on the database server.
- *table_name* is the name of table you would like to create. After that, you can define a set of columns which is usually in this form: column_name data_type(size) [NOT] NULL.
- You can specify the storage engine type you prefer to use for the table. MySQL supports various storage engines such as InnoDB, MyISAM... If you don't explicit declare storage engine type, MySQL will use MyISAM by default.

In our **classicmodels** sample database, to create ***employees*** table, we can use the CREATE TABLE statement as follows:

```
CREATE TABLE employees (
       employeeNumber int(11) NOT NULL,
       lastName varchar(50) NOT NULL,
       firstName varchar(50) NOT NULL,
       extension varchar(10) NOT NULL,
       email varchar(100) NOT NULL,
       officeCode varchar(10) NOT NULL,
       reportsTo int(11) default NULL,
       jobTitle varchar(50) NOT NULL,
       PRIMARY KEY  (employeeNumber)
       );
```

First, you specify table name *employees* after CREATE TABLE statement.
Then you list the columns of the table with their attributes such as data type, size, NOT NULL.
And finally you specify the primary key of the table, in this case the primary key is*employeeNumber*.
If the table has more than one primary key, you can seperate them by a comma. For example, the *payments* table has two primary keys *customerNumber* and *checkNumber*. You can create***payments*** table by executing following query:

```
CREATE TABLE payments (
       customerNumber int(11) NOT NULL,
       checkNumber varchar(50) NOT NULL,
       paymentDate datetime NOT NULL,
       amount double NOT NULL,
       PRIMARY KEY  (customerNumber,checkNumber)
       );
```

It is important to note that by default MySQL uses MyISAM storage engine for the table it created.

Showing and Describing Tables in a Database

In order to show all tables in a database, you use SHOW TABLES statement. By executing the SHOW TABLES statement, MySQL will returns all tables' name of the current selected database that you're working with.

```
SHOW TABLES
```

Here is the output of ***classicmodels*** database:

```
+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| customers               |
| employees               |
| offices                 |
| orderdetails            |
| orders                  |
| payments                |
| productlines            |
| products                |
+-------------------------+
8 rows in set (0.00 sec)
```

In some cases, you need to see the table's metadata, you can use DESCRIBE statement as follows:

```
DESCRIBE table_name;
```

For instance, we can describe employees table like below query:

```
DESCRIBE employees;
```

The output return from the database server:

```
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| employeeNumber | int(11)     | NO   | PRI | NULL    |       |
| lastName       | varchar(50) | NO   |     | NULL    |       |
| firstName      | varchar(50) | NO   |     | NULL    |       |
| extension      | varchar(10) | NO   |     | NULL    |       |
| email          | varchar(100)| NO   |     | NULL    |       |
| officeCode     | varchar(10) | NO   |     | NULL    |       |
| reportsTo      | int(11)     | YES  |     | NULL    |       |
| jobTitle       | varchar(50) | NO   |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
8 rows in set (0.02 sec)
```

Chapter 6

Working with Tables - Part II

Summary: *following with the previous tutorial on how to create database tables in MySQL, in this tutorial, you will learn how to modify and remove existing database tables.*

Altering Table Structures

Besides creating table, MySQL allows you to alter existing table structures with various options. To modify existing database table structure you use the ALTER TABLE statement.
The following illustrates the ALTER TABLE statement syntax:

```
ALTER [IGNORE] TABLE table_name options[, options...]
      options:
            ADD [COLUMN] create_definition [FIRST | AFTER col_name ]
      or    ADD [COLUMN] (create_definition, create_definition,...)
      or    ADD INDEX [index_name] (index_col_name,...)
      or    ADD PRIMARY KEY (index_col_name,...)
      or    ADD UNIQUE [index_name] (index_col_name,...)
      or    ADD FULLTEXT [index_name] (index_col_name,...)
      or    ADD [CONSTRAINT symbol] FOREIGN KEY [index_name]
      (index_col_name,...)
            [reference_definition]
      or    ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
      or    CHANGE [COLUMN] old_col_name create_definition
                  [FIRST | AFTER column_name]
      or    MODIFY [COLUMN] create_definition [FIRST | AFTER col_name]
      or    DROP [COLUMN] col_name
      or    DROP PRIMARY KEY
      or    DROP INDEX index_namr
      or    DISABLE KEYS
      or    ENABLE KEYS
      or    RENAME [TO] new_table_name
      or    ORDER BY col_name
      or    table_options
```

Most of these options are obvious. We will explain some here:

- The CHANGE and MODIFY are the same, they allow you to change the definition of the column or its position in the table.
- The DROP COLUMN will drop the column of the table permanently, if the table contain data all the data of the column will be lost.
- The DROP PRIMARY KEY and DROP INDEX only remove the primary key or index of the column.
- The DISABLE and ENABLE KEYS turn off and on updating indexes for MyISAM table only.
- The RENAME Clause allows you the change the table name to the new one.

Deleting Tables

To delete table from the database, you can use DROP TABLE statement:

```
DROP [TEMPORARY] TABLE [IF EXISTS] table_name [, table_name,...]
```

TEMPORARY keyword is used for deleting temporary tables. MySQL allows you to drop multiple tables at once by listing them and separated each by a comma. IF EXISTS is used to prevent you from deleting table which does not exist in the database.

Empty Table's Data

In some cases, you want to delete all table data in a fast way and reset all auto increment columns. MySQL also provides you SQL TRUNCATE table statement to allow you to do so. The SQL TRUNCATE statement is as follows:

```
TRUNCATE TABLE table_name
```

There are some points you should remember before using TRUNCATE TABLE statement:

- TRUNCATE TABLE statement drop table and recreate it therefore it is much faster than DELETE TABLE statement. However it is not transaction-safe.
- The number of deleted rows is not returned like SQL DELETE TABLE statement.
- ON DELETE triggers are not invoked because TRUNCATE does not use DELETE statement.

Chapter 7

Changing Table Structure Using MySQL ALTER TABLE

Summary:*In this tutorial you will learn how to use **MySQL ALTER TABLE** statement to change the structure of existing tables.*

MySQL ALTER TABLE syntax

MySQL ALTER TABLE statement is used to change the structure of existing tables. You can use MySQL ALTER TABLE to add or drop column, change column data type, add primary key, rename table and a lot more. The following illustrates the MySQL ALTER TABLE syntax:

```
ALTER TABLE table_name action1[,action2,…]
```

Followed by the keyword ALTER TABLE is name of table that you want to make the changes. After the table name is the action you want apply to the table. An action can be anything from add a new column, add primary key…. to rename table. MySQL allows you to do multiple actions at a time, separated by a comma.

Let's create a new table for practicing MySQL ALTER TABLE statement. We're going to create a new table called *tasks* in our sample database *classicmodels* as follows*:*

```
CREATE  TABLE 'tasks' (
      'task_id' INT NOT NULL ,
      'subject' VARCHAR(45) NULL ,
      'start_date' DATETIME NULL ,
      'end_date' DATETIME NULL ,
      'description' VARCHAR(200) NULL ,
      PRIMARY KEY ('task_id') ,
      UNIQUE INDEX 'task_id_UNIQUE' ('task_id' ASC) );
```

Changing columns using MySQL ALTER TABLE statement

Using MySQL ALTER TABLE to add auto-increment for a column

Suppose we want the task id is increased by one automatically whenever we insert a new task. In order to accomplish this, we need to use the MySQL ALTER TABLE statement to change the column task id to make it auto increment as follows:

```
ALTER TABLE tasks
CHANGE COLUMN task_id task_id INT(11) NOT NULL AUTO_INCREMENT;
```

Using MySQL ALTER TABLE to add a new column into a table

Because of the new business requirement, we need to add a new column called *complete* to store completion percentage for each task in the *tasks* table. In this case, we can use MySQL ALTER TABLE to add a new column as follows:

```
ALTER TABLE tasks ADD COLUMN 'complete' DECIMAL(2,1) NULL
AFTER 'description' ;
```

Using MySQL ALTER TABLE to drop a column from a table

Let's say we don't want to store task description in the task table anymore so we have to remove that column. Here is the SQL command to drop a column from a table:

```
ALTER TABLE tasks
DROP COLUMN description ;
```

Renaming table using MySQL ALTER TABLE statement

We can use MySQL ALTER table statement to rename a table. Note that before renaming a table you should take a serious consideration to see its dependencies from database to application level. We can rename our *tasks* table to *work_items* as follows:

```
ALTER TABLE 'tasks'
RENAME TO  'work_items' ;
```

In this tutorial, you've learned how to use MySQL ALTER TABLE statement to change existing table structure and rename table.

Chapter 8

Managing Database Index in MySQL

Summary: *In this tutorial, you will learn how to work with database index in MySQL. You will learn how to use database index properly to speed up the data retrieval and what SQL statements to use to create or drop database index.*

Database indexes help to speed up the retrieval of data from MySQL database server. When retrieving the data from a database table, MySQL first checks whether the index of table exists; If yes it will use index to select exact physical corresponding rows without scanning the whole table.

In general, it is suggested that you should create index on columns you usually use in retrieval such as columns used in join and sorts.Note that all primary keys are in primary index of database table.

Why not index every column? The most significant is that building and maintaining an indexes table take time and storage space on database. In addition, when insert/ update or remove data from table, the index has to be rebuilt and it decrease the performance when changing table data.

Creating Indexes

Usually you create index when creating table. Any column in creating table statement declared as PRIMARY KEY, KEY, UNIQUE or INDEX will be indexed automatically by MySQL. In addition, you can add indexes to the tables which has data. The statement to create index in MySQL is as follows:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
USING [BTREE | HASH | RTREE]
ON table_name (column_name [(length)] [ASC | DESC],...)
```

First you specify the index based on the table types or storage engine:

- UNIQUE means MySQL will create a constraint that all values in the index must be distinct. Duplicated NULL is allowed in all storage engine except BDB.
- FULLTEXT index is supported only by MyISAM storage engine and only accepted columns which have data type is CHAR,VARCHAR or TEXT.
- SPATIAL index supports spatial column and available in MyISAM storage engine. In addition, the column value must not be NULL.

Then you name the index using index types such as BTREE, HASH or RTREE also based on storage engine. Here is the list:

Storage Engine	Allowable Index Types
MyISAM	BTREE, RTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	HASH

Finally you declare which column on which table using the index.
In our sample database, you can create index to *officeCode* column on ***employees*** table to make the join operation with ***office*** table faster. The SQL statement to create index is as follows:

```
CREATE INDEX officeCode ON employees(officeCode)
```

Removing Indexes

Besides creating index, you can also remove index by using DROP INDEX statement. Interestingly, DROP INDEX statement is also mapped to ALTER TABLE statement. Here is the syntax:

```
DROP INDEX index_name ON table_name
```

For example, if you want to drop index ***officeCode*** which we have added to the ***employees*** table, just execute following query:

```
DROP INDEX officeCode ON employees
```

In this tutorial, you've learned how to manage database index of database table in MySQL including creating and removing index.

Chapter 9

Using MySQL SELECT Statement to Query Data

Summary:*In this tutorial, you will learn how to **MySQL SELECT** statement to query data from database tables.*

MySQL SELECT Statement Syntax

In order to retrieve data from MySQL database table you need to use ***MySQL SELECT***statement. The following illustrates MySQL SELECT statment syntax:

```
SELECT column_name1,column_name2...
FROM tables
[WHERE conditions]
[GROUP BY group
[HAVING group_conditions]]
[ORDER BY sort_columns]
[LIMIT limits];
```

The MySQL SELECT statement has many optional elements that you can use. The order of FROM, WHERE, GROUP BY, HAVING, ORDER BY and LIMIT has to be in the sequence above.

To select all columns in a table you can use asterisk (*) notation instead of listing all column names in the MySQL SELECT statement. For example, if you need to query all the columns in*offices* table, you can use the following query:

```
SELECT * FROM employees
```

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	NULL	President
1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep
1188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1143	Sales Rep
1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143	Sales Rep
1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143	Sales Rep
1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143	Sales Rep

The *MySQL SELECT* statement also allows you to to view partial data of a table by listing columns' name after the SELECT keyword. This is called *projection*. For example if you need to view only *first name*, *last name* and *job title* of employee in the *employees* table, you can use the following query:

```
SELECT lastname,firstname,jobtitle
FROM employees
```

lastname	firstname	jobtitle
Murphy	Diane	President
Patterson	Mary	VP Sales
Firrelli	Jeff	VP Marketing
Patterson	William	Sales Manager (APAC)
Bondur	Gerard	Sale Manager (EMEA)
Bow	Anthony	Sales Manager (NA)
Jennings	Leslie	Sales Rep
Thompson	Leslie	Sales Rep
Firrelli	Julie	Sales Rep
Patterson	Steve	Sales Rep
Tseng	Foon Yue	Sales Rep
Vanauf	George	Sales Rep

WHERE Clause

WHERE clause of the MySQL SELECT statement enables you to select particular rows which match its conditions or search criteria. You use WHERE clause to filter the records based on a certain conditions. For example, you can find the president of company by using the following query:

```
SELECT firstname,lastname,email
FROM employees
WHERE jobtitle="president"
```

firstname	lastname	email
Diane	Murphy	dmurphy@classicmodelcars.com

DISTINCT

With DISTINCT keyword, you can eliminate the duplicate records from the SELECT statement. For example, to find how many *job title* of all employees in the *employees* table, you use DISTINCT keyword in SELECT statement as follows:

```
SELECT DISTINCT jobTitle FROM employees;
```

	jobTitle
▶	President
	VP Sales
	VP Marketing
	Sales Manager (APAC)
	Sale Manager (EMEA)
	Sales Manager (NA)
	Sales Rep

Sorting result with ORDER BY

The ORDER BY clause allows you to sort the result set on one or more columns in ascending or descending order. To sort the result set in ascending order you use ASC and in descending order you use DESC keywords. By default, the ORDER BY will sort the result set in ascending order. For example, to sort the name of employees by ***first name*** and ***job title***, you can execute the following query:

```
SELECT firstname,lastname, jobtitle
FROM employees
ORDER BY firstname ASC,jobtitle DESC;
```

	firstname	lastname	jobtitle
▶	Andy	Fixter	Sales Rep
	Anthony	Bow	Sales Manager (NA)
	Barry	Jones	Sales Rep
	Diane	Murphy	President
	Foon Yue	Tseng	Sales Rep
	George	Vanauf	Sales Rep
	Gerard	Hernandez	Sales Rep
	Gerard	Bondur	Sale Manager (EMEA)
	Jeff	Firrelli	VP Marketing
	Julie	Firrelli	Sales Rep
	Larry	Bott	Sales Rep
	Leslie	Jennings	Sales Rep
	Leslie	Thompson	Sales Rep

In this tutorial, you've learned about basic ***MySQL SELECT*** statement to retrieve data from one database table. You'll learn more about each technique in details in later tutorials.

Chapter 10

How to Use MySQL Distinct to Eliminate Duplicate Rows

Summary: *In this tutorial, you will learn how to use **MySQL DISTINCT** with SELECT statement to eliminate duplicate records in the selected result set.*

Sometimes when retrieving data from database table, you get duplicate records which are not expected. In order to remove those duplicate records, you need to use DISTINCT keyword along with SELECT statement. The syntax of SQL DISTINCT is as follows:

```
SELECT DISTINCT columns
FROM table
WHERE where_conditions
```

Let take a look a simple example of using DISTINCT to select unique last name from *employee*table.

```
SELECT lastname
FROM employees
ORDER BY lastname
```

```
+-----------+
| lastname  |
+-----------+
| Bondur    |
| Bondur    |
| Bott      |
| Bow       |
| Castillo  |
| Firrelli  |
| Firrelli  |
| Fixter    |
| Gerard    |
| Hernandez |
| Jennings  |
| Jones     |
| Kato      |
| King      |
| Marsh     |
| Murphy    |
| Nishi     |
| Patterson |
| Patterson |
| Patterson |
| Thompson  |
| Tseng     |
| Vanauf    |
+-----------+
23 rows in set (0.00 sec)
```

Because in the *employees* table we have some employee records with the same last name so we get duplicate last names in the result set. Let apply DISTINCT keyword in the SELECT statement to remove those duplicate last names.

```
SELECT DISTINCT lastname
FROM employees
ORDER BY lastname
```

```
+-----------+
| lastname  |
+-----------+
| Bondur    |
| Bott      |
| Bow       |
| Castillo  |
| Firrelli  |
| Fixter    |
| Gerard    |
| Hernandez |
| Jennings  |
| Jones     |
| Kato      |
| King      |
| Marsh     |
| Murphy    |
| Nishi     |
| Patterson |
| Thompson  |
| Tseng     |
| Vanauf    |
+-----------+
19 rows in set (0.00 sec)
```

As you can see in the new result set, four duplicate records are eliminated from the list when we used **DISTINCT**.

The **DISTINCT** keyword can be applied with more than one column. In this case, the combination of all columns are used to define the uniqueness of a record in the return result set. For example, to get all cities and states of customers in the *customers* table, we can use the following query:

```
SELECT DISTINCT city, state
FROM customers
```

Chapter 11

How to Use MySQL Limit to Constrain Number of Returned Records

Summary: *In this tutorial, you will learn how to use **MySQL LIMIT** clause to constrain number of returned records in SQL SELECT statement.*

Most of the times, when you work with master data tables which contain thousand to millions of records and you don't want to write a query to get all the data from those tables because of application's performance and high traffic between database server and application server. MySQL supports a cool feature called LIMIT to allow you to constrain the returned records with SELECT statement. Let take a look at the MySQL LIMIT syntax:

Let's say you have a database table with 10000 records and you want to get just first N records, you can use the following query:

```
SELECT * FROM table
LIMIT N
```

The MySQL LIMIT also allows you to get a range of records where you decide starting record number and how many records you want to retrieve. Here is the syntax of MySQL LIMIT to select a range of records:

```
SELECT columns
FROM table
LIMIT S, N
```

In the query above, S is the starting record index. MySQL specifies that the first record starts with 0. N is the number of records you want to select.

Let's practice MySQL LIMIT with several examples to have a better understanding.
If you want to get the first five employees in the table *employees*, you can use the following query:

```
SELECT firstname,lastname
FROM employees
LIMIT 5
```

```
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Diane     | Murphy    |
| Mary      | Patterson |
| Jeff      | Firrelli  |
| William   | Patterson |
| Gerard    | Bondur    |
+-----------+-----------+
5 rows in set (0.00 sec)
```

Now if you want to get five employees from employee number 10 you can use MySQL LIMIT with offset as follows:

```
SELECT firstname,lastname
FROM employees
LIMIT 10,5
```

```
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Foon Yue  | Tseng     |
| George    | Vanauf    |
| Loui      | Bondur    |
| Gerard    | Hernandez |
| Pamela    | Castillo  |
+-----------+-----------+
5 rows in set (0.00 sec)
```

Chapter 12

Selecting Data with SQL IN

Summary: *In this tutorial, you will learn how to select a result set which their values match any one of a list of values by using SQL IN operator.*

SQL IN Operator Syntax

The SQL IN operator allows you to select values that match any one of a list of values. The usage of the SQL IN operator is listed as follows:

```
SELECT column_list
FROM table_name
WHERE column IN ("list_item1","list_item2"…)
```

The column in WHERE clause does not need to be in *column_list* you selected, but it has to be a column in the table *table_name.* If the list has more than one value, each item has to be separated by a comma. In addition, you can use NOT operator with SQL IN to get values which does not match any value in a list of value.

Let's practice with several examples of SQL IN.
Suppose if you want to find out all offices which are located in US and France, you can perform the following query:

```
SELECT officeCode, city, phone
FROM offices
WHERE country = 'USA'  OR country = 'France'
```

In this case, we can use SQL IN instead of the above query:

```
SELECT officeCode, city, phone
FROM offices
WHERE country IN ('USA','France')
```

Here is the output

```
+------------+--------+-----------------+
| officeCode | city   | phone           |
+------------+--------+-----------------+
| 2          | Boston | +1 215 837 0825 |
| 3          | NYC    | +1 212 555 3000 |
| 4          | Paris  | +33 14 723 5555 |
| 8          | Boston | +1 215 837 0825 |
+------------+--------+-----------------+
```

To get all countries which does are not located in USA and France, we can use NOT IN in the where clause as follows:

```
SELECT officeCode, city, phone
FROM offices
WHERE country NOT IN ('USA','France')
```

Here is the output of offices which does not in USA and France

```
+------------+--------+------------------+
| officeCode | city   | phone            |
+------------+--------+------------------+
| 5          | Tokyo  | +81 33 224 5000  |
| 6          | Sydney | +61 2 9264 2451  |
| 7          | London | +44 20 7877 2041 |
+------------+--------+------------------+
```

SQL IN is used most often in sub-query. For example, if you want to find out all orders in the*orders* table which have total cost greater than $60000, we can use SQL IN with sub-query.
First to select all the orders which has total cost greater than $60000, you can retrieve it from*orderDetails* table as follows:

```
SELECT orderNumber
FROM orderDetails
GROUP BY orderNumber
HAVING SUM(quantityOrdered * priceEach) > 60000
```

You get all the orders which have total cost greater than $60000

```
+-------------+----------------+---------+---------------------+
| orderNumber | customerNumber | status  | shippedDate         |
+-------------+----------------+---------+---------------------+
|       10165 |            148 | Shipped | 2003-12-26 00:00:00 |
|       10287 |            298 | Shipped | 2004-09-01 00:00:00 |
|       10310 |            259 | Shipped | 2004-10-18 00:00:00 |
+-------------+----------------+---------+---------------------+
```

Chapter 13

Retrieving Data in a Range Using SQL BETWEEN

Summary: *In this tutorial, you will learn how to retrieve data from database tables which its value are in a range by using **SQL BETWEEN** operator.*

SQL BETWEEN Operator Syntax

The SQL BETWEEN operator allows you to retrieve values within a specific range. the SQL between must be used in the WHERE clause of the SQL SELECT statement. The following illustrates the SQL BETWEEN syntax:

```
SELECT column_list
FROM table_name
WHERE column_1 BETWEEN lower_range AND upper_range
```

MySQL returns all records in which the column_1 value is in the range of *lower_rage* and*upper_range* as well as the values lower_rage and upper_range. The query which is equivalent to SQL BETWEEN to get the same result is

```
SELECT column_list
FROM table_name
WHERE column_1 >= lower_range AND column_1 <= upper_range
```

Let's practice with several examples of using SQL BETWEEN to search values in a range.

Suppose we want to find all products which buy price is in a range of 90$ and 100$, we can perform the following query to do so:

```
SELECT productCode,ProductName,buyPrice
FROM products
WHERE buyPrice BETWEEN 90 AND 100
ORDER BY buyPrice DESC
```

Here is the output

```
+-------------+---------------------------------------+----------+
| productCode | ProductName                           | buyPrice |
+-------------+---------------------------------------+----------+
| S10_1949    | 1952 Alpine Renault 1300              |    98.58 |
| S24_3856    | 1956 Porsche 356A Coupe               |     98.3 |
| S12_1108    | 2001 Ferrari Enzo                     |    95.59 |
| S12_1099    | 1968 Ford Mustang                     |    95.34 |
| S18_1984    | 1995 Honda Civic                      |    93.89 |
| S18_4027    | 1970 Triumph Spitfire                 |    91.92 |
| S10_4698    | 2003 Harley-Davidson Eagle Drag Bike  |    91.02 |
+-------------+---------------------------------------+----------+
```

The output contains all products in the range of 90$ and 100$, and if there is a product with buy price 90$ or 100$, it will be included in the output too.

In order to find all records which are not in a range we use NOT BETWEEN. To find all products that buy price outside the range of 20 and 100, we can operate following query:

```
SELECT productCode,ProductName,buyPrice
FROM products
WHERE buyPrice NOT BETWEEN 20 AND 100
```

```
+-------------+-------------------------------------+----------+
| productCode | ProductName                         | buyPrice |
+-------------+-------------------------------------+----------+
| S10_4962    | 1962 LanciaA Delta 16V              |   103.42 |
| S18_2238    | 1998 Chrysler Plymouth Prowler      |   101.51 |
| S24_2972    | 1982 Lamborghini Diablo             |    16.24 |
| S24_2840    | 1958 Chevy Corvette Limited Edition |    15.91 |
+-------------+-------------------------------------+----------+
```

The query above is equivalent to the following query

```
SELECT productCode,ProductName,buyPrice
FROM products
WHERE buyPrice < 20 OR buyPrice > 100
ORDER BY buyPrice DESC
```

In this tutorial, you've learned how to use SQL BETWEEN to select data from database tables in a range.

Chapter 14

How to Use MySQL LIKE to Select Data Based on Patterns Matching

Summary: *MySQL provides LIKE operator in SQL standard. The **MySQL LIKE** operator is commonly used to select data based on patterns matching. Using **MySQL LIKE** in appropriate way is essential to increase application's performance. In this tutorial, you will learn how to use**MySQL LIKE** and when to avoid using it to increase the speed of retrieving data from database table.*

MySQL LIKE allows you to perform pattern matching in your characters column in a database table. MySQL LIKE is often used with SELECT statement in WHERE clause. MySQL provides you two wildcard characters for using with LIKE, the percentage % and underscore _.

- Percentage (%) wildcard allows you to match any string of zero or more characters
- Underscore (_) allows you to match any single character.

Let's practice with couples of examples which use MySQL Like with different wildcard characters.

Suppose you want to search for employee in *employees* table who has first name starting with character 'a', you can do it as follows:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE firstName LIKE 'a%'
```

```
+----------------+----------+-----------+
| employeeNumber | lastName | firstName |
+----------------+----------+-----------+
|           1611 | Fixter   | Andy      |
+----------------+----------+-----------+
1 row in set (0.00 sec)
```

MySQL scans the whole *employees* table to find all employees which have first name starting with character 'a' and followed by any number of characters.

To search all employees which have last name ended with 'on' string you can perform the query as follows:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastName LIKE '%on'
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|           1088 | Patterson | William   |
|           1216 | Patterson | Steve     |
+----------------+-----------+-----------+
2 rows in set (0.00 sec)
```

If you know a searched string is embedded somewhere in a column, you can put the percentage wild card at the beginning and the end of it to find all possibilities. For example, if you want to find all employees which have last name containing 'on' string you can execute following query:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastname LIKE '%on%'
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|           1088 | Patterson | William   |
|           1102 | Bondur    | Gerard    |
|           1216 | Patterson | Steve     |
|           1337 | Bondur    | Loui      |
|           1504 | Jones     | Barry     |
+----------------+-----------+-----------+
5 rows in set (0.00 sec)
```

To search all employees whose name are such as Tom, Tim… You can use underscore wildcard

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE firstname LIKE ''T_m"
```

```
+----------------+----------+-----------+
| employeeNumber | lastName | firstName |
+----------------+----------+-----------+
|           1619 | King     | Tom       |
+----------------+----------+-----------+
1 row in set (0.00 sec)
```

The MySQL LIKE allows you to put the NOT keyword to find all strings which are unmatched with a specific pattern. Suppose you want to search for all employees whose last name are not starting with 'B', you can use the following query

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastName NOT LIKE 'B%'
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|           1088 | Patterson | William   |
|           1188 | Firrelli  | Julie     |
|           1216 | Patterson | Steve     |
|           1286 | Tseng     | Foon Yue  |
|           1323 | Vanauf    | George    |
|           1370 | Hernandez | Gerard    |
|           1401 | Castillo  | Pamela    |
|           1504 | Jones     | Barry     |
|           1611 | Fixter    | Andy      |
|           1612 | Marsh     | Peter     |
|           1619 | King      | Tom       |
|           1621 | Nishi     | Mami      |
```

```
|           1625 | Kato      | Yoshimi   |
|           1702 | Gerard    | Martin    |
+----------------+-----------+-----------+
14 rows in set (0.00 sec)
```

Be noted that SQL LIKE is not case sensitive so 'b%' and 'B%' are the same.

What if you want to search for records which have a field starting with a wildcard character? In this case, you can use ESCAPE to shows that the wildcard characters followed it has literal meaning not wildcard meaning. If ESCAPE does not specify explicitly, the escape character in MySQL by default is '\'. For example, if you want to find all products which as product code which has _20 embedded on it, you can perform following query.

```
SELECT productCode, productName
FROM products
WHERE productCode LIKE '%\_20%'
```

```
+-------------+-----------------------------------------+
| productCode | productName                             |
+-------------+-----------------------------------------+
| S10_2016    | 1996 Moto Guzzi 1100i                   |
| S24_2000    | 1960 BSA Gold Star DBD34                |
| S24_2011    | 18th century schooner                   |
| S24_2022    | 1938 Cadillac V-16 Presidential Limousine |
| S700_2047   | HMS Bounty                              |
+-------------+-----------------------------------------+
5 rows in set (0.00 sec)
```

MySQL LIKE gives you a convenient way to find records which have character columns match specified patterns. Because MySQL LIKE scans the whole table to find all the matching records therefore it does not allow database engine to use the index for fast searching. When the data in the table is big enough, the performance of MySQL LIKE will degrade. In some cases you can avoid this problem by using other techniques to achieve the same result as MySQL LIKE. For example, if you want to find all employees which have first name starting with a specified string you can use LEFT function in where clause like the following query:

```
SET @str = 'b';
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE LEFT(lastname,length(@str)) = @str;
```

It returns the same result as the query below but it faster because we can leverage the index on the column *lastname*.

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastname LIKE 'b%'
```

And another technique to achieve all string which end with a specified string by using RIGHT function. Suppose we want to retrieve all employees which have last name ended with 'on' string, we can use RIGHT function instead of MySQL LIKE as follows:

```
SET @str = 'on';
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE RIGHT (lastname,length(@str)) = @str;
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|           1088 | Patterson | William   |
|           1216 | Patterson | Steve     |
+----------------+-----------+-----------+
2 rows in set (0.00 sec)
```

It returns the same result as the following query

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastname LIKE '%on'
```

Chapter 15

Combining Result Sets with MySQL UNION

Summary: *In this tutorial, you will learn how to use **MySQL UNION** statement to combine two or more result sets from multiple SQL SELECT statements into a single result set.*

Like SQL standard, MySQL UNION allows you to combine two or more result sets from multiple tables together. The syntax of using MySQL UNION is as follows:

```
SELECT statement
UNION [DISTINCT | ALL]
SELECT statement
UNION [DISTINCT | ALL]
…
```

In order to use UNION statement, there are some rules you need to follow:

- The number of columns in each SELECT statement has to be the same .
- The data type of the column in the column list of the SELECT statement must be the same or at least convertible.

By default the MySQL UNION removes all duplicate rows from the result set even if you don't explicit using DISTINCT after the keyword UNION.
If you use UNION ALL explicitly, the duplicate rows remain in the result set. You only use this in the cases that you want to keep duplicate rows or you are sure that there is no duplicate rows in the result set.

Let's practice with couples of examples with MySQL UNION to get a better understanding.

Suppose you want to combine ***customers*** and ***employees*** infomation into one result set, you use the following query:

```
SELECT customerNumber id, contactLastname name
FROM customers
UNION
SELECT employeeNumber id,firstname name
FROM employees
```

Here is the excerpt of the output:

```
id  name
------  ---------------
   103  Schmitt
   112  King
   114  Ferguson
   119  Labrune
   121  Bergulfsen
   124  Nelson
   125  Piestrzeniewicz
```

```
128  Keitel
129  Murphy
131  Lee
```

When using ORDER BY to sort the result with UNION, you have to use it in the last SQL SELECT statement. It would be the best to parenthesize all the SELECT statements and place ORDER BY at the end.

Suppose you want to sort the combination of ***employees*** and ***customers*** in the query above by***name*** and ***ID*** in ascending order.

```
(SELECT customerNumber id,contactLastname name
FROM customers)
UNION
(SELECT employeeNumber id,firstname name
FROM employees)
ORDER BY name,id
```

What will be displayed in the output if we don't use alias for each column in the SELECT statements? MySQL will use the column names of the first SELECT statement as the label of the output.

```
(SELECT customerNumber, contactLastname
FROM customers)
UNION
(SELECT employeeNumber, firstname
FROM employees)
```

MySQL also provides you another option to sort the result set based on column position in the ORDER BY clause as the following query:`contactLastname, customerNumber`

```
(SELECT customerNumber, contactLastname
FROM customers)
UNION
(SELECT employeeNumber,firstname
FROM employees)
ORDER BY 2, 1
```

Chapter 16

MySQL INNER JOIN

***Summary**: In this tutorial, you will learn how to use **MySQL INNER JOIN** clause to select data from multiple tables based on join conditions.*

Introducing MySQL INNER JOIN clause

The *MySQL INNER JOIN* clause is an optional part of SQL SELECT statement. The MySQL INNER JOIN clause appears immediately after the FROM clause. Before using MySQL INNER JOIN clause, you have to specify the following criteria:

- First, you need to specify the tables you want to join with the main table. The main table appear in the FROM clause. The table you want to join with appear after keyword INNER JOIN. Theoretically, you can join a table with unlimited number of tables. However, for better performance you should limit the number of tables to join based on join conditions and volume of data in those tables.
- Second, you need to specify the join condition or join predicate. The join condition appears after the keyword ON of MySQL INNER JOIN clause. The join condition is the rule for matching rows between the main table and other tables being joined with.

The syntax of the MySQL INNER JOIN is as follows:

```
SELECT column_list
FROM t1
INNER JOIN t2 ON join_condition1
INNER JOIN t3 ON join_condition2
...
WHERE where_conditions;
```

For example, if you join two tables A and B, the MySQL INNER JOIN clause compares each record of the table A with each record of table B to find all pair of records that satisfy the join-condition. When the join-condition are satisfied, column values for each matched pair of record of table A and table B are combined into a returned record. Note that the records on both tables have to match based on the join-condition. If no record on both table A and B matches, the query will return an empty result.

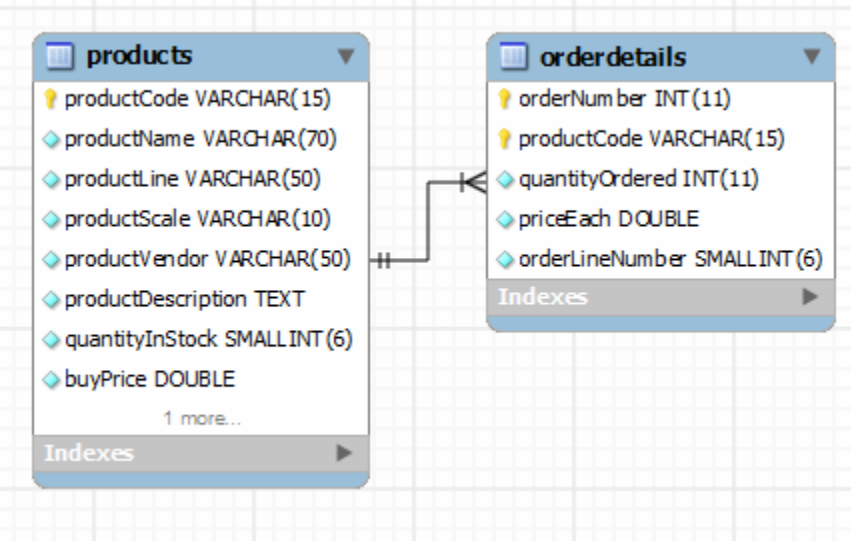
Avoid column ambiguous error in MySQL INNER JOIN

If you join multiple tables that has column with similar name, you have to use table qualifier to refer to column to avoid column ambiguous error. Suppose if table *tbl_A* and *tbl_B* has the same column *M*. In the SELECT statement with MySQL INNER JOIN clause, you have to refer to column *M* by using the table qualifier as *tbl_A.M* or *tbl_B.M* (*table_name.column_name*).

Another effective way to avoid column ambiguous is by using table alias. For example, you can give A as the table alias of the table *tbl_A* and refer to the column *M* as *A.M* so you don't have to type again and again the long table name in your SQL statement.

Example of MySQL INNER JOIN clause

Let's take a look at two tables: products and orderDetails in our sample database.



The products table is the master data table that stores all products. Whenever a product is sold, it is stored in the orderDetails table with other information. The link between products table and orderDetails table is productCode.

Now, if you want to know what product was sold in which order, you can use the *MySQL INNER JOIN* clause as follows:

```
SELECT A.productCode, A.productName, B.orderNumber
FROM products A
INNER JOIN orderDetails B on A.productCode = B.productCode;
```

productCode	productName	orderNumber
S18_1749	1917 Grand Touring Sedan	10100
S18_2248	1911 Ford Town Car	10100
S18_4409	1932 Alfa Romeo 8C2300 Spider Sport	10100
S24_3969	1936 Mercedes Benz 500k Roadster	10100
S18_2325	1932 Model A Ford J-Coupe	10101
S18_2795	1928 Mercedes-Benz SSK	10101
S24_1937	1939 Chevrolet Deluxe Coupe	10101
S24_2022	1938 Cadillac V-16 Presidential Limousine	10101
S18_1342	1937 Lincoln Berline	10102
S18_1367	1936 Mercedes-Benz 500K Special Roadster	10102
S10_1949	1952 Alpine Renault 1300	10103
S10_4962	1962 LanciaA Delta 16V	10103
S12_1666	1958 Setra Bus	10103
S18_1097	1940 Ford Pickup Truck	10103

There are more returned rows that are not listed on this screenshot

The MySQL INNER JOIN clause compares each row of table products and orderDetails table to find a pair of rows that has the same productCode. If a pair of rows that have the same, the product code, product name and order number are combined into a returned row.

In this tutorial, you have learned how to use MySQL INNER JOIN to select data from multiple tables. You have also learned how to use table qualifier to avoid column ambiguous error in MySQL INNER JOIN clause.

Chapter 17

MySQL LEFT JOIN

Summary: *In this tutorial, you will learn how to use **MySQL LEFT JOIN** clause to retrieve data from more than one table.*

Introducing to MySQL LEFT JOIN

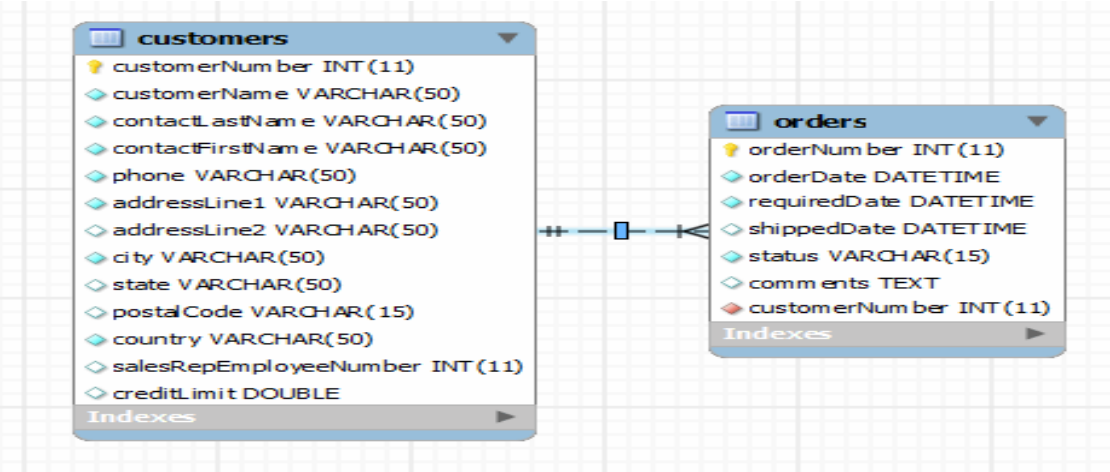
The MySQL LEFT JOIN clause is an optional element of the SQL SELECT statement to allow you to retrieve data from additional tables. The MySQL LEFT JOIN clause consists of LEFT JOIN keyword followed by the second table you want to join. Next element is the ON keyword followed by the join condition. In the join condition, you specify the columns to be used for matching row in the two tables. The syntax of MySQL is as follows:

```
SELECT t1.c1, t1.c2,…t2.c1,t2.c2
FROM t1
LEFT JOIN t2 ON t1.c1 = t2.c1 …(join_condition)
WHERE where_condition
```

The MySQL LEFT JOIN clause works like this: when a row from the left table matches a row from the right table based on join_condition, the row's content are selected as an output row. When row in the left table has no match it is still selected for output, but combined with a "fake" row from the right table that contains NULL in all columns. In short, the MySQL LEFT JOIN clause allows you to select all rows from the left table even there is no match for them in the right table.

Example of MySQL LEFT JOIN

Let's take a look at two table customers and orders: If you want to know which customer has which order and each order's status. You can use the MySQL LEFT JOIN as follows:




```
SELECT c.customerNumber, customerName,orderNUmber, o.status
FROM customers c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber;
```

customerNumber	customerName	orderNUmber	status
124	Mini Gifts Distributors L...	10371	Shipped
124	Mini Gifts Distributors L...	10382	Shipped
124	Mini Gifts Distributors L...	10385	Shipped
124	Mini Gifts Distributors L...	10390	Shipped
124	Mini Gifts Distributors L...	10396	Shipped
124	Mini Gifts Distributors L...	10421	In Process
125	Havel & Zbyszek Co	NULL	NULL
128	Blauer See Auto, Co.	10101	Shipped
128	Blauer See Auto, Co.	10230	Shipped
128	Blauer See Auto, Co.	10300	Shipped
128	Blauer See Auto, Co.	10323	Shipped

There are more rows which are not listed on this screenshot.

The left table is the customers table so you see all customers are listed as the way MySQL LEFT JOIN clause works. However, there are rows that we have customer information but all order information are NULL. This means those customers do not have any order in our database. The MySQL LEFT JOIN clause is very useful when you want to find the records in the left table that are unmatched by the right table. You can accomplish this by add a WHERE clause to select only that have NULL values in a right table column. So to find all customers who does not have any order in our database we can use the MySQL LEFT JOIN clause as follows:

```
SELECT c.customerNumber, customerName,orderNUmber, o.status
FROM customers c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber
WHERE orderNumber is NULL
```

customerNumber	customerName	orderNUmber	status
125	Havel & Zbyszek Co	NULL	NULL
168	American Souvenirs Inc	NULL	NULL
169	Porto Imports Co.	NULL	NULL
206	Asian Shopping Network, Co	NULL	NULL
223	Natürlich Autos	NULL	NULL
237	ANG Resellers	NULL	NULL
247	Messner Shopping Network	NULL	NULL
273	Franken Gifts, Co	NULL	NULL
293	BG&E Collectables	NULL	NULL
303	Schuyler Imports	NULL	NULL

There are more rows which are not listed on this screenshot.

As the result, the query only returns customers which do not have any order represented by NULL values.

In this tutorial, you have learned how to use ***MySQL LEFT JOIN*** clause to select data from multiple tables. You've also learned how to use ***MySQL LEFT JOIN*** to find unmatched records between two tables.

Chapter 18

MySQL GROUP BY

Summary: *In this tutorial, you will learn how to use **MySQL GROUP BY** to group selected records into a set of summary records by the one or more column's value or expression.*

Introducing to MySQL GROUP BY clause

The ***MySQL GROUP BY*** clause is used with SQL SELECT statement to to group selected records into a set of summary records by the one or more column's value or expression. The MySQL GROUP BY clause is an optional part of the SQL SELECT statement. The MySQL GROUP BY clause must appear after the WHERE clause or FROM clause if WHERE clause is omitted of the SQL SELECT statement. The MySQL GROUP BY clause consists of GROUP BY keyword followed by a list of expressions separated by commas. The following illustrates the MySQL GROUP BY clause:

```
SELECT col1_,col_2,... col_n, aggregate_function(expression)
FROM table
WHERE where_conditions
GROUP BY col_1, col_2, ... col_n
ORDER BY column_list
```

We will examine the GROUP BY clause in details. Let's take a look at the order table in our sample database.

Example of MySQL GROUP BY clause

Now, suppose you want to get groups for each order, you can use the MySQL GROUP BY clause as follows:

```
SELECT status
FROM orders
GROUP BY status
```

```
+------------+
| status     |
+------------+
| Cancelled  |
| Disputed   |
| In Process |
| On Hold    |
| Resolved   |
| Shipped    |
+------------+
6 rows in set (0.00 sec)
```

It seems that the GROUP BY clause only scans for unique occurrences in the status column and return the result set. However if you look at the data of the orders table you will see that each row in result set above is summary of records that represent a group orders that have the same status on the status column.

MySQL GROUP BY with aggregate function

The aggregate functions allow you to perform calculation of a set of records and return a single value. The most common aggregate functions are SUM, AVG, MAX, MIN and COUNT. For more information on aggregate functions, please refer to the ***aggregate functions in MySQL*** tutorial.

Aggregate functions are used with MySQL GROUP BY clause to perform calculation on each group of records on return a single value for each row. Let's say if you want to know how many orders in each status group you can use the COUNT function as follows:

```
SELECT status, count(*)
FROM orders
GROUP BY status
```

```
+------------+----------+
| status     | count(*) |
+------------+----------+
| Cancelled  |        6 |
| Disputed   |        3 |
| In Process |        6 |
| On Hold    |        4 |
| Resolved   |        4 |
| Shipped    |      303 |
+------------+----------+
6 rows in set (0.00 sec)
```

The query counts number of orders for each order's status.

MySQL GROUP BY vs. ANSI SQL GROUP BY

MySQL follows ANSI SQL. However, there are two differences the way GROUP BY works in MySQL and ANSI SQL.

- In ANSI SQL you must group by all columns you specifies in the SELECT clause. MySQL does not have this restriction. You can have additional columns in the SELECT clause that are not in the GROUP BY clause.
- MySQL also allows you to sort the group order in which the results are returned. The default order is ascending.

If you want to see the result of the query above in the descending order, you can do it as follows:

```
SELECT status, count(*)
FROM orders
GROUP BY status DESC;
```

```
+------------+----------+
| status     | count(*) |
+------------+----------+
| Shipped    |      303 |
| Resolved   |        4 |
| On Hold    |        4 |
| In Process |        6 |
| Disputed   |        3 |
| Cancelled  |        6 |
+------------+----------+
6 rows in set (0.00 sec)
```

As you see the status of orders now are in reverse alphabetical order. By default it is ascending order determined by ASC.

In this tutorial, you have learned how to use MySQL GROUP BY to retrieve data records in-group. You've also learned how to sort the result set in the MySQL GROUP BY clause supported only in MySQL.

Chapter 19

MySQL HAVING

***Summary**: In this tutorial, you will learn how to use **MySQL HAVING** clause to specify a filter condition for a group of records or an aggregate.*

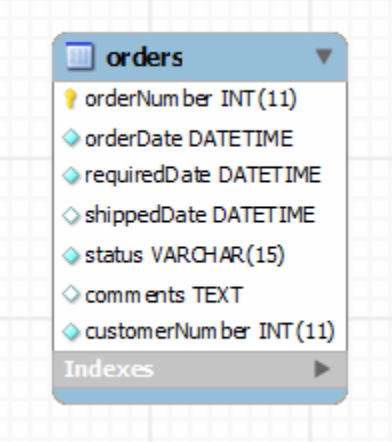
Introducing MySQL HAVING clause

The MySQL HAVING clause is an optional part of and used only with the SQL SELECT statement. The MySQL HAVING clause specifies a filter condition for a group of record or an aggregate. The MySQL HAVING is often used with MySQL GROUP BY clause. When using with MYSQL GROUP BY clause, you can apply filter condition of the HAVING clause only to the columns appear in the GROUP BY clause. If the MySQL GROUP BY clause is omitted, the MySQL HAVING clause will behave like a WHERE clause. Notes that the MySQL HAVING clause applies to groups as a whole while the WHERE clause applies to individual rows.

Examples of MySQL HAVING clause

Let's take a look at an example of using MySQL HAVING clause to have a better understanding.

We have *orderDetails* table in our sample database. We can use the MySQL GROUP BY clause to get all orders, number of items sold and total values in each order as follows:




```
SELECT ordernumber,
       sum(quantityOrdered) AS itemsCount,
       sum(priceeach) AS total
FROM orderdetails
GROUP BY ordernumber
```

ordernumber	itemsCount	total
10100	151	301.84
10101	142	352
10102	80	138.68
10103	541	1520.37
10104	443	1251.89
10105	545	1479.71
10106	675	1427.28
10107	229	793.21
10108	561	1432.86
10109	212	700.89

Now you can ask what order has total value greater than $1000. In this case, you need to use the MySQL HAVING clause on aggregate to answer that question.

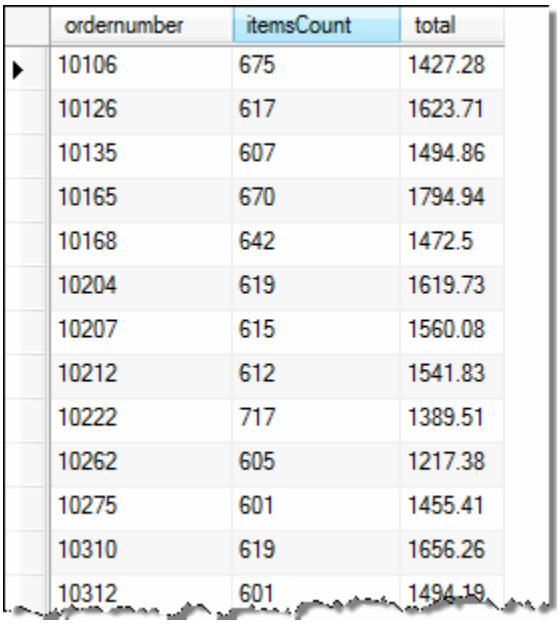
```
SELECT ordernumber,
       sum(quantityOrdered) AS itemsCount,
       sum(priceeach) AS total
FROM orderdetails
GROUP BY ordernumber
HAVING total > 1000
```

ordernumber	itemsCount	total
10103	541	1520.37
10104	443	1251.89
10105	545	1479.71
10106	675	1427.28
10108	561	1432.86
10110	570	1338.47
10117	402	1307.47
10119	442	1081.44
10120	525	1322.67
10122	545	1598.27
10126	617	1623.71
10127	540	1601.39
10135	607	1494.86

We use column alias for the aggregate *sum(priceeach)* as *total* so in the HAVING clause we just have to specify that column alias *total* instead of typing the aggregate *sum(priceeach)* again.

You can use a complex condition in the MySQL HAVING clause such as OR, AND operators. For example if you want to know what order has total value greater than $1000 and has more than 600 items in it. You can use the following query to find out:

```
SELECT ordernumber,
       sum(quantityOrdered) AS itemsCount,
       sum(priceeach) AS total
FROM orderdetails
GROUP BY ordernumber
HAVING total > 1000 AND itemsCount > 600
```



ordernumber	itemsCount	total
10106	675	1427.28
10126	617	1623.71
10135	607	1494.86
10165	670	1794.94
10168	642	1472.5
10204	619	1619.73
10207	615	1560.08
10212	612	1541.83
10222	717	1389.51
10262	605	1217.38
10275	601	1455.41
10310	619	1656.26
10312	601	1494.19

The MySQL HAVING clause is useful only with the MySQL GROUP BY clause for building output of high-level reports. For example, you can use the MySQL HAVING clause to answer questions like how many order has total values more than 1000 this month, this quarter and this year?...

In this tutorial, you have learned how to use the ***MySQL HAVING*** clause together with the MySQL GROUP BY to specify filter condition on a group or aggregate.

Chapter 20

Inserting Data into Database Tables

Summary: *In the previous tutorials you learn different ways to query data from database table by using SQL SELECT statement. Are you wonder that how data is added into those table? In this tutorial, you will learn do it by using SQL INSERT statement.*

INSERT Statement

INSERT statement allows you to insert one or more rows to the table. In MySQL, the INSERT statement forms are listed as follows:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] table_name [(column_name,...)]
       VALUES ((expression | DEFAULT),...),(...),...

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] table_name [(column_name,...)]

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] table_name
       SET column_name=(expression | DEFAULT), ...
```

As you can see INTO in the INSERT statement is optional. In the first form, you insert a new data row into an existing table by specifying the column name and data for each. As an example to insert a new office to the *offices* table in the sample database you can do as follows:

```
INSERT INTO classicmodels.offices
       (officeCode,
       city,
       phone,
       addressLine1,
       addressLine2,
       state,
       country,
       postalCode,
       territory
       )
       VALUES
       ('8',
       'Boston',
       '+1 215 837 0825',
       '1550 dummy street',
       'dummy address',
       'MA',
       'USA',
       '02107',
       'NA'
       )
```

In the second form, instead of providing explicit data, you select it from other table by using SELECT statement. This form allows you to copy some or some part of data from other table to the inserted table. As an example, we can create a temporary table and insert all offices which locate in US into that one by using this query:

```
INSERT INTO temp_table
SELECT * FROM offices WHERE country = 'US'
```

The third form enables you to specify the column you want to insert the data. For example, we have the query like this:

```
INSERT INTO productlines
SET productLine = 'Luxury Cars'
```

It means we only insert the data into *productLine* column in *productLines* table.

Chapter 21

Updating Data in Database Tables

***Summary**: Updating existing data is one of the most important task when you work with database. In this tutorial, you will learn how to use **MySQL UPDATE** statement to update data in database tables.*

SQL UPDATE statement is used to update existing data in database tables. It can be used to change values of single row, group of rows or even all rows in a table. In MySQL, the SQL UPDATE statement form is as below:

```
UPDATE [LOW_ PRIORITY] [IGNORE] table_name [, table_name...]
       SET column_name1=expr1
              [, column_name2=expr2 ...]
       [WHERE condition]
```

Let's examine the MySQL UPDATE statement in details:

- Followed by the UPDATE keyword is the name of a table you want to change the data. In MySQL, you can change the data of multiple tables at a time. If an UPDATE command violates any integrity constraint, MySQL does not perform the update and it will issue an error message.
- The SET clause determines the columns' name and the changed values. The changed values could be a constant value, expression or even a subquery.
- WHERE clause determines which rows of the tables will be updated. It is an optional part of SQL UPDATE statement. If WHERE clause is ignored, all rows in the tables will be updated. The WHERE clause is so important that you should not forget. Sometimes you want to change just one row of a table but you forget WHERE clause so you accidentally update the whole table.
- LOW_ PRIORITY keyword is used to delay the execution until no other client applications reading data from the table. This is used for controlling the update process in MySQL database server.
- IGNORE keyword is used to execute the update even errors occurred during the execution. The errors in the update process could be duplicate value on unique column, or new data does not match with the column data type. In the first situation data is not updated and in the second one MySQL tries to convert the data into closest valid values.

Let's practice with a couple of examples in our sample database to understand more about SQL UPDATE statement.

In ***employees*** table, if you want to update the email of Mary Patterson with employeeNumber 1 with the new email as mary-patterso@classicmodelcars.com, you can execute the following query:
Make sure we are truly updating the data we select it first to see what the current data looks like.

```
SELECT firstname,
       lastname,
       email
FROM employees
WHERE employeeNumber = 1
```

```
+-----------+-----------+--------------------------------+
| lastname  | firstname | email                          |
+-----------+-----------+--------------------------------+
| Patterson | Mary      | mpatterso@classicmodelcars.com |
+-----------+-----------+--------------------------------+
1 row in set (0.02 sec)
```

Update her email to the new email mary-patterso@classicmodelcars.com

```
UPDATE employees
SET email = 'mary-patterso@classicmodelcars.com'
WHERE employeeNumber = 1
```

Execute the select query above again; you will see the email change to the new value.

```
+-----------+-----------+---------------------------------+
| lastname  | firstname | email                           |
+-----------+-----------+---------------------------------+
| Patterson | Mary      | mary-patterso@classicmodelcars.com |
+-----------+-----------+---------------------------------+
1 row in set (0.00 sec)
```

In this tutorial, you've learned how to use SQL UPDATE statement in MySQL to update data of database tables.

Chapter 22

Deleting Records from Tables Using MySQL DELETE

Summary: *The **MySQL DELETE** statement not only allows you to delete record from one table but also multiple tables. In this tutorial, you will learn how to use **MySQL DELETE** statement with examples.*

To remove all rows or records from a database table you use ***MySQL DELETE*** statement. The following illustrates **MySQL DELETE** statement:

```
DELETE [LOW_PRIORITY] [QUICK]
       FROM table_name
[WHERE conditions] [ORDER BY ...] [LIMIT rows]

DELETE [LOW_PRIORITY] [QUICK]
       table_name[.*] [, table_name[.*] ...]
FROM table-references
[WHERE where_definition]

DELETE [LOW_PRIORITY] [QUICK]
FROM table_name[.*] [, table_name[.*] ...]
USING table-references
[WHERE where_definition]
```

Let's examine the MySQL DELETE statements in details as below:

- In the first form of MySQL DELETE statement, followed the DELETE FROM part is the table name where you want to delete records. The WHERE clause specifies condition to limit which rows you want to to remove. If a record meets WHERE condition, it will be removed from the database table permanently. If the WHERE clause is ignored in the MySQL DELETE statement, all rows of the table are deleted.
- The second form of MySQL DELETE statement, It deletes records from multiple tables which reference to other table.
- The third form of MySQL DELETE statement is quite similar to the second one except Using keyword is used instead of FROM keyword.

Let's have a couple of examples of using MySQL DELETE statement in the sample database.
It is recommended that you make a copy of employee table before practicing with the MySQL DELETE statement.
Suppose you want to delete all employees in an office with *officeNumber* is 4, just execute the following query:

```
DELETE FROM employees
WHERE officeCode = 4
```

To delete all employees from all offices, just remove the WHERE condition as follows:

```
DELETE FROM employees
```

It will remove all rows from *employees* table.

If you want to delete all employee who work for office with *officecode* 1 and also that office. You can use the second form of MySQL DELETE statement to delete data from multiple tables as follows:

```
DELETE employees,offices
FROM employees,offices
WHERE employees.officeCode = offices.officeCode  AND
      offices.officeCode = 1
```

You can achieve the same above effect by using the third form of DELETE statement as below:

```
DELETE FROM employees,offices
USING employees,offices
WHERE employees.officeCode = offices.officeCode  AND
      offices.officeCode = 1
```

In this tutorial, you've learned various forms of MySQL DELETE statement to delete records from one or multiple database tables.

Chapter 23

How to Use MySQL REPLACE to Insert or Update Data

Summary: *In this tutorial, you will learn how to use **MySQL REPLACE** statement to insert or update data in database tables.*

MySQL REPLACE statement is a MySQL extension to SQL standard. MySQL REPLACE works like the INSERT statement with the following rules:

- If the record being inserting does not exist, MySQL REPLACE will insert a new record.
- If the record being inserting exists, MySQL REPLACE will delete the old record first and then insert a new record with new values.

In order to use **MySQL REPLACE** you need to have at least both INSERT and DELETE privileges for the table.

The first form of **MySQL REPLACE** statement is similar to the INSERT statement except the keyword INSERT is replaced by the REPLACE keyword as follows:

```
REPLACE INTO table_name(column_name1,column_name2,…)
VALUES(value1,value2,…)
```

For example if you want to insert a new office into *offices* table, you use the following query:

```
REPLACE INTO offices(officecode,city)
VALUES(8,'San Jose')
```

Note that all values of missing columns in the REPLACE statement will be set to default value.

Now if you want to update the inserted office with San Mateo city, we can do it as follows:

```
REPLACE INTO offices(officecode,city)
VALUES(8,'San Mateo')
```

You see that two rows affected by the query above because the existing record is deleted and the new record is inserted.

The second form of MySQL REPLACE like UPDATE statement as follows:

```
REPLACE INTO table_name
SET column_name1 = value1 AND
    column2 = value2
```

Note that there is no WHERE clause is specified in the REPLACE statement. For example, if you want to update office in San Mateo with *officecode* 8 you can do it as follows:

```
REPLACE INTO offices
SET officecode = 8 AND
    city = ‘Santa Cruz’
```

The third form of MySQL REPLACE is similar to SQL INSERT INTO SELECT statement as follows:

```
REPLACE INTO table_name1(column_name1,column_name2,…)
SELECT column_name1, column_name2…
FROM table_name2
WHERE where_condition
```

Let’s say if we want to copy the office with officecode 1, we can do it as follows:

```
REPLACE INTO offices(officecode,
                city,
                phone,
                addressline1,
                addressline2,
                state,
                country,
                postalcode,
                territory)
SELECT (SELECT MAX(officecode) + 1 FROM offices),
        city,
        phone,
        addressline1,
        addressline2,
        state,
        country,
        postalcode,
        territory
FROM offices
WHERE officecode = 1
```

There are several important points you need to know before using MySQL REPLACE statement:

- If you are developing an application that potentially supports not only MySQL database, try to avoid using MySQL REPLACE because other database management system may not support the REPLACE statement. You can use the combination of INSERT and DELETE statement instead.
- If you are using MySQL REPLACE in the table which has *triggers* associate with it and if the deletion of duplicate key happens, the *triggers* are fired in the following orders:
    - Before insert
    - Before delete
    - After delete
    - After insert
- It is preferred using MySQL UPDATE statement to using MySQL REPLACE in case you just want to update data. Because MySQL UPDATE performs faster than MySQL REPLACE.

In this tutorial, you've learned different forms of **MySQL REPLACE** statement to insert or update data in database tables.

Chapter 24

MySQL Transaction

Summary: *In this tutorial, you'll learn what **MySQL transaction** is and how to use MySQL COMMIT and MySQL ROLLBACK statements to manage transactions in MySQL.*

Introducing to MySQL Transaction

To understand what the transaction in MySQL is, let take a look at an example of adding a new sale order in our sample database. The steps of adding a sale order are as below:

- Get latest sale order number from *orders* table, and use the next sale order number as the new sale order number.
- Insert a new sale order into *orders* table for a given customer
- Insert new sale order items into *orderdetails* table
- Get data from both table *orders* and *orderdetails* tables to confirm the changes

Now imagine what would happen to your data if one or more steps above fail because of database failure such as table lock security …etc? If the step of adding order items into *orderdetails* table failed, you would have an empty sale order in your system without knowing it. Your data may not be integrity and the effort you have to spend to fix it is tremendous. How do you solve this problem? That's why the transaction processing comes to the rescue. MySQL transaction enables you to execute sets of MySQL operations to ensure that the databases never contain the result of partial operations. In the set of operations, if one of them fails, the rollback occurs to restore the database. If no error occurred, the entire set of statements is committed to the database.

Using MySQL Transaction

Let's review the most important MySQL transaction statements before we are using them for the adding sale order example above. To start a transaction you use the START TRANSACTION statement To undo MySQL statements you use ROLLBACK statement. Note that there are several SQL statements you cannot use ROLLBACK such as:

```
CREATE / ALTER / DROP DATABASE
CREATE / ALTER / DROP / RENAME / TRUNCATE TABLE
CREATE / DROP INDEX
CREATE / DROP EVENT
CREATE / DROP FUNCTION
CREATE / DROP PROCEDURE
…
```

To write the changes into the database within a transaction you use the COMMIT statement. It is important to note that MySQL automatically commit the changes to the database by default. To force MySQL not to commit changes automatically, you need to use the following statement:

```
SET autocommit = 0;
```

MySQL transaction example

In order to use MySQL transaction, you first have to break your SQL statements into logical portion and determine when data should be committed or rollback. Let's take a look an example of using MySQL transaction to add new sale order into our sample database above and add the transaction processing steps:

- Start a transaction using START TRANSACTION
- Get latest sale order number from *orders* table, and use the next sale order number as the new sale order number.
- Insert a new sale order into *orders* table for a given customer
- Insert new sale order items into *orderdetails* table
- Commit changes using COMMIT statement
- Get data from both table *orders* and *orderdetails* tables to confirm the changes

Here is the MySQL code to perform the above steps:

```
-- start a new transaction
start transaction;

-- get latest order number
select @orderNumber := max(orderNUmber)
from orders;
-- set new order number
set @orderNumber = @orderNumber  + 1;

-- insert a new order for customer 145
insert into orders( orderNumber,
                    orderDate,
                    requiredDate,
                    shippedDate,
                    status,
                    customerNumber)
values(@orderNumber,
```

```
       now(),
       date_add(now(), INTERVAL 5 DAY),
       date_add(now(), INTERVAL 2 DAY),
       'In Process',
       145);
-- insert 2 order line items
insert into orderdetails( orderNumber,
                          productCode,
                          quantityOrdered,
                          priceEach,
                          orderLineNumber)
values(@orderNumber,'S18_1749', 30, '136', 1),
       (@orderNumber,'S18_2248', 50, '55.09', 2);
-- commit changes
commit;

-- get the new inserted order
select * from orders a
inner join orderdetails b on a.ordernumber = b.ordernumber
where a.ordernumber = @ordernumber;
```

In this tutorial, you've learned how to use MySQL transaction statements including START TRANSACTION, COMMIT ad ROLLBACK to manage transactions in MySQL to ensure data integrity.

Chapter 25

MySQL Prepared Statement

In this tutorial, you will learn how to use ***MySQL prepared statement*** to utilize binary protocol implemented since MySQL 4.1 to make your code more secure and faster.

Introduction to MySQL Prepared Statement

Before MySQL 4.1, the query is sent to the MySQL server in textual format and the data is returned back to client was using textual protocol only. The query has to be parsed fully each time and the result also has to be converted to string on server side before returning back to client side. This textual protocol has serious performance implication. In MySQL 4.1 a new feature called prepared statement was added. The prepared statement takes advantage of client/server binary protocol. Basically prepared statement passed to the MySQL server contains placeholders such as

```
SELECT * FROM products WHERE productCode = ?
```

When executing query with different product code, MySQL server does not have to parse the query fully. It is not only faster when the query is executed multiple times but also more secure. Because prepared statement uses placeholder so it saves your application from many variants of SQL injection.

MySQL Prepared Statement Syntax

MySQL prepared statement is based on other three MySQL statements as follow:

- PREAPRE to prepare statement for execution
- EXECUTE to execute a prepare statement prepare by PREPAPRE statement
- DEALLOCATE PREPARE to release a prepared statement.

Let's take a look at an example of using MySQL prepared statement.

```
PREPARE stmt1 FROM 'SELECT productCode, productName
                    FROM products
                    WHERE productCode = ?';

SET @pc = 'S10_1678';
EXECUTE stmt1 USING @pc;

DEALLOCATE PREPARE stmt1;
```

In the above example, first we used **PREPARE** statement to prepare a statement for execution. We used SQL SELECT statement to retrieve a product from products table for a given product code. We used product code as a place holder. Next we declared product code variable *@pc* and set it values to *S10_1678*. Then we use **EXECUTE** statement to execute the prepared statement product code variable *@pc*. Finally we used **DEALLOCATE PREPARE** to release the prepared statement.

In this tutorial, you've learned how to use MySQL prepared statement to execute a query with placeholder to improve the speed of the query by taking advantage of client/server binary protocol and make your code more secured.