Basic MySQL Tutorial

Manage Database in MySQL

**Summary**: in this tutorial, you will learn how to manage databases in MySQL. You will learn how to create new databases, remove existing databases and display all databases in the MySQL database server.

Let’s start creating a new database in MySQL.

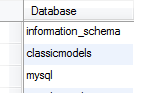
Creating Database

|  |  |  |
| --- | --- | --- |
|  | **CREATE DATABASE [IF NOT EXISTS] database\_name;** | |
|  | | Ex: CREATE DATABASE classicmodels; |

Displaying Databases

The **SHOW DATABASE** statement displays all databases in the MySQL database server.

|  |  |
| --- | --- |
|  | **SHOW DATABASES;** |



Selecting a database to work with

|  |  |
| --- | --- |
| 1 | **USE database\_name**; |

You can select the classicmodels [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx) using the USE statement as follows:

|  |  |
| --- | --- |
| 1 | USE classicmodels; |

Removing Databases

To delete a database, you use the **DROP DATABASE** statement as follows:

|  |  |
| --- | --- |
| 1 | DROP DATABASE [IF EXISTS] database\_name; |

|  |  |
| --- | --- |
| 1  2  3 | CREATE DATABASE IF NOT EXISTS temp\_database;  SHOW DATABASES;  DROP DATABASE IF EXISTS temp\_database; |

In this tutorial, you’ve learned various statements to manage databases in MySQL including creating a new database, removing an existing database, selecting a database to work with and displaying all databases in a MySQL database server.

Understanding MySQL Table Types, or Storage Engines

**Summary**: in this tutorial, you will learn various **MySQL table types,**or storage engines. It is essential to understand the features of each table type in MySQL so that you can use them effectively to maximize the performance of your databases.

MySQL provides various storage engines for its tables as below:

* MyISAM
* InnoDB
* MERGE
* MEMORY (HEAP)
* ARCHIVE
* CSV
* FEDERATED

Each storage engine has its own advantages and disadvantages. It is crucial to understand each storage engine features and choose the most appropriate one for your tables to maximize the performance of the database. In the following sections we will discuss about each storage engine and its features so that you can decide which one to use.

MyISAM

MyISAM extends the former ISAM storage engine. The MyISAM tables are optimized for compression an speed. MyISAM tables are also portable between platforms and OSes.

The size of MyISAM table can be up to 256TB, which is huge. In addition, MyISAM tables can be compressed into read-only tables to save space. At startup, MySQL checks MyISAM tables for corruption and even repair them in case of errors. The MyISAM tables are not transaction-safe.

Before MySQL version 5.5, MyISAM is the default storage engine when you create a table without explicitly specify the storage engine. From version 5.5, MySQL uses InnoDB as the default storage engine.

InnoDB

The InnoDB tables fully support ACID-compliant and transactions. They are also very optimal for performance. InnoDB table supports foreign keys, commit, rollback, roll-and forward operations. The size of the InnoDB table can be up to 64TB.

Like MyISAM, the InnoDB tables are portable between different platforms and OSes. MySQL also checks and repair InnoDB tables, if necessary, at startup.

MERGE

A MERGE table is a virtual table that combines multiple MyISAM tables, which has similar structure, into one table. The MERGE storage engine is also known as the MRG\_MyISAM engine. The MERGE table does not have its own indexes; it uses indexes of the component tables instead.

Using MERGE table, you can speed up performance in joining multiple tables. MySQL only allows you to perform [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx), [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx)and [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx)operations on the MERGE tables. If you use [DROP TABLE statement](http://www.mysqltutorial.org/mysql-drop-table) on a MERGE table, only MERGE specification is removed. The underlying tables will not be affected.

Memory

The memory tables are stored in memory and used hash indexes so that they are faster than MyISAM tables. The lifetime of the data of the memory tables depends on the up time of the database server. The memory storage engine is formerly known as HEAP.

Archive

The archive storage engine allows you to store a large number of records, which for archiving purpose, into a compressed format to save disk space. The archive storage engine compresses a record when it is inserted and decompress it using *zlib* library as it is read.

The archive tables only allow INSERT and SELECT commands. The archive tables do not support indexes, so reading records requires a full table scanning.

CSV

The CSV storage engine stores data in comma-separated values file format. A CSV table brings a convenient way to migrate data into non-SQL applications such as spreadsheet software.

CSV table does not support NULL data type and read operation requires a full table scan.

FEDERATED

The FEDERATED storage engine allows you to manage data from a remote MySQL server without using cluster or replication technology. The local federated table stores no data. When you query data from a local federated table, the data is pull automatically from the remote federated tables.

Choosing MySQL Table Types

You can download the following checklist to choose the most appropriate storage engine, or table type, based on various criteria.

[**MySQL Storage Engine Feature Summary**](http://www.mysqltutorial.org/download/3)**(109.25 kB)**4920 downloads

MySQL Data Types

**Summary***:*in this tutorial, you will learn about **MySQL data types**and how to use them effectively in the MySQL database design.

Database table contains multiple columns with specific data types such as numeric or string. MySQL provides more data types other than just numeric or string. Each data type in MySQL can be determined by the following characteristics:

* Kind of values it can represent.
* The space that takes up and whether the values are fixed-length or variable-length.
* Do the values of the data type can be indexed.
* How MySQL compares the value of a specific data type.

Numeric Data Types

| **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

| **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

| **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 supports many spatial data types that contain various kinds of geometrical and geographical values as shown in the following table:

| **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 |

Creating Tables Using MySQL CREATE TABLE Statement

**Summary***:*in this tutorial, we will show you how to create new tables in a particular database using**MySQL CREATE TABLE** statement.

MySQL CREATE TABLE syntax

In order to create a new table within a database, you use the MySQL CREATE TABLE statement. TheCREATE TABLE statement is one of the most complex statement in MySQL.

|  |  |
| --- | --- |
| 1  2  3 | CREATE TABLE [IF NOT EXISTS] table\_name(          column\_list          ) engine=table\_type |

Let’s examine the syntax in greater detail:

* First, you specify the name of table that you want to create after the CREATE TABLE keywords.  The table name must be unique within a database. The IF NOT EXISTS is an optional part of the statement that allows you to check if the table you are creating already exists in the database. If this is the case, MySQL will ignore the whole statement and it will not create any new table. It is highly recommended that you to use IF NOT EXISTS in every CREATE TABLE statement for preventing from an error of creating a new table that already exists.
* Second*,* you specify a list of columns for the table in the column\_list section. Columns are separated by a comma ( ,).  We will show you how to define columns in more detail in the next section.
* Third, you need to specify the [storage engine](http://www.mysqltutorial.org/understand-mysql-table-types-innodb-myisam.aspx) for the table in the engine clause. You can use any storage engine such as InnoDB, MyISAM, HEAP, EXAMPLE, CSV, ARCHIVE, MERGE FEDERATED or NDBCLUSTER. If you don’t declare the storage engine explicitly, MySQL will use InnoDBby default.

InnoDB became the default storage engine since MySQL version 5.5. The InnoDB table type brings many benefits of relational database management system such as ACID transaction, referential integrity and crash recovery.  In the previous versions, MySQL used MyISAM as the default storage engine.

To define a column for the table in the CREATE TABLE statement, you use the following syntax:

|  |  |
| --- | --- |
| 1  2 | column\_name data\_type[size] [NOT NULL|NULL] [DEFAULT value]  [AUTO\_INCREMENT] |

The most important components of the syntax above are:

* The*column\_name* specifies the name of the column. Each column always associates with  a specific [data type](http://www.mysqltutorial.org/mysql-data-types.aspx) and the size e.g.,   VARCHAR(255).
* TheNOT NULL*or*NULL indicates that the column accepts NULL value or not.
* The DEFAULT*value* is used to specify the default value of the column.
* The AUTO\_INCREMENT indicates that the value of column is increased by one whenever a new row is inserted into the table. Each table has one and only one AUTO\_INCREMENT column.

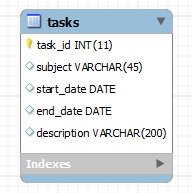
If you want to set particular columns of the table as the primary key, you use the following syntax:

|  |  |
| --- | --- |
| 1 | PRIMARY KEY (col1,col2,...) |

Example of MySQL CREATE TABLE statement

Let’s practice with an example of creating a new table named tasks in our [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx) as follows:

You can use the CREATE TABLE statement to create the tasks table as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE IF NOT EXISTS tasks (    task\_id int(11) NOT NULL AUTO\_INCREMENT,    subject varchar(45) DEFAULT NULL,    start\_date DATE DEFAULT NULL,    end\_date DATE DEFAULT NULL,    description varchar(200) DEFAULT NULL,    PRIMARY KEY (task\_id)  ) ENGINE=InnoDB |

In this tutorial, you have learned how to use MySQL CREATE TABLE to create a new tables within a database.

MySQL Sequence

**Summary**: in this tutorial, we will show you how to use **MySQL sequence** to automatically generate unique numbers for ID columns of tables.

MySQL create sequence

In MySQL, a sequence is a list of integers generated in the ascending order i.e., 1,2,3… Many applications need sequences to generate unique numbers mainly for identification e.g., customer ID in CRM, employee number in HR, equipment number in services management system, etc.

To create a sequence in MySQL automatically, you set the AUTO\_INCREMENT attribute to a column, which typically is primary key column. The following are rules that you must follow when you useAUTO\_INCREMENT attribute:

* Each table has only one AUTO\_INCREMENT column whose data type is typically integer or float which is very rare.
* The   AUTO\_INCREMENT column must be indexed, which means it can be either PRIMARY KEY or UNIQUE index.
* The AUTO\_INCREMENT column must have NOT NULL constraint. When you setAUTO\_INCREMENT attribute to a column, MySQL will make it NOT NULL for you in case you don’t define it explicitly.

MySQL create sequence example

The following example creates employees table whose emp\_no column is AUTO\_INCREMENTcolumn:

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE employees(      emp\_no INT(4) AUTO\_INCREMENT PRIMARY KEY,      first\_name VARCHAR(50),      last\_name  VARCHAR(50)  )ENGINE = INNODB; |

How MySQL sequence works

The AUTO\_INCREMENT column has the following attributes:

* The starting value of an AUTO\_INCREMENT column is 1 and it is increased by 1 when you insertNULL value into the column or when you omit its value in the [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx)statement.
* To obtain the last generated sequence number, you use the LAST\_INSERT\_ID() function. You often use the last insert ID for the subsequent statements e.g., insert data into child tables. The last generated sequence is unique across sessions.In other words, if another connection generates a sequence number, from your connection you can obtain it by using the LAST\_INSERT\_ID()function. For more details on LAST\_INSERT\_ID() function, check it out the [MySQL LAST\_INSERT\_ID() function tutorial](http://www.mysqltutorial.org/mysql-last_insert_id.aspx).
* If you [insert a new row into a table](http://www.mysqltutorial.org/mysql-insert-statement.aspx) and specify a value for the sequence column, MySQL will insert the sequence number if the sequence number does not exist in the column or issue an error if it already exists. If you insert a new value that is greater than the next sequence number, MySQL will use the new value as the starting sequence number and generate a unique sequence number greater than the current one for the next use. This creates gaps in the sequence.
* If you use [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx)statement to update an AUTO\_INCREMENT column to a value that already exists, MySQL will issue a duplicate-key error if the column has a unique index. If you update anAUTO\_INCREMENT column to a value that is larger than the existing values in the column, MySQL will use the next number of the last insert sequence number for the next row e.g., if the last insert sequence number is 3, you update it to 10, the sequence number for the new row is 4. See the example in the below section.
* If you use [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx)statement to delete the last insert row, MySQL may or may not reuse the deleted sequence number depending on the [storage engine](http://www.mysqltutorial.org/understand-mysql-table-types-innodb-myisam.aspx) of the table. A MyISAM table does not reuse the deleted sequence numbers if you delete a row e.g., the last insert id in the table is 10, if you remove it, MySQL still generates the next sequence number which is 11 for the new row. Similar to MyISAM tables, InnoDB tables do use reuse sequence number when rows are deleted.

Once you set AUTO\_INCREMENT attribute for a column, you can [reset auto increment](http://www.mysqltutorial.org/mysql-reset-auto-increment) value in various ways e.g., by using  [ALTER TABLE](http://www.mysqltutorial.org/mysql-alter-table.aspx" \o "MySQL ALTER TABLE)statement.

Let’s practice with the MySQL sequence.

First, insert two new employees into the employees table:

|  |  |
| --- | --- |
| 1  2  3 | INSERT INTO employees(first\_name,last\_name)  VALUES('John','Doe'),        ('Mary','Jane'); |

Second, select data from the employees table:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM employees; |

mysql sequence insert

Third, delete the second employee whose emp\_no is 2:

|  |  |
| --- | --- |
| 1  2 | DELETE FROM employees  WHERE emp\_no = 2; |

mysql sequence delete

Fourth, insert a new employee:

|  |  |
| --- | --- |
| 1  2 | INSERT INTO employees(first\_name,last\_name)  VALUES('Jack','Lee'); |

mysql sequence insert after delete

Because the storage engine of the employees table is InnoDB, it does not reuse the deleted sequence number. The new row has emp\_no 3.

Fifth, update an existing employee with emp\_no 3 to 1:

|  |  |
| --- | --- |
| 1  2  3  4 | UPDATE employees  SET first\_name = 'Joe',      emp\_no = 1  WHERE emp\_no = 3; |

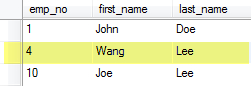
MySQL issued an error of duplicate entry for the primary key. Let’s fix it:

|  |  |
| --- | --- |
| 1  2  3  4 | UPDATE employees  SET first\_name = 'Joe',      emp\_no = 10  WHERE emp\_no = 3; |

mysql sequence update

Sixth, insert a new employee after updating the sequence number to 10:

|  |  |
| --- | --- |
| 1  2 | INSERT INTO employees(first\_name,last\_name)  VALUES('Wang','Lee'); |



The next sequence number of the last insert is 4, therefore MySQL use 4 for the new row instead of 11.

In this tutorial, you have learned how to use MySQL sequence to generate unique numbers for a primary key column by assigning the column AUTO\_INCREMENT attribute.

MySQL Primary Key

**Summary**: in this tutorial, you will learn how to use **MySQL primary key** constraint to create a primary key for the table.

Introduction to MySQL primary key

A primary key is a column or a set of columns that uniquely identifies each row in the table. The following are the rules that you must follow when you define a primary key for a table:

* A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.
* A primary key column cannot contain NULL values. It means that you have to declare the primary key column with NOT NULL attribute. If you don’t, MySQL will force the primary key column as NOT NULL implicitly.
* A table has only one primary key.

Because MySQL works faster with integers, the primary key column’s type should be an integer type e.g.,INT or BIGINT. You can choose a smaller integer type such as TINYINT, SMALLINT, etc., however you should make sure that the range of values of the integer type for the primary key is sufficient for storing all possible rows that the table may have.

A primary key column often has AUTO\_INCREMENT attribute that generates a unique [sequence](http://www.mysqltutorial.org/mysql-sequence/)for the key automatically. The the primary key of the next row is greater than the previous one.

MySQL creates an index named PRIMARY with PRIMARY type for the primary key in a table.

Defining MySQL PRIMARY KEY Constraints

MySQL allows you to to create a primary key by defining a primary key constraint when you create or modify the table.

Defining MySQL PRIMARY KEY constraints using CREATE TABLE statement

MySQL allows you to create the primary key when you create the table by using the [CREATE TABLE](http://www.mysqltutorial.org/mysql-create-table/)statement. To create a PRIMARY KEY constraint for the table, you specify the PRIMARY KEY in the primary key column’s definition.

The following example creates users table whose primary key is user\_id column:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE TABLE users(     user\_id INT AUTO\_INCREMENT PRIMARY KEY,     username VARCHAR(40),     password VARCHAR(255),     email VARCHAR(255)  ); |

You can also specify the PRIMARY KEY at the end of the CREATE TABLE statement as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE roles(     role\_id INT AUTO\_INCREMENT,     role\_name VARCHAR(50),     PRIMARY KEY(role\_id)  ); |

In case the primary key consists of multiple columns, you must specify them at the end of the CREATE TABLE statement. You put a coma-separated list of primary key columns

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE userroles(     user\_id INT NOT NULL,     role\_id INT NOT NULL,     PRIMARY KEY(user\_id,role\_id),     FOREIGN KEY(user\_id) REFERENCES users(user\_id),     FOREIGN KEY(role\_id) REFERENCES roles(role\_id)  ); |

Besides creating the primary key that consists of user\_id and role\_id columns, the statement also created two [foreign key](http://www.mysqltutorial.org/mysql-foreign-key/) constraints.

PRIMARY KEY vs. UNIQUE KEY vs. KEY

A KEY is a synonym for INDEX. You use KEY when you want to [create an index](http://www.mysqltutorial.org/mysql-create-drop-index.aspx) for a column or a set of column that is not a part of a primary key or unique key.

A UNIQUE index creates a constraint for a column whose values must be unique. Unlike the PRIMARYindex**, MySQL allows NULL values in the UNIQUE index**. A table can also have multiple UNIQUEindexes.

For example, the email and username of user in the users table must be unique. You can define UNIQUE indexes for the email and username column as the following  statement:

Add a   UNIQUE index for the username column.

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE users  ADD UNIQUE INDEX username\_unique (username ASC) ; |

Add a UNIQUE index for the email column.

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE users  ADD UNIQUE INDEX  email\_unique (email ASC) ; |

In this tutorial, you have learned how to create a primary key for a new table or add a primary key for an existing table.

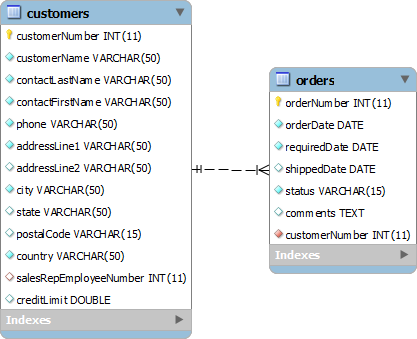
MySQL Foreign Key

**Summary**: in this tutorial, you will learn about **MySQL foreign key** and how to create, add, and drop foreign key constraints in MySQL.

Introduction to MySQL foreign key

A foreign key is a field in a table that matches a field of another table. A foreign key places constraints on data in the related tables that, which enables MySQL to maintain referential integrity.

Let’s take a look at the following database diagram in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx).



We have two tables: customers and orders. Each customer has zero or more orders and each order belongs to only one customer. The relationship between customers table and orders table is one-to-many, and it is established by a foreign key in the orders table specified by thecustomerNumber field. The customerNumber field in the orders table relates to thecustomerNumber primary key field in customers table.

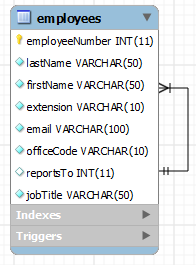
The customers table is called *parent table* or *referenced table*, and the orders table is known as*child table* or *referencing table*.

A foreign key has not only one column but also a set of columns. The columns in the child table often refer to the primary key columns in the parent table.

A table may have more than one foreign key, and each foreign key in the child table can have a differentparent table.

A row in the child table must contain values that exist in the parent table e.g., each order record in theorders table must have a customerNumber that exists in the customers table. Multiple orders can refer to the same customer therefore this relationship is called one (customer) to many (orders), or one-to-many.

Sometimes, the child and parent table is the same table. The foreign key refers back to the primary key ofthe table e.g., the following employees table :



The reportTo column is a foreign key that refers to the employeeNumber column which is the primary key of the employees table to reflect the reporting structure between employees i.e., each employee reports to anther employee and an employee can have zero or more direct reports.

The reportTo foreign key is also known as *recursive* or *self-referencing* foreign key.

Foreign keys enforce referential integrity that helps you maintain the consistency and integrity of the data automatically. For example, you cannot create an order for a non-existent customer.

In addition, you can set up a cascade on delete action for the customerNumber foreign key so that when you delete a customer in the customers table, all the orders associated with the customer are also deleted. This saves you time and efforts of using multiple [DELETE statements](http://www.mysqltutorial.org/mysql-delete-statement.aspx)or a [DELETE JOIN statement](http://www.mysqltutorial.org/mysql-delete-join/).

The same as deletion, you can also define a cascade on update action for the customerNumberforeign key to perform cross-table update without using multiple [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx) statements or an [UPDATE JOIN statement](http://www.mysqltutorial.org/mysql-update-join/).

In MySQL, the InnoDB [storage engine](http://www.mysqltutorial.org/understand-mysql-table-types-innodb-myisam.aspx) supports foreign keys so that you must create InnoDB tables in order to use foreign key constraints.

MySQL create table foreign key

MySQL create foreign key syntax

The following syntax illustrates how to define a foreign key in a child table in [CREATE TABLE](http://www.mysqltutorial.org/mysql-create-table/) statement.



|  |  |
| --- | --- |
| 1  2  3  4  5 | CONSTRAINT constraint\_name  FOREIGN KEY foreign\_key\_name (columns)  REFERENCES parent\_table(columns)  ON DELETE action  ON UPDATE action |

Let’s examine the syntax in greater detail:

* The CONSTRAINT clause allows you to define constraint name for the foreign key constraint. If you omit it, MySQL will generate a name automatically.
* The FOREIGN KEY clause specifies the columns in the child table that refer to primary key columns in the parent table. You can put a foreign key name after FOREIGN KEY clause or leave it to let MySQL to create a name for you. Notice that MySQL automatically creates an index with theforeign\_key\_name name.
* The REFERENCES clause specifies the parent table and its columns to which the columns in thechild table refer. The number of columns in child table and parent table specified in the FOREIGN KEY and REFERENCES must be the same.
* The ON DELETE clause allows you to define what happens to the records in the child table when the records in the parent table are deleted. If you omit the ON DELETE clause and delete a record in the parent table that has records in the child table refer to, MySQL will reject the deletion. In addition, MySQL also provides you with actions so that you can have other options such as [ON DELETE CASCADE](http://www.mysqltutorial.org/mysql-on-delete-cascade/) that lets MySQL to delete records in the child table that refer to a record in theparent table when the record in the parent table is deleted. If you don’t want the related records in the child table to be deleted, you use the ON DELETE SET NULL action instead. MySQL will set the foreign key column values in the child table to NULL when the record in the parent table is deleted, with a condition that the foreign key column in the child table must accept NULL values. Notice that if you use ON DELETE NO ACTION or ON DELETE RESTRICT action, MySQL will reject the deletion.
* The ON UPDATE clause enables you to specify what happens to the rows in the child table when rows in the parent table are updated. You can omit the ON UPDATE clause to let MySQL to reject any update to the rows in the child table when the rows in the parent table are updated. The ON UPDATE CASCADE action allows you to perform cross-table update, and the ON UPDATE SET NULL action resets the values in the rows in the child table to NULL values when the rows in the parent table are updated. The ON UPDATE NO ACTION or UPDATE RESTRICT actions reject any updates.

MySQL create table foreign key example

The following example creates a dbdemo database and two tables: categories and products. Each category has one or more products and each product belongs to only one category. The cat\_idfield in the products table is defined as a foreign key with UPDATE ON CASCADE and DELETE ON RESTRICT actions.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | CREATE DATABASE IF NOT EXISTS dbdemo;    USE dbdemo;    CREATE TABLE categories(     cat\_id int not null auto\_increment primary key,     cat\_name varchar(255) not null,     cat\_description text  ) ENGINE=InnoDB;    CREATE TABLE products(     prd\_id int not null auto\_increment primary key,     prd\_name varchar(355) not null,     prd\_price decimal,     cat\_id int not null,     FOREIGN KEY fk\_cat(cat\_id)     REFERENCES categories(cat\_id)     ON UPDATE CASCADE     ON DELETE RESTRICT  )ENGINE=InnoDB; |

MySQL add foreign key

MySQL add foreign key syntax

To add a foreign key to an existing table, you use the [ALTER TABLE](http://www.mysqltutorial.org/mysql-alter-table.aspx) statement with the foreign key definition syntax above:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ALTER table\_name  ADD CONSTRAINT constraint\_name  FOREIGN KEY foreign\_key\_name(columns)  REFERENCES parent\_table(columns)  ON DELETE action  ON UPDATE action |

MySQL add foreign key example

Now, let’s add a new table named vendors and change the products table to include the vendor id field:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | USE dbdemo;    CREATE TABLE vendors(      vdr\_id int not null auto\_increment primary key,      vdr\_name varchar(255)  )ENGINE=InnoDB;    ALTER TABLE products  ADD COLUMN vdr\_id int not null AFTER cat\_id; |

To add a foreign key to the products table, you use the following statement:



|  |  |
| --- | --- |
| 1  2  3  4  5 | ALTER TABLE products  ADD FOREIGN KEY fk\_vendor(vdr\_id)  REFERENCES vendors(vdr\_id)  ON DELETE NO ACTION  ON UPDATE CASCADE; |

Now, the products table has two foreign keys, one refers to the categories table and another refers to the vendors table.

MySQL drop foreign key

You also use the ALTER TABLE statement to drop foreign key as the following statement:



|  |  |
| --- | --- |
| 1  2 | ALTER TABLE table\_name  DROP FOREIGN KEY constraint\_name |

In the statement above:

* First, you specify the table name from which you want to remove the foreign key.
* Second, you put the constraint name after the DROP FOREIGN KEY clause.

Notice that constraint\_name is the name of the constraint specified when you created or added the foreign key to the table. If you omit it, MySQL generates a constraint name for you.

To obtain the generated constraint name of a table, you use the SHOW CREATE TABLE statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW CREATE TABLE table\_name |

For example, to see the foreign keys of the products table, you use the following statement:



|  |  |
| --- | --- |
| 1 | SHOW CREATE TABLE products |

The following is the output of the statement:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | CREATE TABLE products (    prd\_id int(11) NOT NULL AUTO\_INCREMENT,    prd\_name varchar(355) NOT NULL,    prd\_price decimal(10,0) DEFAULT NULL,    cat\_id int(11) NOT NULL,    vdr\_id int(11) NOT NULL,    PRIMARY KEY (prd\_id),    KEY fk\_cat (cat\_id),    KEY fk\_vendor(vdr\_id),      CONSTRAINT products\_ibfk\_2    FOREIGN KEY (vdr\_id)    REFERENCES vendors (vdr\_id)    ON DELETE NO ACTION    ON UPDATE CASCADE,      CONSTRAINT products\_ibfk\_1    FOREIGN KEY (cat\_id)    REFERENCES categories (cat\_id)    ON UPDATE CASCADE  ) ENGINE=InnoDB; |

The products table has two foreign key constraints: products\_ibfk\_1 andproducts\_ibfk\_2.

You can drop the foreign keys of the products table by using the following statement:



|  |  |
| --- | --- |
| 1  2  3  4  5 | ALTER TABLE products  DROP FOREIGN KEY products\_ibfk\_1;    ALTER TABLE products  DROP FOREIGN KEY products\_ibfk\_2; |

MySQL disable foreign key checks

Sometimes, it is very useful to disable foreign key checks e.g., when you load data into the tables that have foreign keys. If you don’t disable foreign key checks, you have to load data into a proper order i.e., you have to load data into parent tables first and then child tables, which can be tedious. However if you disable the foreign key checks, you can load data into any orders.

Another example is that, unless you disable the foreign key checks, you cannot [drop a table](http://www.mysqltutorial.org/mysql-drop-table) that is referenced by a foreign key constraint. When you drop a table, any constraints that you defined for the table are also removed.

To disable foreign key checks, you use the following statement:



|  |  |
| --- | --- |
| 1 | SET foreign\_key\_checks = 0 |

And of course, you can enable it by using the statement below:



|  |  |
| --- | --- |
| 1 | SET foreign\_key\_checks = 1 |

In this tutorial, we have covered a lot about MySQL foreign key. We also introduced you to some very handy statements that allow you to manage foreign keys effectively in MySQL.

Changing Table Structure Using MySQL ALTER TABLE

**Summary**: in this tutorial, you will learn about the **MySQL ALTER TABLE** statement that changes existing table structure such as adding or removing column, changing column attribute, etc.

MySQL ALTER TABLE syntax

The ALTER TABLE statement is used to change the structure of existing tables. You can use theALTER TABLE statement to add or drop column, change [data type](http://www.mysqltutorial.org/mysql-data-types.aspx) of column, add primary key, rename table and many more. The following illustrates the ALTER TABLE statement syntax:



|  |  |
| --- | --- |
| 1 | ALTER TABLE table\_name action1[,action2,…] |

To alter an existing table:

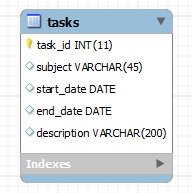
* First, you specify the table name that you want to change after the ALTER TABLE keywords.
* Second, you list a set of actions that you want to apply to the table. An action can be anything such as adding a new column, adding primary key, renaming table, etc. The  ALTER TABLE statement allows you to apply multiple actions in a single ALTER TABLE statement, each action is separated by a comma (,).

Let’s create a new table for practicing the ALTER TABLE statement.

We’re going to create a new table named  tasksin our [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx). The following is the script for creating the tasks table.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE  TABLE tasks (    task\_id INT NOT NULL ,    subject VARCHAR(45) NULL ,    start\_date DATE NULL ,    end\_date DATET 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 statement to set auto-increment attribute for a column

Suppose you want thevalue of the task\_idcolumn to be increased automatically by one whenever you [insert a new record](http://www.mysqltutorial.org/mysql-insert-statement.aspx) into the  tasks table. To do this, you use the ALTER TABLE statement to set the attribute of the task\_id  column to AUTO\_INCREMENT as follows:



|  |  |
| --- | --- |
| 1  2 | ALTER TABLE tasks  CHANGE COLUMN task\_id task\_id INT(11) NOT NULL AUTO\_INCREMENT; |

You can verify the change by adding some records to the tasks table.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | INSERT INTO tasks(subject,                    start\_date,                    end\_date,            description)  VALUES('Learn MySQL ALTER TABLE',         Now(),         Now(),        'Practicing MySQL ALTER TABLE statement');    INSERT INTO tasks(subject,                    start\_date,                    end\_date,                description)  VALUES('Learn MySQL CREATE TABLE',         Now(),         Now(),        'Practicing MySQL CREATE TABLE statement'); |

And you can query data to see if the value of the  task\_id column is  increased by 1 each time you insert a new record:



|  |  |
| --- | --- |
| 1  2 | SELECT task\_id, description  FROM tasks |

MySQL ALTER TABLE - AUTOINCREMENT

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

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



|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE tasks  ADD COLUMN complete DECIMAL(2,1) NULL  AFTER description; |

Using MySQL ALTER TABLE to drop a column from a table

Suppose you don’t want to store the description of tasks in the tasks table and you have to remove it. The following statement allows you to remove the  descriptioncolumn of the tasks table:



|  |  |
| --- | --- |
| 1  2 | ALTER TABLE tasks  DROP COLUMN description; |

Renaming table using MySQL ALTER TABLE statement

You can use the ALTER TABLE statement to rename a table. Notice that before renaming a table, you should take a serious consideration to see if the change affects both database and application layers.

The following statement rename the  tasks table to work\_items:



|  |  |
| --- | --- |
| 1  2 | ALTER TABLE tasks  RENAME TO work\_items; |

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

MySQL DROP TABLE example

We are going to remove the tasks table that we created in the previous tutorial on [creating tables using CREATE TABLE statement](http://www.mysqltutorial.org/mysql-create-table/). In addition, we also remove a non-existent table to practice with theSHOW WARNING statement. The statement to remove the tasks table and a non-existent table is as follows:



|  |  |
| --- | --- |
| 1 | DROP TABLE IF EXISTS tasks, nonexistent\_table; |

If you check the database, you will see that the tasks table was removed. You can check the NOTE, which is generated by MySQL because of non-existent table, by using the SHOW WARNING statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW WARNINGS; |

MySQL DROP TABLE - NOTE

MySQL DROP TABLE LIKE

Image you have a lot of tables whose names start with test in your database and you want to save time by removing all of them using a single DROP TABLE statement. Unfortunately, MySQL does not provide the DROP TABLE LIKE statement that can remove tables based on pattern matching like the following:



|  |  |
| --- | --- |
| 1 | DROP TABLE LIKE '%pattern%' |

However, there are some workarounds. We will discuss one of them here for your reference.

Let’s start creating *test*\* tables for the sake of demonstration.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE IF NOT EXISTS test1(    id int(11) NOT NULL AUTO\_INCREMENT,    PRIMARY KEY(id)  );    CREATE TABLE IF NOT EXISTS test2 LIKE test1;  CREATE TABLE IF NOT EXISTS test3 LIKE test1;  CREATE TABLE IF NOT EXISTS test4 LIKE test1; |

We’ve created four tables named test1, test2, test3 and test4 with the same table structure.

Suppose you want to remove all test\* tables at a time, you can follow the steps below:

First, you declare two variables that accept database schema and a pattern that you want to the tables to match:



|  |  |
| --- | --- |
| 1  2  3 | -- set table schema and pattern matching for tables  SET @schema = 'classicmodels';  SET @pattern = 'test%'; |

Next, you need to build a dynamic DROP TABLE statement:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | -- build dynamic sql (DROP TABLE tbl1, tbl2...;)  SELECT CONCAT('DROP TABLE ',GROUP\_CONCAT(CONCAT(@schema,'.',table\_name)),';')  INTO @droplike  FROM information\_schema.tables  WHERE @schema = database()  AND table\_name LIKE @pattern; |

Basically, the query instructs MySQL to go to the information\_schema table, which contains data on all tables in all databases, and to concatenate all tables in the database @schema (classicmodels *)*that matches the pattern @pattern *( test%)*with the prefix DROP TABLE*.*The GROUP\_CONCAT function creates a comma-separated list of tables.

Then, we can display the dynamic SQL to verify if it works correctly:



|  |  |
| --- | --- |
| 1  2 | -- display the dynamic sql statement  SELECT @droplike; |

mysql drop table like

You can see that it works as expected.

After that, you can execute the statement using [prepared statement in MySQL](http://www.mysqltutorial.org/mysql-prepared-statement.aspx) as follows:



|  |  |
| --- | --- |
| 1  2  3  4 | -- execute dynamic sql  PREPARE stmt FROM @dropcmd;  EXECUTE stmt;  DEALLOCATE PREPARE stmt; |

For more information on MySQL prepared statement, check it out the [MySQL prepared statement](http://www.mysqltutorial.org/mysql-prepared-statement.aspx) tutorial.

Putting it all together.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | -- set table schema and pattern matching for tables  SET @schema = 'classicmodels';  SET @pattern = 'test%';    -- build dynamic sql (DROP TABLE tbl1, tbl2...;)  SELECT CONCAT('DROP TABLE ',GROUP\_CONCAT(CONCAT(@schema,'.',table\_name)),';')  INTO @droplike  FROM information\_schema.tables  WHERE @schema = database()  AND table\_name LIKE @pattern;    -- display the dynamic sql statement  SELECT @droplike;    -- execute dynamic sql  PREPARE stmt FROM @dropcmd;  EXECUTE stmt;  DEALLOCATE PREPARE stmt; |

So if you want to drop multiple tables that have a specific pattern in a database, you just use the script above to save time. All you need to do is replacing the *pattern* and the*database schema* in @patternand @schema variables. If you often have to deal with this task, you can always develop a [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) based on the script and reuse the stored procedure in the future.

In this tutorial, we’ve shown you how to use the DROP TABLE statement to remove existing tables in a particular database. We also discussed about a workaround that allows you to use the DROP TABLE statement to remove tables based on pattern matching.

Create MySQL temporary table

Like the [CREATE TABLE statement](http://www.mysqltutorial.org/mysql-create-table/), MySQL provides many options to create a temporary table. To create a temporary table, you just add the TEMPORARY keyword to the CREATE TABLE statement.

For example, the following statement creates a top 10 customers by revenue temporary table based on the result set of a SELECT statement:

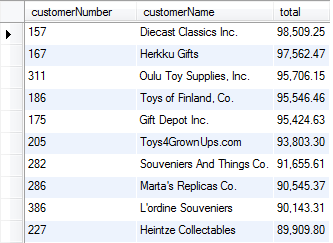


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TEMPORARY TABLE top10customers  SELECT p.customerNumber,         c.customerName,         FORMAT(SUM(p.amount),2) total  FROM payments p  INNER JOIN customers c ON c.customerNumber = p.customerNumber  GROUP BY p.customerNumber  ORDER BY total DESC  LIMIT 10 |

Now, you can query data from the top10customers temporary table as from a permanent table:



|  |  |
| --- | --- |
| 1 | SELECT \* FROM top10customers |



Drop MySQL temporary table

You can use the [DROP TABLE statement](http://www.mysqltutorial.org/mysql-drop-table) to remove temporary tables however it is good practice to use the DROP TEMPORARY TABLE statement instead. Because the DROP TEMPORARY TABLEremoves only temporary tables, not the permanent tables. In addition, the DROP TEMPORARY TABLEstatement helps you avoid the mistake of removing a permanent table when you name your temporary table the same as the name of the permanent table.

For example, to remove the top10customers temporary table, you use the following statement:



|  |  |
| --- | --- |
| 1 | DROP TEMPORARY TABLE top10customers |

Notice that if you try to remove a permanent table with the DROP TEMPORARY TABLE statement, you will get an error message saying that the table you are trying drop is unknown.

Note if you develop an application that uses a connection pooling or persistent connections, it is not guaranteed that the temporary tables are removed automatically when your application is terminated. Because the database connection that the application used may be still open and is placed in a connection pool for other clients to reuse it. This means you should always remove the temporary tables that you created whenever you are done with them.

In this tutorial, you have learned about MySQL temporary table and its characteristic. We also gave you an example of how to create, use and drop a temporary table.

MySQL Managing Database Index

**Summary**: in this tutorial, you will learn how to work with **MySQL index** and how to take advantages of  the index to speed up the data retrieval. We will introduce you several useful statements that allows you to manage MySQL indexes.

Database index, or just index, helps speed up the retrieval of data from tables. When you query data from a table, first MySQL checks if the indexes exist, then MySQL uses the indexes to select exact physical corresponding rows of the table instead of scanning the whole table.

A database index is similar to an index of a book. If you want to find a topic, you look up in the index first, and then you open the page that has the topic without scanning the whole book.

It is highly recommended that you should create index on columns of table from which you often query the data. Notice that all primary key columns are in the primary index of  the table automatically.

If index helps speed up the querying data, why don’t we use indexes for all columns? If you create an index for every column, MySQL has to build and maintain the index table. Whenever a change is made to the records of the table, MySQL has to rebuild the index, which takes time as well as decreases the performance of the database server.

Creating MySQL Index

You often create indexes when you create tables. MySQL automatically add any column that is declared as PRIMARY KEY, KEY, UNIQUE or INDEX to the index. In addition, you can add indexes to the tables that already have data.

In order to create indexes, you use the CREATE INDEX statement. The following illustrates the syntax of the CREATE INDEX statement:



|  |  |
| --- | --- |
| 1  2  3 | 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 type or storage engine:

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

Then, you name the index and its type after the USING keyword such as BTREE, HASH or RTREEalso based on the storage engine of the table.

Here are the storage engines of the table with the corresponding allowed index types:

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

Third, you declare table name and a list columns that you want to add to the index.

Example of creating index in MySQL

In the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx), you can add  officeCode column of  the employees table to the index by using the CREATE INDEX statement as follows:



|  |  |
| --- | --- |
| 1 | CREATE INDEX officeCode ON employees(officeCode) |

Removing Indexes

Besides creating index, you can also remove index by using the DROP INDEX statement. Interestingly, the DROP INDEX statement is also mapped to [ALTER TABLE statement](http://www.mysqltutorial.org/mysql-alter-table.aspx). The following is the syntax of removing the index:



|  |  |
| --- | --- |
| 1 | DROP INDEX index\_name ON table\_name |

For example, if you want to drop index officeCode of the employees table*,*which we have created above, you can execute following query:



|  |  |
| --- | --- |
| 1 | DROP INDEX officeCode ON employees |

In this tutorial, you’ve learned about indexes and how to manage MySQL index including creating and removing indexes.

Using MySQL SELECT Statement to Query Data

**Summary**: in this tutorial, you will learn how to**MySQL SELECT**statement to query data from adatabase table.

The MySQL SELECT statement allows you to retrieve zero or more rows from tables or [views](http://www.mysqltutorial.org/mysql-views-tutorial.aspx). The SELECT statement is the one of the most commonly used queries in MySQL.

The SELECT statement returns a result that is a combination of columns and rows, which is also known as a result set.

MySQL SELECT syntax

The following illustrates the syntax of the SELECT statement:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT column\_1,column\_2...  FROM table\_1  [INNER | LEFT |RIGHT] JOIN table\_2 ON conditions  WHERE conditions  GROUP BY group  HAVING group\_conditions  ORDER BY column\_1 [ASC | DESC]  LIMIT offset, row\_count |

The SELECT statement is composed of several clauses:

* SELECT chooses which columns of  the table you want to get the data.
* FROM specifies the table from which you get the data.
* [JOIN gets data from multiple table based on certain join conditions.](http://www.mysqltutorial.org/mysql-inner-join.aspx)
* [WHERE filters rows to select](http://www.mysqltutorial.org/mysql-where/).
* [GROUP BY group rows to apply aggregate functions on each group.](http://www.mysqltutorial.org/mysql-group-by.aspx)
* [HAVING filters group based on groups defined by GROUP BY clause](http://www.mysqltutorial.org/mysql-having.aspx).
* [ORDER BY specifies the order of the returned result set.](http://www.mysqltutorial.org/mysql-order-by)
* [LIMIT constrains number of returned rows.](http://www.mysqltutorial.org/mysql-limit.aspx)

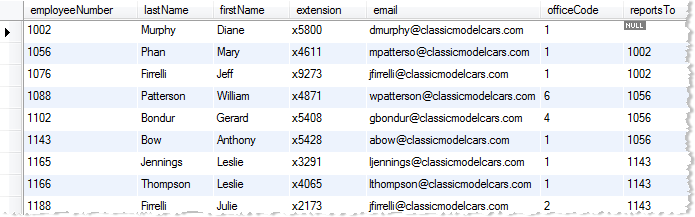
You will learn about each clause in more detail in the next tutorial. In this tutorial, we are going to focus on the simple form of the SELECT statement.

MySQL SELECT Examples

To select all columns and rows from the employees table, you use the following query:



|  |  |
| --- | --- |
| 1 | SELECT \* FROM employees |

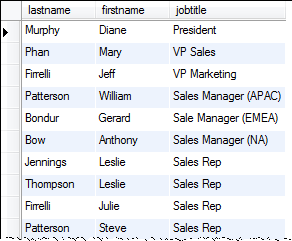


The asterisk (\*) notation is a shorthand of selecting all columns in the table.

The SELECT statement also allows you to query partial data of a table by specifying a list of comma-separated columns in the SELECT clause. For example, if you want to view only first name,last name and job title of the employees,  you use the following query:



|  |  |
| --- | --- |
| 1  2  3  4 | SELECT lastname,         firstname,         jobtitle  FROM employees |



In this tutorial, you’ve learned about the basic MySQL SELECT statement to query data from a database table in MySQL

Filter Rows Using MySQL WHERE

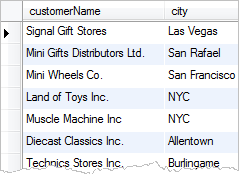
**Summary*:*** you will learn how to use **MySQL WHERE** clause to filter rows returned from the SELECT statement.

If you use the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx)to query the data from a table without the WHERE clause, you will get all rows in the  table, which sometimes brings more data than you need. The WHERE clause allows you to specify exact rows to select based on given conditions e.g., find all customers in the U.S.

The following query selects all customers whose country is U.S. from the customers table. We use the WHERE clause to filter the customers. In the WHERE clause, we compare the values of thecountry column with the USA literal string.



|  |  |
| --- | --- |
| 1  2  3 | SELECT customerName, city  FROM customers  WHERE country = 'USA'; |



You can form a simple condition like the query above, or a very complex one that combines multiple expressions with logical operators such as AND and OR. For example, to find all customers in the U.S . and also in the New York city, you use the following query:



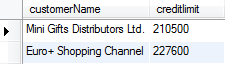
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT customerName, city  FROM customers  WHERE country = 'USA' AND        city    = 'NYC'; |



You can test the condition for not only equality but also inequality. For example, to find all customers whose credit limit is greater than 200.000 USD, you use the following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT customerName, creditlimit  FROM customers  WHERE creditlimit > 200000; |



There are several useful operators that you can use in the WHERE clause to form more practical queries such as:

* [BETWEEN](http://www.mysqltutorial.org/sql-between-mysql.aspx)selects values within a range of values.
* [LIKE](http://www.mysqltutorial.org/sql-like-mysql.aspx)matches value based on pattern matching.
* [IN](http://www.mysqltutorial.org/sql-in.aspx)specifies if the value matches any value in a list.
* IS NULL checks if the value is NULL

The WHERE clause is used not only with the SELECT statement but also other SQL statements to filter rows such as [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx)and [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx).

MySQL ORDER BY

**Summary**: in this tutorial, you will learn how to sort a result set by using **MySQL ORDER BY** clause.

Introduction to MySQL ORDER BY clause

When you use the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) to query data from a table, the result set is not sorted in a specific order. To sort the result set, you use the ORDER BY clause. The ORDER BY clause allows you to:

* Sort a result set by a single column or multiple columns.
* Sort a result set by different columns in ascending or descending order.

The following illustrates the syntax of the ORDER BY clause:



|  |  |
| --- | --- |
| 1  2  3 | SELECT col1, col2,...  FROM tbl  ORDER BY col1 [ASC|DESC], col2 [ASC|DESC],... |

The ASC stands for ascending and the DESC stands for descending. By default, the ORDER BY clause sorts the result set in ascending order  if you don’t  specify ASC or DESC explicitly

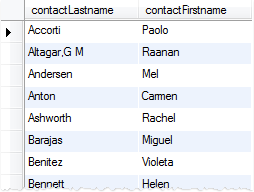
Let’s practice with some examples of using the ORDER BY clause.

MySQL ORDER BY examples

The following query selects contacts from the customers table and sorts the contacts by last name in ascending order.



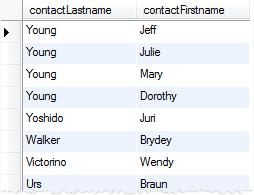
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT contactLastname,         contactFirstname  FROM customers  ORDER BY contactLastname; |



If you want to sort the contact by last name in descending order, you specify the DESC after thecontactLastname column in the ORDER BY clause as the following query:



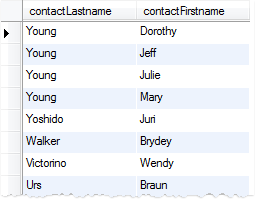
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT contactLastname,         contactFirstname  FROM customers  ORDER BY contactLastname DESC |



If you want to sort the contacts by last name in descending order and first name in ascending order, you specify both  DESC and ASC in the corresponding column as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT contactLastname,         contactFirstname  FROM customers  ORDER BY contactLastname DESC,           contactFirstname ASC; |



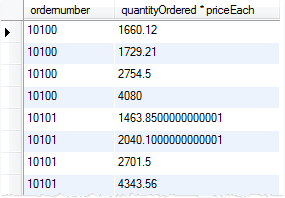
In the query above, the ORDER BY clause sorts the result set by  last name in descending order first, and then sorts the sorted result set by first name in ascending order to produce the final result set.

MySQL ORDER BY sort by an expression example

The ORDER BY clause also allows you to sort the result set based on an expression. The following query selects the order line items from the orderdetails table. It calculates the subtotal for each line item and sorts the result set based on the order number and subtotal.



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT ordernumber,         quantityOrdered \* priceEach  FROM orderdetails  ORDER BY ordernumber,           quantityOrdered \* priceEach |



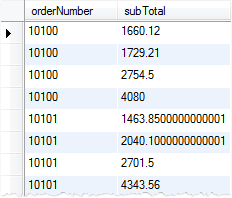
To make the result more readable, you can use a [column alias](http://www.mysqltutorial.org/mysql-alias/), and sort the result based on the column alias.



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT orderNumber,         quantityOrdered \* priceEach AS subTotal  FROM orderdetails  ORDER BY orderNumber,           subTotal; |



|  |  |
| --- | --- |
| 1 |  |



In the query above, we used subtotal as the column alias for the quantityOrdered \* priceEach expression and sorted the result set based on the subtotal alias.

If you use  a [function](http://www.mysqltutorial.org/mysql-functions.aspx)that returns a value whose data type is different from the column’s and sort the result based on the alias, the ORDER BY clause will sort the result set based on the return type of the function, which may not work as expected.

For example, if you use the DATE\_FORMAT function to format the date values and sort the result set based on the strings returned by the DATE\_FORMAT function, the order is not always correct. For more information, check it out the example in the [DATE\_FORMAT](http://www.mysqltutorial.org/mysql-date_format/)function tutorial.

MySQL ORDER BY with customer sort order

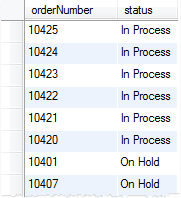
The ORDER BY clause enables you to define your own custom sort order for the values in a column using the FIELD() function. For example, if you want to sort the orders based on the following status by the following order:

1. In Process
2. On Hold
3. Cancelled
4. Resolved
5. Disputed
6. Shipped

You can use the FIELD() function to map those values to a list of numeric values and use the numbers for sorting; See the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT orderNumber, status  FROM orders  ORDER BY FIELD(status, 'In Process',                         'On Hold',                         'Cancelled',                         'Resolved',                         'Disputed',                         'Shipped'); |



In this tutorial, we’ve shown you various techniques to sort a result set by using the MySQL ORDER BYclause in the   SELECT statement.

How to Use MySQL DISTINCT to Eliminate Duplicate Rows

**Summary**: in this tutorial, you will learn how to use **MySQL DISTINCT**operatorwith the SELECTstatement to eliminate duplicate rows in the result set.

When querying data from a table, you may get duplicate rows. In order to remove the duplicate rows, you use the DISTINCT operator in the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx). The syntax of using the DISTINCToperator is as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT DISTINCT columns  FROM table\_name  WHERE where\_conditions |

Let’s take a look a simple example of using the DISTINCT operator to select the distinct last names of employees from the employees table.

First, we query the last names of employees from the employees table using the SELECTstatement as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT lastname  FROM employees  ORDER BY lastname |



Some employees has the same last name  Bondur, Firrelli, etc. To remove the duplicate last names, you use the DISTINCT operator in the SELECT clause as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT DISTINCT lastname  FROM employees  ORDER BY lastname |

  
The duplicate last names are eliminated in the result set when we use the DISTINCT operator.

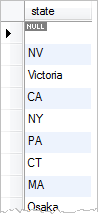
MySQL DISTINCT and NULL values

If a column has NULL values and you use the DISTINCT operator for that column, MySQL will keep one NULL value and eliminate the other because the DISTINCT operator treats all NULL values as the same value.

For example, in the customers table, we have many rows with state column has NULL values. When we use the DISTINCT operator to query states of customers, we will see distinct states plus a NULL value as the following query:



|  |  |
| --- | --- |
| 1  2 | SELECT DISTINCT state  FROM customers |



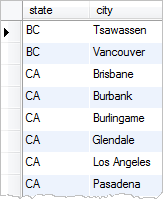
MySQL DISTINCT with multiple columns

You can use the DISTINCT operator with more than one column. The combination of all columns will be used to define the uniqueness of the row in the result set.

For example, to get the unique combination of city and state from the customers table, you use the following query:



|  |  |
| --- | --- |
| 1  2  3  4 | SELECT DISTINCT state, city  FROM customers  WHERE state IS NOT NULL  ORDER BY state, city |



Without the DISTINCT operator, you will get duplicate combination state and city as follows:



|  |  |
| --- | --- |
| 1  2  3  4 | SELECT state, city  FROM customers  WHERE state IS NOT NULL  ORDER BY state, city |



DISTINCT vs. GROUP BY Clause

If you use the [GROUP BY clause](http://www.mysqltutorial.org/mysql-group-by.aspx) in the SELECT statement without using [aggregate functions](http://www.mysqltutorial.org/mysql-functions/mysql-aggregate-functions/), theGROUP BY clause will behave like the DISTINCT operator. The following queries produce the same result set:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT DISTINCT state  FROM customers;    SELECT state  FROM customers  GROUP BY state; |

The difference between DISTINCT operator and GROUP BY clause is that the GROUP BY clause sorts the result set whereas the DISTINCT operator does not.

MySQL DISTINCT and COUNT aggregate function

The DISTINCT operator is used with the [COUNT](http://www.mysqltutorial.org/mysql-count/) function to count unique records in a table. In this case, it ignores the NULL values. For example, to count the unique states of customers in the U.S., you use the following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT COUNT(DISTINCT state)  FROM customers  WHERE country = 'USA'; |

MySQL Distinct Count

In this tutorial, we have shown you various ways of using MySQL DISTINCT operator such as eliminating duplicate records and counting non-NULL values.

Using MySQL LIMIT

**Summary***:*in this tutorial, you will learn how to use **MySQL LIMIT** clause to select records from the beginning, middle and end of a result set.

MySQL LIMIT syntax

The LIMIT clause is used in the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) to constrain the number of rows in a result set. The LIMIT clause accepts one or two arguments. The values of both arguments must be zero or positive integer constants.

The following illustrates the LIMIT clause syntax with 2 arguments:



|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM tbl  LIMIT offset, count |

Let’s see what the offset and count mean in the LIMIT clause:

* The offset specifies the offset of the first row to return. The offset of the first row is 0, not 1.
* The count specifies maximum number of rows to return.

When you use LIMIT with one argument, this argument will be used to specifies the maximum number of rows to return from the beginning of the result set.



|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM tbl  LIMIT count |

The query above is equivalent to the following query with the LIMIT clause that accepts two arguments:



|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM tbl  LIMIT 0, count |

Using MySQL LIMIT to get the first N rows

You can use the LIMIT clause to select the first N rows in a table as follows:

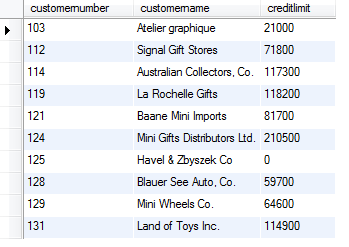


|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM tbl  LIMIT N |

For example, to select the first 10 customers, you use the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT customernumber,         customername,         creditlimit  FROM customers  LIMIT 10; |



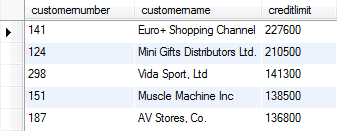
Using MySQL LIMIT to get the highest and lowest values

The LIMIT clause often used with [ORDER BY clause](http://www.mysqltutorial.org/mysql-order-by). First, you use the ORDER BY clause to sort the result set based on a certain criteria, and then you use LIMIT clause to find lowest or highest values.

For example, to select 5 customers who have the highest credit limit, you use the following query:



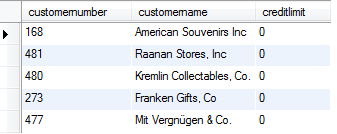
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT customernumber,         customername,         creditlimit  FROM customers  ORDER BY creditlimit DESC  LIMIT 5; |



And the following query returns 5 customers who have the lowest credit limit:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT customernumber,         customername,         creditlimit  FROM customers  ORDER BY creditlimit ASC  LIMIT 5; |



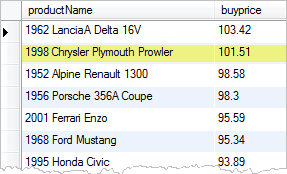
Using MySQL LIMIT to get the N highest values

One of the toughest questions in MySQL is how to select the N highest values in a result set e.g., select the second most expensive product, which you cannot use [MAX or MIN functions](http://www.mysqltutorial.org/mysql-functions/mysql-aggregate-functions/) to answer. However, you can use MySQL LIMIT to answer those kinds of questions.

Let’s take a look at the products result set of the following query:



|  |  |
| --- | --- |
| 1  2  3  4 | SELECT productName,         buyprice  FROM products  ORDER BY buyprice DESC; |



Our task is to get the highlight product, which is the second most expensive product in the products result set. In order to do so, you use LIMIT clause to select 1 row from the second row as the following query: (notice that the offset starts from zero)



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productName,         buyprice  FROM products  ORDER BY buyprice DESC  LIMIT 1, 1 |

mysql limit most second expensive product

Querying Data with MySQL IN Operator

**Summary**: in this tutorial, you will learn how to use the **MySQL IN** operator that determines if a specified value matches any value a list or a subquery.

Introduction to the MySQL IN Operator

The IN operator allows you to determine if a specified value matches any one of a list or a [subquery](http://www.mysqltutorial.org/mysql-subquery/). The following illustrates the syntax of the IN operator.



|  |  |
| --- | --- |
| 1  2  3 | SELECT column\_list  FROM table\_name  WHERE (expr|column) IN ('value1','value2',...) |

In the query above:

* You can use a column or an expression ( expr) with the IN operator in the [WHERE clause](http://www.mysqltutorial.org/mysql-where/) of the [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx)statement.
* The values in the list must be separated by a comma (,)
* The IN operator can also be used in the WHERE clause of other statements such as [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx),[UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx" \o "MySQL UPDATE), [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx), etc.

The IN operator returns 1 if the value of the  column orthe result of the expr expression is equal to any value in the list, otherwise it returns 0.

When the values in the list are all constants:

* First, MySQL evaluates the values based on the type of the column *or*result of the expr.
* Second, MySQL sorts the values.
* Third, MySQL searches for values using binary search algorithm which is very fast.

Therefore a query that uses the IN operator with a list of constants will perform very fast.

If the expr or any value in the list is NULL, the IN operator returns NULL.

You can combine the IN operator with the NOT operator to determine if a value does not match any value in a list or a subquery.

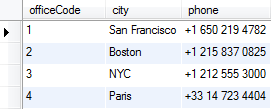
Let’s practice with some examples of using the IN operator.

MySQL IN examples

If you want to find out all offices which locates in the U.S. and France, you can use the IN operator as the following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT officeCode, city, phone  FROM offices  WHERE country IN ('USA','France') |

  
You can achieve the same result with the OR operator as the following query:



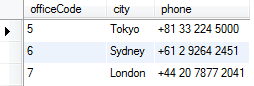
|  |  |
| --- | --- |
| 1  2  3 | SELECT officeCode, city, phone  FROM offices  WHERE country = 'USA' OR country = 'France' |

In case the list has many values, you have to construct a very long statement with multipleOR operators. Hence the IN operator allows you to shorten the query and make the query more readable.

To get offices that does not locate in USA and France, you can use NOT IN in the WHERE clause as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT officeCode, city, phone  FROM offices  WHERE country NOT IN ('USA','France') |

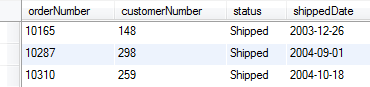


MySQL IN with subquery

The IN operator is often used with a [subquery](http://www.mysqltutorial.org/mysql-subquery/). For example, if you want to find orderwhose total amount is greater than $60K, you can use the IN operator as the following  query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | SELECT orderNumber,         customerNumber,         status,         shippedDate  FROM orders  WHERE orderNumber IN (      SELECT orderNumber      FROM   orderDetails      GROUP BY orderNumber      HAVING SUM(quantityOrdered \* priceEach) > 60000) |



In this tutorial, we have shown you how to use MySQL IN operator to determine if a value matches any value in a list or a subquery.

MySQL BETWEEN Operator Explained

**Summary**: in this tutorial, you will learn how to use **MySQL BETWEEN** operator to specify a range to test.

Introduction to MySQL BETWEEN Operator

The BETWEEN operator allows you to specify a range to test. The following illustrates the syntax of theBETWEEN operator:



|  |  |
| --- | --- |
| 1 | expr (NOT) BETWEEN begin\_expr AND end\_expr |

In the expression above:

* All expressions: expr*,* begin\_expr and end\_expr must return values with the same data type.
* The BETWEEN operator returns 1 if the value of the expr  is greater than or equal to (>=) thevalue of begin\_expr and less than or equal to (<= ) the value of end\_expr*,*otherwise it returns 0.
* The NOT BETWEEN returns 1 if the value of expr  is less than (<) the value of begin\_expror greater than the value of end\_expr*, o*therwise it returns 0.
* If any expression above is NULL, the BETWEEN returns NULL.

The BETWEEN operator is typically used in the [WHERE clause](http://www.mysqltutorial.org/mysql-where/) of [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx),[UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx) and [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx) statements.

MySQL BETWEEN examples

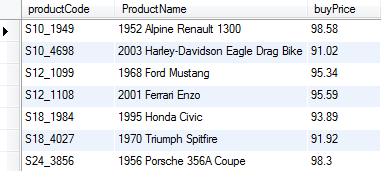
Let’s practice with some examples of using the BETWEEN operator.

MySQL BETWEEN with number examples

Suppose you want to find product whose buy price within the range of $90 and $100, you can use the BETWEEN operator as the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productCode,         productName,         buyPrice  FROM products  WHERE buyPrice BETWEEN 90 AND 100 |



You can achieve the same result by using the greater than or equal ( >=) and less than or equal (<=) operators as the following query:

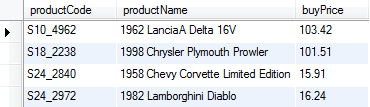


|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productCode,         productName,         buyPrice  FROM products  WHERE buyPrice >= 90 AND buyPrice <= 100 |

To find the product whose buy price is out of the range of $20 and $100, you use combine theBETWEEN operator with the NOT operator as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productCode,         productName,         buyPrice  FROM products  WHERE buyPrice NOT BETWEEN 20 AND 100 |



The query above is equivalent to the following query that uses the comparison operators, greater thanoperator ( >) and less than operator ( <) and a logical operator OR.



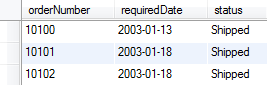
|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productCode,         productName,         buyPrice  FROM products  WHERE buyPrice < 20 OR buyPrice > 100 |

MySQL BETWEEN with dates example

When you use the BETWEEN operator with date values, to get the best result, you should use the[CAST function](http://www.mysqltutorial.org/mysql-cast/) to explicitly convert the type of column or expression to the DATE type. For example, to get the orders whose required date is from 01/01/2003 to 01/31/2003, you use the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SELECT orderNumber,         requiredDate,         status  FROM orders  WHERE requireddate  BETWEEN CAST('2003-01-01' AS DATE) AND      CAST('2003-01-31' AS DATE) |



In the query above, because the data type of the required date column is DATE so we used the CASTfunction to convert the literal strings ‘ 2003-01-01‘ and ‘ 2003-12-31‘ to the DATE data type.

In this tutorial, you have learned how to use the BETWEEN operator to test if a value falls within a range of values. You also learn how to combine the BETWEEN operator with the NOT operator to select data whose value that are not in a range of values.

Using MySQL LIKE Operator to Select Data Based On Patterns

**Summary***:* in this tutorial, you will learn how to use **MySQL LIKE**operator to select data based on patterns.

The LIKE operator is commonly used to select data based on patterns. Using the LIKE operatorin appropriate way is essential to increase the query performance.

The LIKE operator allows you to select data from a table based on a specified pattern. Therefore theLIKE operator is often used in the [WHERE clause](http://www.mysqltutorial.org/mysql-where/) of the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx).

MySQL provides two wildcard characters for using with the LIKE operator, the percentage % and underscore \_.

* The percentage ( %) wildcard allows you to match any string of zero or more characters.
* The underscore ( \_) wildcard allows you to match any single character.

MySQL LIKE examples

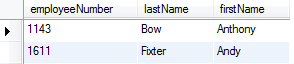
Let’s practice with some examples of how to use the LIKE operator.

MySQL LIKE with percentage (%) wildcard

Suppose you want to search for employee whose first name starts with character ‘ a‘, you can use the percentage wildcard ( %) at the end of the pattern as follows:



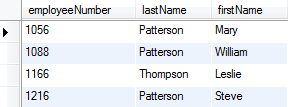
|  |  |
| --- | --- |
| 1  2  3 | SELECT employeeNumber, lastName, firstName  FROM employees  WHERE firstName LIKE 'a%' |

  
MySQL scans the whole employees table to find employee whose first name starts with character ‘a’ and followed by any number of characters.

To search for employee whose last name ends with ‘ on‘ string e.g., Patterson, Thompson, you can use the % wildcard at the beginning of the pattern as the following query:



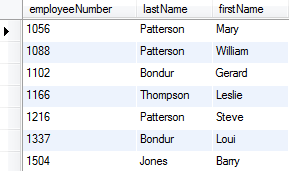
|  |  |
| --- | --- |
| 1  2  3 | SELECT employeeNumber, lastName, firstName  FROM employees  WHERE lastName LIKE '%on' |



If you know the searched string is embedded inside in the column, you can use the percentage ( %) wildcard at the beginning and the end of the pattern. For example, to find all employees whose last names contain ‘ on‘ string, you can execute following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT employeeNumber, lastName, firstName  FROM employees  WHERE lastname LIKE '%on%' |



MySQL LIKE with underscore( \_) wildcard

To find employee whose first name starts with T, ends with m and contains any single character between e.g., Tom, Tim, you use the underscore wildcard to construct the pattern as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT employeeNumber, lastName, firstName  FROM employees  WHERE firstname LIKE 'T\_m' |

mysql like example 4

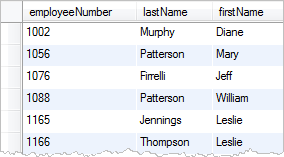
MySQL LIKE operator with NOT operator

The MySQL allows you to combine the NOT operator with the LIKE operator to find string that does not match a specific pattern.

Suppose you want to search for employee whose last name does not start with character ‘ B‘, you can use the NOT LIKE with the pattern as the following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT employeeNumber, lastName, firstName  FROM employees  WHERE lastName NOT LIKE 'B%' |

  
Notice that the pattern is not case sensitive with the LIKE operator therefore the ‘b%’ and ‘B%’ patterns produce the same result.

MySQL LIKE with ESCAPE clause

Sometimes the pattern, which you want to match, contains wildcard character e.g., 10%, \_20… etc. In this case, you can use the ESCAPE clause to specify the escape character so that MySQL interprets the wildcard character as literal character. If you don’t specify the escape character explicitly, the backslash character ‘ \‘ is the default escape character.

For example, if you want to find product whose product code contains string \_20, you can perform following query:

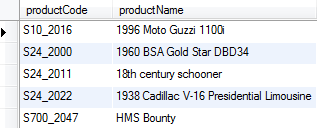


|  |  |
| --- | --- |
| 1  2  3 | SELECT productCode, productName  FROM products  WHERE productCode LIKE '%\\_20%' |

Or specify a different escape character e.g., ‘ $‘ by using the ESCAPE clause:



|  |  |
| --- | --- |
| 1  2  3 | SELECT productCode, productName  FROM products  WHERE productCode LIKE '%$\_20%' ESCAPE '$' |



The pattern %$\_20% matches any string that contains \_20 string.

The LIKE operator forces MySQL to scan the whole table to find the matching rows therefore it does not allow the database engine to use[index](http://www.mysqltutorial.org/mysql-create-drop-index.aspx) for fast searching. As the result, the performance of the query that uses the LIKE operator degrades when you query data from a table with a large number of rows.

In this tutorial, you have learned how to use the LIKE operator to query data based on patterns, which is more flexible than using comparison operators.

MySQL Alias

**Summary**: in this tutorial, you will learn how to use **MySQL alias** to improve the readability of the queries.

MySQL supports two kinds of aliases which are known as column alias and table alias. Let’s examine each kind of alias in detail.

MySQL alias for columns

Sometimes the names of columns are so technical that make the query’s output difficult to understand. To give a column a descriptive name, you use a column alias. The following illustrates how to use the column alias:

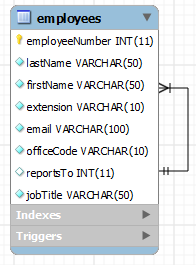


|  |  |
| --- | --- |
| 1  2 | SELECT [col1 | expression] AS `descriptive name`  FROM table\_name |

To give a column an alias, you use the AS keyword followed by the alias. If the alias contains space, you must quote the it as shown in the syntax. Because the AS keyword is optional, you can omit it in the statement.

Note that you can also give an expression an alias.

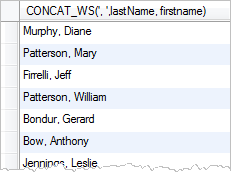
Let’s look at the employees table in the [sample database.](http://www.mysqltutorial.org/mysql-sample-database.aspx)



The following query selects first names and last names of employees, and combine them to produce the full names. The CONCAT\_WS function is used to [concatenate](http://www.mysqltutorial.org/sql-concat-in-mysql.aspx)first name and last name.



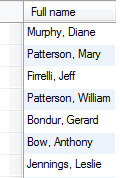
|  |  |
| --- | --- |
| 1  2 | SELECT CONCAT\_WS(', ',lastName, firstname)  FROM employees; |



The column heading is quite difficult to read. You can assign the heading of the output a column alias to make it more readable as the following query:



|  |  |
| --- | --- |
| 1  2 | SELECT CONCAT\_WS(', ',lastName, firstname) AS `Full name`  FROM employees; |



In MySQL, you can use the column alias in the [ORDER BY](http://www.mysqltutorial.org/mysql-order-by/), [GROUP BY](http://www.mysqltutorial.org/mysql-group-by.aspx) and [HAVING](http://www.mysqltutorial.org/mysql-having.aspx)clauses to refer to the column.

The following query uses the column alias in the ORDER BY clause to sort the employee’s full names alphabetically:



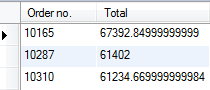
|  |  |
| --- | --- |
| 1  2  3 | SELECT CONCAT\_WS(', ',lastName, firstname) `Full name`  FROM employees  ORDER BY `Full name`; |



The following statement selects the *order* whose total amount is greater than 60000. It uses column aliases in GROUP BY and HAVING clauses.



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT orderNumber `Order no.`,  SUM(priceEach \* quantityOrdered) Total  FROM orderDetails  GROUP BY `Order no.`  HAVING total > 60000; |



Notice that you cannot use column alias in the [WHERE clause](http://www.mysqltutorial.org/mysql-where/). The reason is that when MySQL evaluates the WHERE clause, the values of columns specified in the [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) clause may not be determined yet.

MySQL alias for tables

An alias also gives a table a different name. You assign a table an alias by using the AS keyword as the following syntax:

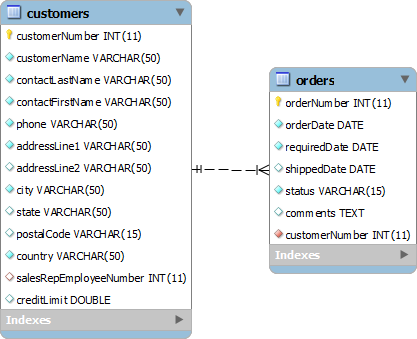


|  |  |
| --- | --- |
| 1 | table\_name AS table\_alias |

The alias for the table is called table alias. Like the column alias, the AS keyword is optional so you can omit it.

You often use the table alias in the statement that contains [INNER JOIN](http://www.mysqltutorial.org/mysql-inner-join.aspx), [LEFT JOIN](http://www.mysqltutorial.org/mysql-left-join.aspx), [self join](http://www.mysqltutorial.org/mysql-self-join/) clauses, and in [subqueries](http://www.mysqltutorial.org/mysql-subquery/).

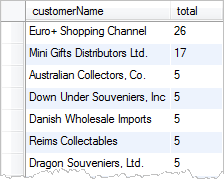
Let’s look at the customers and orders tables:



See the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT customerName,  COUNT(o.orderNumber) total  FROM customers c  INNER JOIN orders o ON c.customerNumber = o.customerNumber  GROUP BY customerName  ORDER BY total DESC |



The query above selects customer name and the number of orders from the customers andorders tables. It uses c as a table alias for the customers table and o as a table alias for theorders table. The columns in the customers and orders tables is referred via the tablealiases.

If you do not use alias in the query above, you have to use the table name to refer to its columns, which makes the query lengthy and less readable.

In this tutorial, we have shown you how to use MySQL alias to make your query easy to read and simple to understand.

Combining Result Sets by Using MySQL UNION

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

MySQL UNION Operator

MySQL UNION operator allows you to combine two or more result sets from multiple tables into a single result set. The syntax of the MySQL UNION is as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT column1, column2  UNION [DISTINCT | ALL]  SELECT column1, column2  UNION [DISTINCT | ALL]    … |

There are some rules that you need to follow in order to use the UNION operator:

* The number of columns appears in the corresponding [SELECT statements](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) must be equal.
* The columns appear in  the corresponding positions of each SELECT statement must have the same [data type](http://www.mysqltutorial.org/mysql-data-types.aspx) or at least convertible data type.

By default, the UNION operator eliminates duplicate rows from the result even if you don’t use[DISTINCT](http://www.mysqltutorial.org/mysql-distinct.aspx)operator explicitly. Therefore it is said that UNION clause is the shortcut of UNION DISTINCT.

If you use the UNION ALL explicitly, the duplicate rows, if available, remain in the result. The UNION ALL performs faster than the UNION DISTINCT.

MySQL UNION example

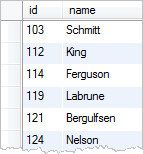
Let’s practice with an example of using MySQL UNION to get a better understanding.

Suppose you want to combine data from the  customers and employees tables into a single result set, you can UNION operator as the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT customerNumber id, contactLastname name  FROM customers  UNION  SELECT employeeNumber id,firstname name  FROM employees |

Here is the output:



MySQL UNION without Alias

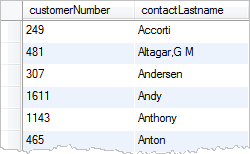
In the example above, we used the [column alias](http://www.mysqltutorial.org/mysql-alias/) for each column in the SELECT statements. What would be the output if we didn’t use the column alias? MySQL uses the names of columns in the firstSELECT statement as the labels for the output.

Let’s try the query that combines *customers* and *employees* information without using column alias:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | (SELECT customerNumber, contactLastname  FROM customers)  UNION  (SELECT employeeNumber, firstname  FROM employees)  ORDER BY contactLastname, customerNumber |

The result has customerNumber and contactLastname as the label*,*which are the names of columns in the first SELECT statement*.*



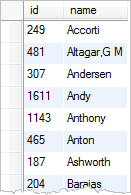
 MySQL UNION with ORDER BY

If you want to sort the results returned from the query using the UNION operator, you need to use[ORDER BY clause](http://www.mysqltutorial.org/mysql-order-by) in the last SQL SELECT statement. You can put each SELECT statement in the parentheses and use the ORDER BY clause as the last statement.

Let’s take a look at the following example:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | (SELECT customerNumber id,contactLastname name  FROM customers)  UNION  (SELECT employeeNumber id,firstname name  FROM employees)  ORDER BY name,id |



In the query above, first we combine id and name of both *employees* and *customers* into one result set using the UNION operator. Then we sort the result set by using the ORDER BY clause. Notice that we put the SELECT statements inside the parentheses and place the ORDER BY clause as the last statement.

If you place the ORDER BY clause in each SELECT statement, it will not affect the order of the rows in the final result produced by the UNION operator.

MySQL also provides you with alternative option to sort the result set based on column position usingORDER BY clause as the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | (SELECT customerNumber, contactLastname  FROM customers)  UNION  (SELECT employeeNumber,firstname  FROM employees)  ORDER BY 2, 1 |

In this tutorial, you have learned how to use MySQL UNION statement to combine data from multiple tables into a single result set.

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 matches rows in one table with rows in other tables and allows you to query rows that contain columns from both tables.

The MySQL INNER JOIN clause an optional part of the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx). It appears immediately after the FROM clause.

Before using MySQL INNER JOIN clause, you have to specify the following criteria:

* First, you have to specify the main table that appears in the FROM clause.
* Second, you need to specify the table that you want to join with the main table, which appears in the INNER JOIN clause. Theoretically, you can join a table with many tables. However, for better query performance, you should limit the number of tables to join.
* Third, you need to specify the join condition or join predicate. The join condition appears after the keyword ON of the INNER JOIN clause. The join condition is the rule for matching rows between the main table and the other tables.

The syntax of the MySQL INNER JOIN clause is as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT column\_list  FROM t1  INNER JOIN t2 ON join\_condition1  INNER JOIN t3 ON join\_condition2  ...  WHERE where\_conditions; |

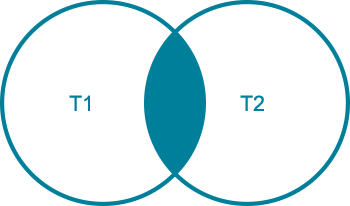
Let’s simplify the syntax above by assuming that we are joining two tables T1 and T2 using theINNER JOIN clause.

For each record in the T1 table, the MySQL INNER JOIN clause compares it with each record of theT2 table to check if both of them satisfy the join condition. When the join condition is matched, it will return that record that combine columns in either or both T1 and T2 tables.

Notice that the records on both T1 and T2 tables have to be matched based on the join condition. If no match found, the query will return an empty result set.

The logic is applied if we join more than 2 tables.

The following Venn diagram illustrates how the MySQL INNER JOIN clause works.

****

**MySQL INNER JOIN Venn Diagram**

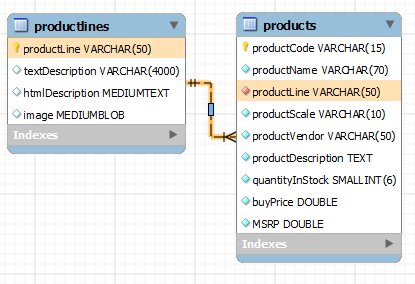
Avoid ambiguous column error in MySQL INNER JOIN

If you join multiple tables that have the same column name, you have to use table qualifier to refer to that column in the SELECT clause to avoid ambiguous column error. For example, if both   T1 and T2tables have the same column named C*;* in the SELECT clause, you have to refer to Ccolumn usingthe table qualifiers as T1.C or T2.C .

To save time typing the table qualifiers, you can use [table aliases](http://www.mysqltutorial.org/mysql-alias/) in the query. For example, you can give the verylongtablename table an alias T and refer to its columns using T.column instead of verylongtablename.column.

Examples of using MySQL INNER JOIN clause

Let’s take a look at two tables: products and productlines tables in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx).



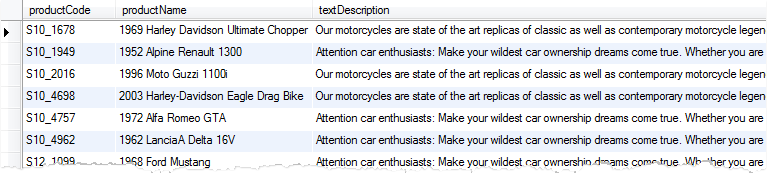
Now, if you want to get

* The*product code* and *product name* from the products table.
* The *text description* of product lines from the productlines table*.*

You need to select data from both tables and match rows by comparing the productline column from the products table with the productline column from the productlines table  as the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productCode,         productName,         textDescription  FROM products T1  INNER JOIN productlines T2 ON T1.productline = T2.productline; |

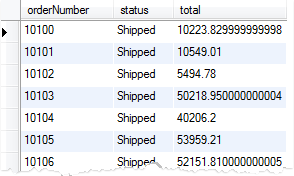


MySQL INNER JOIN with GROUP BY clause

We can get the order number, order status and total sales from the orders and orderdetailstables using the INNER JOIN clause with the [GROUP BY clause](http://www.mysqltutorial.org/mysql-group-by.aspx) as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT T1.orderNumber,         status,         SUM(quantityOrdered \* priceEach) total  FROM orders AS T1  INNER JOIN orderdetails AS T2 ON T1.orderNumber = T2.orderNumber  GROUP BY orderNumber |



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

MySQL LEFT JOIN

**Summary**: in this tutorial, you will learn about **MySQL LEFT JOIN** clause and how to apply it to query data from two or more database tables.

Introducing to MySQL LEFT JOIN

The MySQL LEFT JOIN clause allows you to query data from two or more database tables. TheLEFT JOIN clause is an optional part of the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), which appears after the FROMclause.

Let’s assume that we are going to query data from two tables T1 and T2. The following is the syntax of the LEFT JOIN clause that joins the two tables:

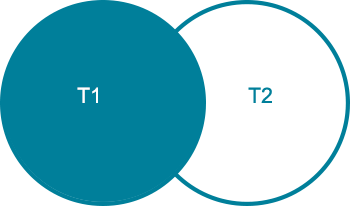


|  |  |
| --- | --- |
| 1  2  3 | SELECT T1.c1, T1.c2,... T2.c1,T2.c2  FROM T1  LEFT JOIN T2 ON T1.c1 = T2.c1... |

When we join the T1 table  to the T2 table using the LEFT JOIN clause, if a row from the left table T1 matches a row from the right table T2 based on the join condition ( T1.c1 = T2.c1), this row is included in the result set. In case the row in the left table does not match the row in the right table, the row in the left table is also selected and combined with a “fake” row from the right table. The fake row contains NULL values for all corresponding columns in theSELECT clause.

In other words, the LEFT JOIN clause allows you to select rows from the both left and right tables that match, plus all rows from the left table ( T1) even there is no match found for them in the right table ( T2).

The following Venn diagram helps you visualize how the MySQL LEFT JOIN clause works. The intersection between two circles are rows that match in both tables, and the remaining part of the left circle are rows in the T1 table that do not have matches in the T2 table. All rows in the left table are included in the result set.

****

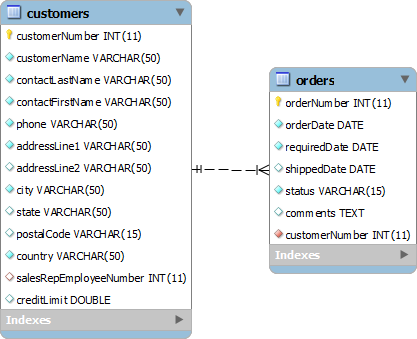
**MySQL LEFT JOIN – Venn Diagram**

Notice that the returned rows must also match the condition in the [WHERE](http://www.mysqltutorial.org/mysql-where/)and [HAVING](http://www.mysqltutorial.org/mysql-having.aspx)clauses if those clauses are available in the query.

MySQL LEFT JOIN Examples

MySQL LEFT JOIN clause – joining  2 tables example

Let’s take a look at the customers and orders tables.



In the database diagram above:

* Each order in the orders table must belong to a customer in the customers table.
* Each customer in the customers table can have zero or more orders in theorders table.

TO find all orders that belong to each customer, you can use the LEFT JOIN clause as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT c.customerNumber,         c.customerName,         orderNumber,         o.status  FROM customers c  LEFT JOIN orders o ON c.customerNumber = o.customerNumber |



The left table is the customers therefore all customers are included in the result set. However, there are rows in the result set that have customer data but no order data e.g. 168, 169, etc. The order data in these rows are NULL. It means that those customers do not have any order in the orders table.

If you replace the LEFT JOIN clause by the [INNER JOIN](http://www.mysqltutorial.org/mysql-inner-join.aspx) clause, you only get the customers who have orders in the orders table.

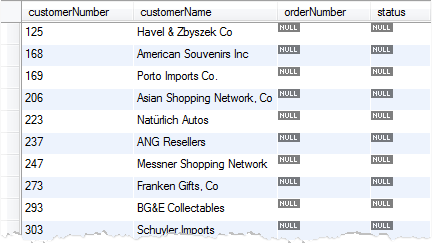
MySQL LEFT JOIN clause to find unmatched rows

The LEFT JOIN clause is very useful when you want to find the rows in the left table that do not match with the rows in the right table. To find the unmatched rows between two tables, you add a [WHERE clause](http://www.mysqltutorial.org/mysql-where/) to the SELECT statement to select only rows whose column values in the right table contains the NULL values.

For example, to find all customers who have not ordered any product, you can use the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SELECT c.customerNumber,         c.customerName,         orderNumber,         o.status  FROM customers c  LEFT JOIN orders o ON c.customerNumber = o.customerNumber  WHERE orderNumber IS NULL |



In this tutorial, we have explained the MySQL LEFT JOIN clause and shown you how to apply it to query data from multiple database tables.

MySQL Self Join

**Summary*:***in this tutorial, you will learn how to use **MySQL self join** that joins a table to itself using join statement.

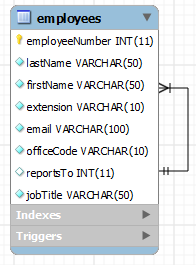
In the previous tutorial, you have learned how to join a table to the other tables using [INNER JOIN](http://www.mysqltutorial.org/mysql-inner-join.aspx), [LEFT JOIN](http://www.mysqltutorial.org/mysql-left-join.aspx) or RIGHT JOIN statement. However, there is a special case that you can join a table to itself, which is known as self join.

You use self join when you want to combine records with other records in the same table. To perform the self join operation, you must use a [table alias](http://www.mysqltutorial.org/mysql-alias/) to help MySQL distinguish the left table from the right table of the same table.

MySQL Self Join Examples

Let’s take a look at the employees table in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx).

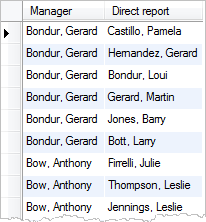
In the employees table, we store not only employees data but also organization structuredata. The reportsto column is used to determine the manager ID of an employee.



In order to get the whole organization structure, we can join the employees table to itself using the employeeNumber and reportsTo columns. The employees table has two roles: one is *Manager* and the other is *Direct Report.*



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT CONCAT(m.lastname,', ',m.firstname) AS 'Manager',         CONCAT(e.lastname,', ',e.firstname) AS 'Direct report'  FROM employees e  INNER JOIN employees m ON m.employeeNumber = e.reportsto  ORDER BY manager |

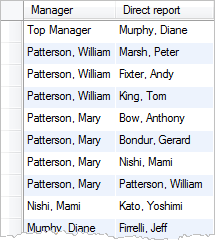


In the above example, we only see employees who have manager. However, we don’t see the top manager because his name is filtered out due to the INNER JOIN clause. The top manager is the employee who does not have manager or his manager no. is NULL.

Let’s change the INNER JOIN clause to the LEFT JOIN clause in the query above to include the top manager. We also need to use the [IFNULL](http://www.mysqltutorial.org/mysql-ifnull/)function to display the top manager if the manger’s name is NULL.



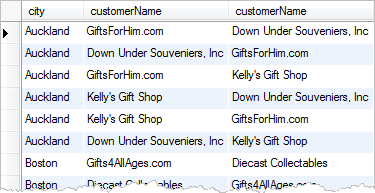
|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT IFNULL(CONCAT(m.lastname,', ',m.firstname),'Top Manager') AS 'Manager',         CONCAT(e.lastname,', ',e.firstname) AS 'Direct report'  FROM employees e  LEFT JOIN employees m ON m.employeeNumber = e.reportsto  ORDER BY manager DESC |



By using MySQL self join, we can display a list of customers who locate in the same city by joining the  customers table to itself.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT c1.city,         c1.customerName,         c2.customerName  FROM customers c1  INNER JOIN customers c2  ON c1.city = c2.city AND     c1.customername <> c2.customerName  ORDER BY c1.city |



We joined the customers table to itself with the following join conditions:

* c1.city = c2.city to make sure that both customers have the same city
* c.customerName <> c2.customerName to ensure that we don’t get the same customer.

In this tutorial, we have introduced you to MySQL self join that allows you to join a table to itself by using INNER JOIN or LEFT JOIN clauses.

MySQL GROUP BY

**Summary:** in this tutorial, you will learn how to use **MySQL GROUP BY** to group rows into subgroups based on columns or values returned by an expression.

Introducing to MySQL GROUP BY clause

The MySQL GROUP BY clause is used with the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) to group rows into subgroups by the one or more values of columns or expressions.

The MySQL GROUP BY clause is an optional part of the SELECT statement. It must appear after the FROM or WHERE clause. The MySQL GROUP BY clause consists of the GROUP BYkeyword followed by a list of comma-separated columns or expressions.

The following illustrates the MySQL GROUP BY clause syntax:



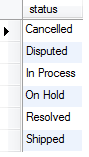
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT c1,c2,... cn, aggregate\_function(expression)  FROM table  WHERE where\_conditions  GROUP BY c1, c2, ... cn |

MySQL GROUP BY Examples

Let’s take a look at the orders table in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx). Suppose you want to group values of the *order status* into subgroups, you use the GROUP BY clause with the statuscolumn as the following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT status  FROM orders  GROUP BY status |



You can see that the GROUP BY clause returns unique occurrences of status values. It works like the [DISTINCT](http://www.mysqltutorial.org/mysql-distinct.aspx)operator as using in the following query:



|  |  |
| --- | --- |
| 1  2 | SELECT DISTINCT status  FROM orders |

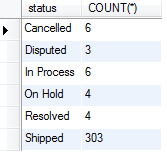
MySQL GROUP BY with aggregate functions

The [aggregate functions](http://www.mysqltutorial.org/mysql-aggregate-functions.aspx) allow you to perform calculation of a set of records and return a singlevalue. The most common aggregate functions are SUM, AVG, MAX, MIN and COUNT.

An aggregate functions is often used with the MySQL GROUP BY clause to perform calculation on each subgroup and return a single value for each subgroup. For example, if you want to know how many *orders* in each *status*, you can use the COUNT function with the GROUP BY clause as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT status, count(\*)  FROM orders  GROUP BY status |



MySQL GROUP BY vs. ANSI SQL GROUP BY

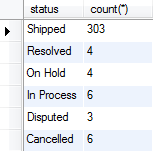
MySQL follows ANSI SQL. However, MySQL gives you more flexibility when using the GROUP BY clause:

* In ANSI SQL, you must list all columns that you use in the SELECT clause in the GROUP BY clause. MySQL does not have this restriction. MySQL allows you to have additional columns in the SELECT clause that are not specified 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 status and the number of orders in descending order, you can use theGROUP BY clause with DESC as the following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT status, count(\*)  FROM orders  GROUP BY status DESC; |



Notice that we use DESC in the GROUP BY clause to sort the status in descending order. You can specify ASC explicitly in the GROUP BY clause to sort the groups in ascending order.

In this tutorial, we have shown you how to use the MySQL GROUP BY clause to group rows into subgroups based on columns or values returned from an expression.

MySQL HAVING

**Summary***:*in this tutorial, you will learn how to use **MySQL HAVING** clause to specify a filter condition for groups of rows or aggregates.

Introducing MySQL HAVING clause

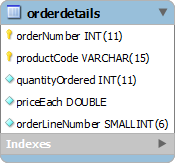
The MySQL HAVING clause is used in the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) to specify filter conditions forgroup of rows or aggregates.

The MySQL HAVING clause is often used with the [GROUP BY](http://www.mysqltutorial.org/mysql-group-by.aspx) clause. When using with the[GROUP BY](http://www.mysqltutorial.org/mysql-group-by.aspx) clause, you can apply a filter condition to the columns that appear in the GROUP BY clause. If the [GROUP BY](http://www.mysqltutorial.org/mysql-group-by.aspx)clause is omitted, the MySQL HAVING clause behaves like the[WHERE clause](http://www.mysqltutorial.org/mysql-where/). Notice that the MySQL HAVING clause applies the condition to each group ofrows, while the WHERE clause applies the condition to each individual row.

Examples of using MySQL HAVING clause

Let’s take a look at an example of using MySQL HAVING clause.

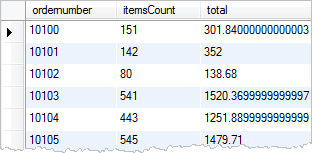
We will use the orderdetails table in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx) for the sake of demonstration.



We can use the [MySQL GROUP BY](http://www.mysqltutorial.org/mysql-group-by.aspx)clause to get order number, the number of items sold per order and total sales for each:



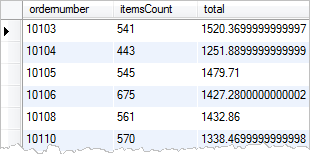
|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT ordernumber,         SUM(quantityOrdered) AS itemsCount,         SUM(priceeach) AS total  FROM orderdetails  GROUP BY ordernumber |



Now, we can find which order has total sales greater than $1000. We use the MySQL HAVING clause on the aggregate as follows:



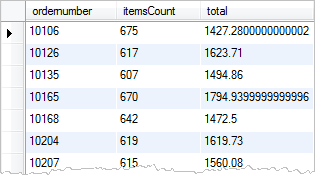
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT ordernumber,         SUM(quantityOrdered) AS itemsCount,         SUM(priceeach) AS total  FROM orderdetails  GROUP BY ordernumber  HAVING total > 1000 |



We can construct a complex condition in the MySQL HAVING clause using logical operators such as OR and AND. Suppose we want to find which order has total sales greater than $1000 and contains more than 600 items, we can use the following query:



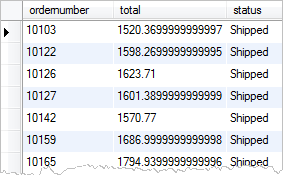
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT ordernumber,         sum(quantityOrdered) AS itemsCount,         sum(priceeach) AS total  FROM orderdetails  GROUP BY ordernumber  HAVING total > 1000 AND itemsCount > 600 |



Suppose we want to find all orders that has shipped and has total sales greater than $1500, we can join the orderdetails table with the orders table by using the [INNER JOIN](http://www.mysqltutorial.org/mysql-inner-join.aspx) clause, and apply condition on the status column and the total aggregate as the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT a.ordernumber,         SUM(priceeach) total,         status  FROM orderdetails a  INNER JOIN orders b ON b.ordernumber = a.ordernumber  GROUP BY ordernumber  HAVING b.status = 'Shipped' AND              total > 1500; |



The MySQL HAVING clause is only useful when we use it with the GROUP BY clause to generate the output of the high-level reports. For example, we can use the MySQL HAVING clause to answer some kinds of queries like give me all the orders in this month, this quarter and this year that have total sales greater than 10K.

In this tutorial, you have learned how to use the MySQL HAVING clause together with the GROUP BY to specify filter condition on groups of records or aggregates.

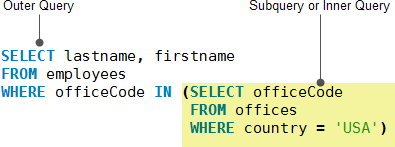
MySQL Subquery

**Summary**: in this tutorial, we will show you how to use the **MySQL subquery** to write complex queries and explain the correlated subquery concept.

A MySQL subquery is a query that is nested inside another query such as [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx),[UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx" \o "MySQL Update)or [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx). A MySQL subquery is also can be nested inside another subquery. A MySQL subquery is also called an inner query, while the query that contains the subquery is called an outer query.

Let’s take a look at the following subquery that returns employees who locate in the offices in theUSA.

* The subquery returns all *offices codes* of the offices that locate in the USA.
* The outer query selects the last name and first name of employees whose office code is in the result set returned from the subquery.



You can use a subquery anywhere an expression can be used. A subquery also must be enclosed in parentheses.

MySQL subquery within a WHERE clause

MySQL subquery with comparison operators

If a subquery returns a single value, you can use comparison operators to compare it with the expression in the [WHERE clause](http://www.mysqltutorial.org/mysql-where/). For example, the following query returns the customer who has the maximum payment.



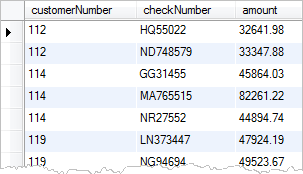
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT customerNumber,         checkNumber,         amount  FROM payments  WHERE amount = (      SELECT MAX(amount)          FROM payments  ) |

mysql subquery with equal operator

You can also use other comparison operators such as greater than (>), less than(<), etc. For example, you can find customer whose payment is greater than the average payment. A subquery is used to calculate the average payment by using the AVG [aggregate function](http://www.mysqltutorial.org/mysql-aggregate-functions.aspx). The outer query selects payments that are greater than the average payment returned from the subquery.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT customerNumber,         checkNumber,         amount  FROM payments  WHERE amount > (      SELECT AVG(amount)      FROM payments  ) |

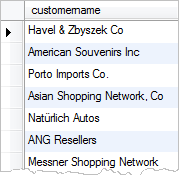


MySQL subquery with IN and NOT IN operators

If a subquery returns more than one value, you can use other operators such as [IN](http://www.mysqltutorial.org/sql-in.aspx)or [NOT IN](http://www.mysqltutorial.org/sql-in.aspx)operator in the WHERE clause. For example, you can use a subquery with NOT IN operator to find customer who has not ordered any product as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT customername  FROM customers  WHERE customerNumber NOT IN(      SELECT DISTINCT customernumber      FROM orders  ) |



MySQL subquery with EXISTS and NOT EXISTS

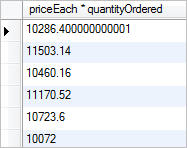
When a subquery is used with EXISTS or NOT EXISTS operator, a subquery returns a Boolean value of TRUE or FALSE. The subquery acts as an existence check.

In the following example, we select a list of customers who have at least one order with total sales greater than 10K.

First, we build a query that checks if there is at least one order with total sales greater than 10K:



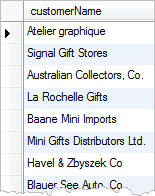
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT priceEach \* quantityOrdered  FROM orderdetails  WHERE priceEach \* quantityOrdered > 10000  GROUP BY orderNumber |



The query returns 6 records so that when we use it as a subquery, it will return TRUE; therefore the whole query will return all customers:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT customerName  FROM customers  WHERE EXISTS (      SELECT priceEach \* quantityOrdered      FROM orderdetails      WHERE priceEach \* quantityOrdered > 10000      GROUP BY orderNumber  ) |



If you replace the EXISTS by NOT EXIST in the query, it will not return any record at all.

MySQL subquery in FROM clause

When you use a subquery in the FROM clause, the result set returned from a subquery is used as a table. This table is referred to as a *derived table* or *materialized subquery*.

The following subquery finds the maximum, minimum and average number of items in sale orders:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT max(items),         min(items),         floor(avg(items))  FROM  (SELECT orderNumber,      count(orderNumber) AS items  FROM orderdetails  GROUP BY orderNumber) AS lineitems |

Notice that the subquery returns the following result set that is used as a derived table for the outer query.

mysql subquery from clause example

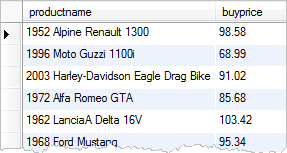
MySQL correlated subquery

In the previous examples, we see the subquery itself is independent. It means that you can execute the subquery as a normal query. However a correlated subquery is a subquery that uses the information from the outer query, or we can say that a correlated subquery depends on the outer query. A correlated subquery is evaluated once for each row in the outer query.

In the following correlated subquery, we select products whose buy price is greater than the average buy price of all products for a particular *product line*.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SELECT productname,         buyprice  FROM products AS p1  WHERE buyprice > (      SELECT AVG(buyprice)      FROM products          WHERE productline = p1.productline) |



The inner query executes for every product line because the product line is changed for every row. Hence the average buy price will also change.

In this tutorial, we have shown you how to use MySQL subquery and correlated subquery to writemore complex queries.

MySQL Replace

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

Introduction to MySQL REPLACE statement

The MySQL REPLACE statement is a MySQL extension to the SQL standard. The MySQLREPLACE statement works like the [INSERT statement](http://www.mysqltutorial.org/mysql-insert-statement.aspx) with the additional rules:

* If the record which you want to insert does not exist, the MySQL REPLACE inserts a new record.
* If the record which you want to insert already exists, MySQL REPLACE deletes the old record first and then insert a new record.

In order to use MySQL REPLACE statement, you need to have at least both INSERT andDELETE privileges.

Please don’t confuse the REPLACE statement with the [REPLACE](http://www.mysqltutorial.org/mysql-string-replace-function.aspx) string function.

MySQL REPLACE statements

MySQL REPLACE INTO statement

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



|  |  |
| --- | --- |
| 1  2 | REPLACE INTO table\_name(column\_name1,column\_name2,…)  VALUES(value1,value2,…) |

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



|  |  |
| --- | --- |
| 1  2 | REPLACE INTO offices(officecode,city)  VALUES(8,'San Jose') |

Notice that the default values of of the columns that does not appear in the REPLACEstatement will be inserted to the corresponding columns.

If you want to update the office that we have inserted with the new city San Mateo*,* you can use the REPLACE statement as follows:



|  |  |
| --- | --- |
| 1  2 | REPLACE INTO offices(officecode,city)  VALUES(8,'San Mateo') |

Two rows affected by the query above because the existing record was deleted and the new one was inserted.

MySQL REPLACE acts like UPDATE statement

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



|  |  |
| --- | --- |
| 1  2  3 | REPLACE INTO table\_name  SET column\_name1 = value1 AND      column2 = value2 |

Notice that there is no [WHERE clause](http://www.mysqltutorial.org/mysql-where/) in the REPLACE statement. For example, if you want to update the office in San Mateocity with officecode value 8, you use the REPLACEstatement as follows:



|  |  |
| --- | --- |
| 1  2  3 | REPLACE INTO offices  SET officecode = 8 and      city = 'Santa Cruz' |

MySQL REPLACE INTO with SELECT statement

The third form of  REPLACE is similar to [INSERT INTO SELECT](http://www.mysqltutorial.org/mysql-insert-statement.aspx) statement:



|  |  |
| --- | --- |
| 1  2  3  4 | REPLACE INTO table\_name1(column\_name1,column\_name2,…)  SELECT column\_name1, column\_name2…  FROM table\_name2  WHERE where\_condition |

Suppose if you want to copy the office with officecode value 1, you use the REPLACE INTO SELECT statement as the following query:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | 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 |

MySQL REPLACE usages

There are several important points you need to know when you use the REPLACE statement:

* If you are developing an application that potentially supports not only MySQL database, try to avoid using the REPLACE statement because other database management systemsmay not support the REPLACE statement. Instead, you can use the combination of the[INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx)and [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx) statements.
* If you are using the REPLACE statement in the table that has [triggers](http://www.mysqltutorial.org/mysql-triggers.aspx)and if the deletion of duplicate key happens, the triggers will be fired in the following sequence: BEFORE INSERT, BEFORE DELETE, AFTER DELETE, AFTER INSERT.
* You should use the [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx) statement in case you want to update data because it performs faster than the REPLACE statement.

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

MySQL Transaction

**Summary**: in this tutorial, you will learn about **MySQL transaction**and how to use MySQL COMMIT statement and MySQL ROLLBACK statement to manage transactions in MySQL.

Introducing to MySQL Transaction

To understand what a transaction in MySQL is, let’s take a look at an example of adding a new sale order in our [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx). The steps of adding a sale order are as described as follows:

* 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? If the step of adding order items intoorderdetails 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 a set of MySQL operations to ensure that the database never contains the result of partial operations. In a 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 in the example above.

To start a transaction you use the START TRANSACTION statement. To undo MySQL statements you use the ROLLBACK statement. Notice that there are several SQL statements you cannot use ROLLBACK such as:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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:



|  |  |
| --- | --- |
| 1 | SET autocommit = 0; |

MySQL transaction example

In order to use MySQL transaction, you first have to break your MySQL 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 statement.
* 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.

The following  is the script that performs the above steps:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | -- 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 and ROLLBACK to manage transactions in MySQL to protect data integrity.

MySQL Prepared Statement

**Summary**: in this tutorial, you will learn how to use **MySQL prepared statement** to make your queries execute faster and more secure.

Introduction to MySQL Prepared Statement

Prior MySQL version 4.1, the query is sent to the MySQL server in the textual format. In turn, MySQL returns the data to the client using textual protocol. MySQL has to parse the query fully and coverts the result set into a string before returning it to the client.

The textual protocol has serious performance implication. To resolve this problem, MySQL added a new feature called prepared statement since version 4.1.

The prepared statement takes advantage of [client/server binary protocol](https://dev.mysql.com/doc/internals/en/client-server-protocol.html). It passes query that contains placeholders (?) to the MySQL server as the following example:



|  |  |
| --- | --- |
| 1  2  3 | SELECT \*  FROM products  WHERE productCode = ? |

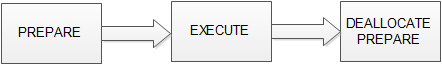
When MySQL executes this query with different productcode values, it does not have to parse the query fully. As a result, this helps MySQL execute the query faster, especially when MySQL executes the query multiple times. Because the prepared statement uses placeholders (?), this helps avoid many variants of SQL injection hence make your application more secure.

MySQL prepared statement usage

In order to use MySQL prepared statement, you need to use other three MySQL statements as follows:

* PREPARE – Prepares statement for execution.
* EXECUTE – Executes a prepared statement preparing by a PREPARE statement.
* DEALLOCATE PREPARE – Releases a prepared statement.

The following diagram illustrates how to use the prepared statement:



MySQL prepared statement example

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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | PREPARE stmt1 FROM 'SELECT productCode, productName                      FROM products                      WHERE productCode = ?';    SET @pc = 'S10\_1678';  EXECUTE stmt1 USING @pc;    DEALLOCATE PREPARE stmt1; |

First we used the PREPARE statement to prepare a statement for execution. We used the [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx)statement to query product data from the  products table based on a specified product code. We used question mark (?) as a placeholder for the product code.

Next, we declared a product code variable  @pc and set it values to S10\_1678.

Then, we used the EXECUTE statement to execute the prepared statement with product code variable @pc.

Finally, we used the  DEALLOCATE PREPARE to release the prepared statement.

In this tutorial, we have shown you how to use MySQL prepared statement to execute a query with placeholders to improve the speed of the query and make your query more secure.

MySQL Character Set

**Summary**: in this tutorial, you will learn about **MySQL character set**. After the tutorial, you will know how to get all character sets in MySQL, how to convert strings between character sets and how to configure proper character sets for client connections.

Introduction to MySQL character set

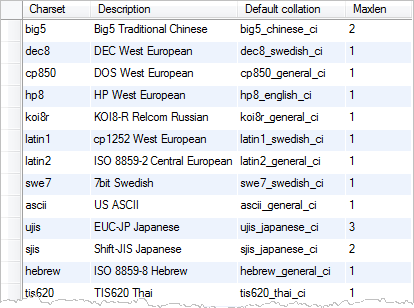
A MySQL character set is a set of characters that are legal in a string. For example, we have an alphabet with letters from a to z. We assign each letter a number a = 1, b = 2, etc. The letter  a is a symbol, and the number 1 that associates with the letter a is the encoding. The combination of all letters from a to z and their corresponding encodings is a character set.

Each character set has one or more collations that define a set of rules for comparing characters within the character set. Check it out the [MySQL collation](http://www.mysqltutorial.org/mysql-collation/) tutorial to learn about the collations in MySQL.

MySQL supports various character sets that allow you to store almost every character in a string. To get all available character sets in MySQL database server, you use the SHOW CHARACTER SET statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW CHARACTER SET; |



The default character set in MySQL is latin1. If you want to store characters from multiple languages in a single column, you can use Unicode character sets, which is utf8 or ucs2.

The values in the Maxlen column specify the number of bytes that a character in a character set holds. Some character sets contain single-byte characters e.g., latin1, latin2, cp850, etc., whereas other character sets contain multi-byte characters.

MySQL provides the LENGTH function to get a length of a string in bytes, and the CHAR\_LENGTH function to get the length of a string in characters. If a string contains multi-bytes character, the result of the LENGTH function is greater than the result of the CHAR\_LENGTH()function. See the following example:



|  |  |
| --- | --- |
| 1  2 | SET @str = CONVERT('MySQL Character Set' USING ucs2);  SELECT LENGTH(@str), CHAR\_LENGTH(@str); |

mysql convert character set

The CONVERT function converts a string into a specific character set. In this example, it converts the character set of the MySQL Character Set string into ucs2. Because ucs2character set contains 2-byte characters, therefore the length of the @str string in bytes is greater than its length in characters.

Notice that some character sets contain multi-byte characters,  but their strings may contain only single-byte characters e.g., utf8 as shown in the following statements:



|  |  |
| --- | --- |
| 1  2 | SET @str = CONVERT('MySQL Character Set' USING utf8);  SELECT LENGTH(@str), CHAR\_LENGTH(@str); |

single-byte character set

However, if a utf8 string contains special character e.g., ü in pingüino string; its length in bytes is different, see the following example:



|  |  |
| --- | --- |
| 1  2 | SET @str = CONVERT('pingüino' USING utf8);  SELECT LENGTH(@str), CHAR\_LENGTH(@str); |

unicode character set

Converting between different character sets

MySQL provides two functions that allow you to convert strings between different character sets:CONVERT and CAST. We have used the CONVERT function several times in the above examples.

The syntax of the CONVERT function is as follows:



|  |  |
| --- | --- |
| 1 | CONVERT(expression USING character\_set\_name) |

The CAST function is similar to the CONVERT function. It converts a string to a different character set:



|  |  |
| --- | --- |
| 1 | CAST(string AS character\_type CHARACTER SET character\_set\_name) |

Take a look at the following example of using the CAST function:



|  |  |
| --- | --- |
| 1 | SELECT CAST(\_latin1'MySQL character set' AS CHAR CHARACTER SET utf8); |

Setting character sets for client connections

When an application exchanges data with a MySQL database server, the default character set islatin1. However, if the database stores Unicode strings in utf8 character set, using latin1character set in the application would not be sufficient. Therefore, the application needs to specify a proper character set when it connects to MySQL database server.

To configure a character set for a client connection, you can do one of the following ways:

* Issue the SET NAME statement after the client connected to the MySQL database server. For example, to set a Unicode character set utf8, you use the following statement:



|  |  |
| --- | --- |
| 1 | SET NAMES 'utf8'; |

* If the application supports the --default-character-set option, you can use it to set the character set. For example, *mysql* client tool supports --default-character-set and you can set it up in the configuration file as follows:



|  |  |
| --- | --- |
| 1  2 | [mysql]  default-character-set=utf8 |

* Some MySQL connectors allow you to set character set, for example, if you use PHP PDO, you can set the character set in the data source name as follows:



|  |  |
| --- | --- |
| 1 | $dsn ="mysql:host=$host;dbname=$db;charset=utf8"; |

Regardless of which way you use, make sure that the character set used by the applicationmatches with the character set stored in the MySQL database server.

In this tutorial, you have learned about MySQL character set, how to convert strings between character sets and how to configure proper character sets for client connections.

MySQL Collation

**Summary**: in this tutorial, you will learn  about **MySQL collation** and how to set character sets and collations for the MySQL server, database, table and column.

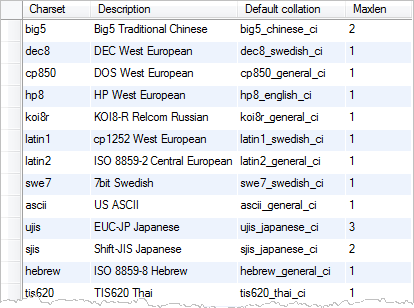
Introduction to MySQL collation

A MySQL collation is a set of rules used to compare characters in a particular [character set](http://www.mysqltutorial.org/mysql-character-set/). Each character set in MySQL can have more than one collation, and has at least one default collation. Two character sets cannot have the same collation.

MySQL provides you with the SHOW CHARACTER SET that allows you to get the default collations of character sets as follows:



|  |  |
| --- | --- |
| 1 | SHOW CHARACTER SET; |



The values of the default collation column specify the default collations for the character sets.

By convention, a collation for a character set begins with the character set name and ends with\_ci (case insensitive) \_cs (case sensitive) or \_bin (binary).

To get all collations for a given character set, you use the SHOW COLLATION statement as follows:

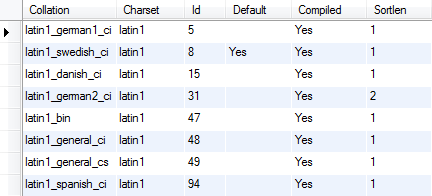


|  |  |
| --- | --- |
| 1 | SHOW COLLATION LIKE 'character\_set\_name%'; |

For example, to get all collations for the latin1 character set, you use the following statement:



|  |  |
| --- | --- |
| 1 | SHOW COLLATION LIKE 'latin1%'; |

****

**MySQL Collations for latin1 Character Set**

As mentioned above, each character set has at a default collation e.g., latin1\_swedish\_ciis the default collation for the latin1 character set.

Setting character sets and collations

MySQL allows you to specify character sets and collations at four levels: server, database, table, and column.

Setting character sets and collations at server Level

Notice MySQL uses latin1 as the default character set therefore its default collation islatin1\_swedish\_ci. You can change these settings at server startup.

If you specify only a character set at server startup, MySQL will use the default collation of the character set. If you specify both a character set and a collation explicitly, MySQL will use the character set and collation for all databases created in the database server.

The following statement sets the utf8 character set and utf8\_unicode\_cs collation for the servervia command line:



|  |  |
| --- | --- |
| 1 | >mysqld --character-set-server=utf8 --collation-server=utf8\_unicode\_ci |

Setting character sets and collations at database level

When you create a database, if you do not specify its character set and collation, MySQL will use the default character set and collation of the server for the database.

You can override the default settings at database level by using [CREATE DATABASE](http://www.mysqltutorial.org/mysql-create-table/) or [ALTER DATABASE](http://www.mysqltutorial.org/mysql-alter-table.aspx) statement as follows:



|  |  |
| --- | --- |
| 1  2  3 | CREATE DATABASE database\_name  CHARACTER SET character\_set\_name  COLLATE collation\_name |



|  |  |
| --- | --- |
| 1  2  3 | ALTER  DATABASE database\_name  CHARACTER SET character\_set\_name  COLLATE collation\_name |

MySQL uses the character set and collation at database level for all tables created within the database.

Setting character sets and collations at table level

A database may contain tables with character sets and collations that are different from the default database’s character set and collation.

You can specify the default character set and collation for a table when you create the table by using the CREATE TABLE statement or when you alter the table’s structure by using the ALTER TABLE statement.



|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TABLE table\_name(  )  CHARACTER SET character\_set\_name  COLLATE collation\_name |



|  |  |
| --- | --- |
| 1  2  3  4 | ALTER TABLE table\_name(  )  CHARACTER SET character\_set\_name  COLLATE collation\_name |

Setting character set and collation at column level

A column of type CHAR, VARCHAR or TEXT can have its own character set and collation that is different from the default character set and collation of the table.

You can specify a character set and a collation for the column in the column’s definition of either CREATE TABLE or ALTER TABLE statement as follows:



|  |  |
| --- | --- |
| 1  2  3 | column\_name [CHAR | VARCHAR | TEXT] (length)  CHARACTER SET character\_set\_name  COLLATE collation\_name |

These are the rules for setting the character set and collation:

* If you specify both a character set and a collation explicitly, the character set and collation are used.
* If you specify a character set and omit the collation, the default collation of the character set is used.
* If you specify a collation without a character set, the character set associated with the collation is used.
* If you omit both character set and collation, the default character set and collation are used.

Let’s take a look at some examples of setting the character sets and collations.

Examples of setting character sets and collations

First, we create a new database with utf8 as the character set and utf8\_unicode\_ci as the default collation:



|  |  |
| --- | --- |
| 1  2  3 | CREATE DATABASE mydbdemo  CHARACTER SET utf8  COLLATE utf8\_unicode\_ci; |

Because we specify the character set and collation for the mydbdemo database explicitly, themydbdemo does not take the default character set and collation at server level.

Second, we create a new table named t1 in the mydbdemo database:



|  |  |
| --- | --- |
| 1  2  3  4 | USE mydbdemo;  CREATE TABLE t1(  c1 char(25)  ); |

We did not specify the character set and collation for the t1 table; MySQL will check the database level to determine the character set and collation for the t1 table. In this case, thet1 table has utf8 as the default character set and utf8\_unicode\_ci as the default collation.

Third, for the t1 table, we change its character set to latin1 and its collation tolatin1\_german1\_ci:



|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE t1  CHARACTER SET latin1  COLLATE latin1\_german1\_ci; |

The c1 column in the t1 table has latin1 as the character set andlatin1\_german1\_ci as the collation.

Fourth, let’s change the character set of the c1 column to latin1:



|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE t2  MODIFY c1 VARCHAR(25)  CHARACTER SET latin1; |

Now, the c1 column has latin2 character set, but what about its collation? Is it inheriting the latin1\_german1\_ci collation from the table’s collation? No, because the default collation of the latin1 character set is latin1\_swedish\_ci, the c1 column haslatin1\_swedish\_ci collation.

In this tutorial, you have learned about MySQL collation and how to specify character sets and collations for MySQL serer, databases, tables and columns.

PART II - MySQL Stored Procedure

Introduction to MySQL Stored Procedures

**Summary**: in this tutorial, you will learn about **MySQL stored procedures**, their advantages and disadvantages.

Definition of stored procedures

A stored procedure is a segment of declarative SQL statements stored inside the database catalog. A stored procedure can be invoked by [triggers](http://www.mysqltutorial.org/mysql-triggers.aspx), other stored procedures or applications such as Java, C#, PHP, etc.

A stored procedure that calls itself is known as a recursive stored procedure. Most database management system supports recursive stored procedures. However MySQL does not support it very well. You should check your version of MySQL database before implementing recursive stored procedures in MySQL.

Stored Procedures in MySQL

MySQL is known as the most popular open source RDBMS which is widely used by both community and enterprise. However, during the first decade of its existence, it did not support stored procedures, stored functions, [trigger](http://www.mysqltutorial.org/mysql-triggers.aspx)s and events. Since MySQL version 5.0, those features were added to MySQL database engine to make it more flexible and powerful.

MySQL stored procedures advantages

* Typically stored procedures help increase the performance of the applications. Once created, stored procedures are compiled and stored in the database. However MySQL implements the stored procedures slightly different. MySQL stored procedures are compiled on demand. After compiling a stored procedure, MySQL puts it to a cache. And MySQL maintains its own stored procedure cache for every single connection. If an application uses a stored procedure multiple times in a single connection, the compiled version is used, otherwise the stored procedure works like a query.
* Stored procedures helps reduce the traffic between application and database server because instead of sending multiple lengthy SQL statements, the application has to send only name and parameters of the stored procedure.
* Stored procedures are reusable and transparent to any applications. Stored procedures expose the database interface to all applications so that developers don’t have to develop functions that are already supported in stored procedures.
* Stored procedures are secure. Database administrator can grant appropriate permissions to applications that access stored procedures in the database without giving any permission on the underlying database tables.

Besides those advantages, stored procedures have their own disadvantages, which you should be aware of before using the store procedures.

MySQL stored procedures disadvantages

* If you use a lot of stored procedures, the memory usage of every connection that is using those stored procedures will increase substantially. In addition, if you overuse a large number of logical operations inside store procedures, the CPU usage will also increase because database server is not well-designed for logical operations.
* A constructs of stored procedures make it more difficult to develop stored procedures that have complicated business logic.
* It is difficult to debug stored procedures. Only few database management systems allow you to debug stored procedures. Unfortunately, MySQL does not provide facilities for debugging stored procedures.
* It is not easy to develop and maintain stored procedures. Developing and maintaining stored procedures are often required specialized skill set that not all application developers possess. This may lead to problems in both application development and maintenance phases.

MySQL stored procedures have their own advantages and disadvantages. When you develop applications, you should decide whether you should or should not use stored procedure based on the business requirements.

In the following tutorials, we will show you how to leverage MySQL stored procedures in your database programming tasks with many practical examples.

Getting Started with MySQL Stored Procedures

**Summary***:*in this tutorial, we will show you step by step how to develop the first **MySQL stored procedure**by using CREATE PROCEDURE statement. In addition, we will show you how to call the stored procedures from SQL statements.

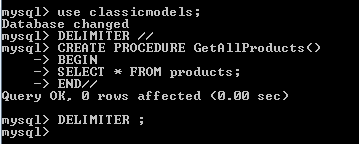
Writing the first MySQL stored procedure

We are going to develop a simple [stored procedure](http://www.mysqltutorial.org/introduction-to-sql-stored-procedures.aspx) named GetAllProducts() to help you get familiar with the syntax. The GetAllProducts() stored procedure selects all products from the productstable.

Launch the mysql client tool and type the following commands:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | DELIMITER //  CREATE PROCEDURE GetAllProducts()     BEGIN     SELECT \*  FROM products;     END //  DELIMITER ; |

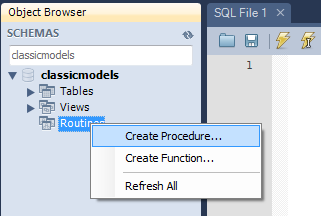
[](http://www.mysqltutorial.org/wp-content/uploads/2009/12/mysql-stored-procedure-command-line.png?44591f)

Let’s examine the stored procedure in greater detail:

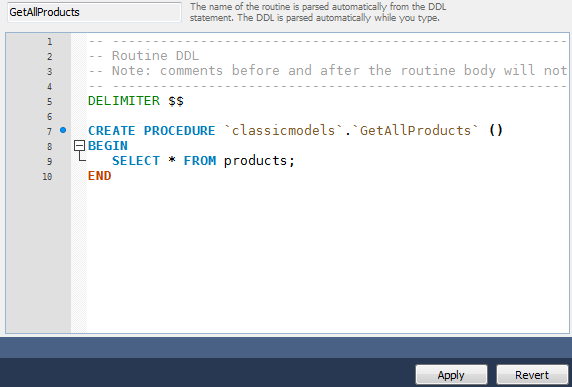
* The first command is DELIMITER //, which is not related to the stored procedure syntax. The DELIMITER statement changes the standard delimiter which is semicolon ( ;) to another. In this case, the delimiter is changed from the semicolon( ;) to double-slashes//. Why do we have to change the delimiter? Because we want to pass the  stored procedure to the server as a whole rather than letting mysql tool to interpret each statement at a time.  Following the END keyword, we use the delimiter // to indicate the end of the stored procedure. The last command ( DELIMITER;*)* changes the delimiter back to the standard one.
* We use the CREATE PROCEDURE statement to create a new stored procedure. We specify the name of stored procedure after the CREATE PROCEDURE statement. In this case, the name of the stored procedure is GetAllProducts*.*We put the parentheses after the name of the stored procedure*.*
* The section between BEGIN and END is called the body of the stored procedure. You put the declarative SQL statements in the body to handle business logic. In this stored procedure, we use a simple [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx)statement to query data from the products table.

It’s kind of tedious to write the stored procedure in mysql client tool, especially when thestored procedure is complex. Most of the GUI tools for MySQL allow you to create new stored procedures via an intuitive interface.

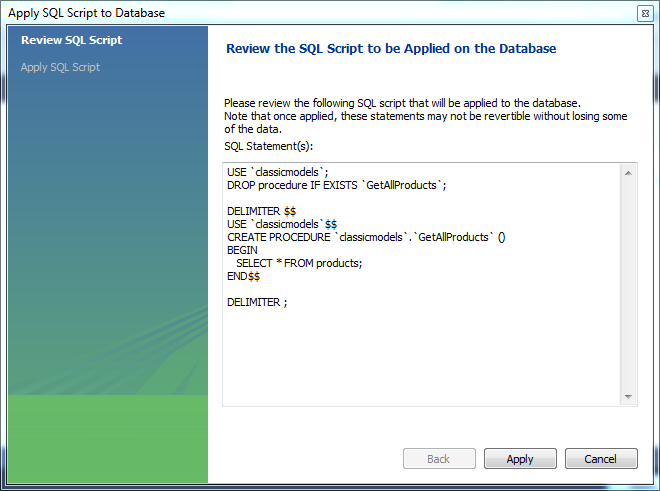
For example, in MySQL Workbench, you can create a new stored procedure as follows:

****

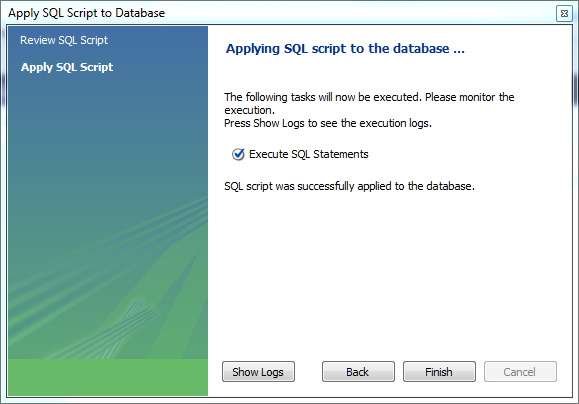
**Right mouse click on the Routines and choose “Create Procedure…”**

****

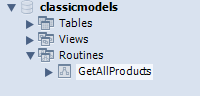
**Enter the stored procedure code and click the Apply button**

**[](http://www.mysqltutorial.org/wp-content/uploads/2009/12/create-mysql-stored-procedure-mysql-workbench-step-3.png?44591f)**

**You can review the code before MySQL stores it in the database. Click Apply button if everything is good.**

****

**MySQL compiles and puts the stored procedure in the database catalog; click the Finish button.**

****

**You can see a new stored procedure created under Routines of the *classicmodels*database**

We have created a new stored procedure. Now, it’s time to learn how to use it.

Calling stored procedures

In order to call a stored procedure, you use the following SQL command:



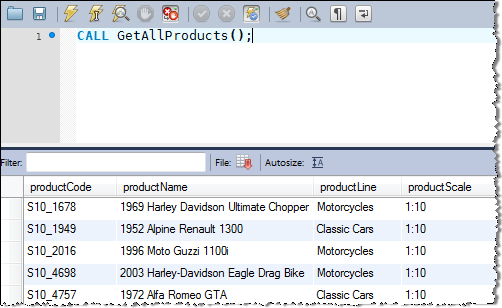
|  |  |
| --- | --- |
| 1 | CALL STORED\_PROCEDURE\_NAME() |

You use the CALL statement to call a stored procedure e.g., to call the GetAllProductsstored procedure, you use the following statement:



|  |  |
| --- | --- |
| 1 | CALL GetAllProducts(); |

If you execute the statement above, you will get all products in the products table.

[](http://www.mysqltutorial.org/wp-content/uploads/2009/12/mysql-stored-procedure.png?44591f)

In this tutorial, you have learned how to write a simple stored procedure by using the CREATE PROCEDURE statement and call it by using the CALL statement.

MySQL Stored Procedure Variables

**Summary**: in this tutorial, you will learn about variables in stored procedure, how to declare, and use variables. In addition, you will learn about the scopes of variables.

A variable is a named data object whose value can change during the stored procedure execution. We typically use the variables in [stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) to hold the immediate results. These variables are local to the stored procedure.

You must declare a variable before you can use it.

Declaring variables

To declare a variable inside a stored procedure, you use the DECLARE statement as follows:



|  |  |
| --- | --- |
| 1 | DECLARE variable\_name datatype(size) DEFAULT default\_value; |

Let’s examine the statement above in more detail:

* First, you specify the variable name after the  DECLARE keyword. The variable name must follow the naming rules of MySQL table column names.
* Second, you specify the data type of the variable and its size. A variable can have any[MySQL data types](http://www.mysqltutorial.org/mysql-data-types.aspx) such as INT, VARCHAR, DATETIME, etc.
* Third, when you declare a variable, its initial value is NULL. You can assign the variable a default value by using DEFAULT keyword.

For example, we can declare a variable named  total\_sale with the data type INT and default value 0 as follows:



|  |  |
| --- | --- |
| 1 | DECLARE total\_sale INT DEFAULT 0 |

MySQL allows you to declare two or more variables that share the same data type using a singleDECLARE statement as following:



|  |  |
| --- | --- |
| 1 | DECLARE x, y INT DEFAULT 0 |

We declared two INT variables  x and  y , and set their default values to zero.

Assigning variables

Once you declared a variable, you can start using it. To assign a variable another value, you use the SET statement, for example:



|  |  |
| --- | --- |
| 1  2 | DECLARE total\_count INT DEFAULT 0  SET total\_count = 10; |

The value of the total\_count variable is 10 after the assignment.

Besides the SET statement, you can use SELECT INTO statement to assign the result of a query to a variable. Notice that the query must return a scalar value.



|  |  |
| --- | --- |
| 1  2  3  4 | DECLARE total\_products INT DEFAULT 0    SELECT COUNT(\*) INTO total\_products  FROM products |

In the example above:

* First, we declare a variable named total\_products and initialize its value to 0.
* Then, we used the SELECT INTO statement to assign the total\_products variable the number of products that we selected from the products from the products table.

Variables scope

A variable has its own scope, which defines its life time. If you declare a variable inside a stored procedure, it will be out of scope when the END statement of stored procedure reached.

If you declare a variable inside BEGIN END block, it will be out of scope if the END is reached. You can declare two or more variables with the same name in different scopes because a variable is only effective in its own scope. However, declaring variables with the same name in different scopes is not good programming practice.

A variable that begins with the @ sign at the beginning is session variable. It is available and accessible until the session ends.

In this tutorial, we have shown you how to declare a variable inside stored procedures and discussed about the variable scopes.

MySQL Stored Procedure Parameters

**Summary**: in this tutorial, we will show you how to write [MySQL stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) with parameters. We will also give you a couple of stored procedure examples to help you understand how to use different kinds of stored procedure parameters.

Introduction to MySQL stored procedure parameters

Almost stored procedures that you develop require parameters. The parameters make the stored procedure more flexible and useful. In MySQL, a parameter has one of three modes IN, OUTor INOUT.

* IN – is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the INparameter.
* OUT – the value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedurecannot access the initial value of the OUT parameter when it starts.
* INOUT – an INOUT parameter is the combination of IN parameter and OUTparameter. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter and pass the new value back to the calling program.

The syntax of defining a parameter in the stored procedures is as follows:



|  |  |
| --- | --- |
| 1 | MODE param\_name param\_type(param\_size) |

* The MODE could be IN, OUT or INOUT , depending on the purpose of parameter in the stored procedure.
* The param\_name is the name of the parameter. The name of parameter must follow the naming rules of the column name in MySQL.
* Followed the parameter name is its data type and size. Like a variable, the data type of the parameter can by any [MySQL data type](http://www.mysqltutorial.org/mysql-data-types.aspx).

Each parameter is separated by a comma ( ;) if the stored procedure has more than one parameter.

Let’s practice with some examples to get a better understanding.

MySQL stored procedure parameter examples

IN parameter example

The following example illustrates how to use the IN parameter in the GetOfficeByCountrystored procedure that selects offices located in a specified country.



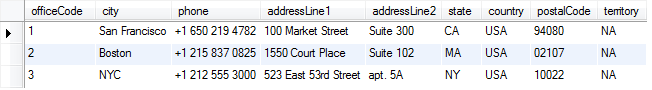
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | DELIMITER //  CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))      BEGIN          SELECT \*          FROM offices          WHERE country = countryName;      END //  DELIMITER ; |

The countryName is the IN parameter of the stored procedure. Inside the stored procedure, we select all offices that locate in the country specified by the countryName parameter*.*

Suppose, you want to get all offices in the USA, you just need to pass a value (USA) to thestored procedure as follows:



|  |  |
| --- | --- |
| 1 | CALL GetOfficeByCountry('USA') |



To get all offices in France, you pass the France literal string to the GetOfficeByCountrystored procedure as follows:



|  |  |
| --- | --- |
| 1 | CALL GetOfficeByCountry('France') |

[IN parameter offices in France](http://www.mysqltutorial.org/wp-content/uploads/2009/12/IN-parameter-offices-in-France.png?44591f)

OUT parameter example

The following stored procedure returns the number of orders by order status. It has two parameters:

* orderStatus: IN parameter that is the order status which you want to count the orders.
* total: OUT parameter that stores the number of orders for a specific order status.

The following is the source code of the CountOrderByStatus stored procedure.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | DELIMITER $$  CREATE PROCEDURE CountOrderByStatus(          IN orderStatus VARCHAR(25),          OUT total INT)  BEGIN      SELECT count(orderNumber)      INTO total      FROM orders      WHERE status = orderStatus;  END$$  DELIMITER ; |

To get the number of shipped orders, we call the CountOrderByStatus stored procedure and pass the order status as Shipped, and also pass an argument ( @total) to get the returnvalue.



|  |  |
| --- | --- |
| 1  2  3 | CALL CountOrderByStatus('Shipped',@total);    SELECT @total; |

MySQL Stored Procedure Parameters - OUT parameter order shipped

To get the number of orders that are in process, we call the CountOrderByStatus stored procedure as follows:



|  |  |
| --- | --- |
| 1  2  3 | CALL CountOrderByStatus('in process',@total);    SELECT @total AS  total\_in\_process; |

MySQL Stored Procedure Parameters - OUT parameter orders in process

INOUT parameter example

The following example demonstrates how to use INOUT parameter in the stored procedure.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | DELIMITER $$  CREATE PROCEDURE set\_counter(INOUT count INT(4),IN inc INT(4))  BEGIN      SET count = count + inc;  END$$  DELIMITER ; |

How it works.

* The set\_counter stored procedure accepts one INOUT parameter ( count) and oneIN parameter ( inc).
* Inside the stored procedure, we increase the counter ( count) by the value of the incparameter.

See how we call the set\_counter stored procedure:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SET @counter = 1;  CALL set\_counter(@counter,1); -- 2  CALL set\_counter(@counter,1); -- 3  CALL set\_counter(@counter,5); -- 8  SELECT @counter; -- 8 |

In this tutorial, we have shown you how to define parameters in stored procedures, and introduced you to different parameter modes including IN, OUT and INOUT.

MySQL IF Statement

**Summary***:*in this tutorial, you will learn how to use **MySQL IF statement** to execute a block of SQL code based on conditions.

The MySQL IF statement allows you to execute a set of SQL statements based on a certain condition or value of an expression. To form an expression in MySQL, you can combine literals,[variables](http://www.mysqltutorial.org/variables-in-stored-procedures.aspx" \o "Variables in Stored Procedures), operators, and even [functions](http://www.mysqltutorial.org/mysql-functions.aspx). An expression can return three value TRUE, FALSE or NULL.

MySQL IF statement syntax

The following illustrates the syntax of the IF statement:



|  |  |
| --- | --- |
| 1  2  3  4 | IF if\_expression THEN commands     [ELSEIF elseif\_expression THEN commands]     [ELSE commands]     END IF; |

If the *if\_expression* evaluates to TRUE the commands in the IF branch will execute. If it evaluates to FALSE, MySQL will check the *elseif\_expression* and execute the commands in ELSEIF branch if the *elseif\_expression* evaluates to TRUE.

The IF statement may have multiple ELSEIF branches to check multiple expressions.  If no expression evaluates to TRUE, the commands in the ELSE branch will execute.

MySQL IF statement examples

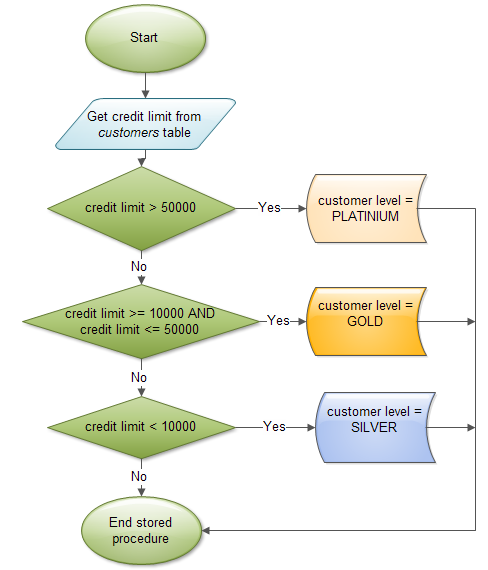
Let’s take a look at an example of how to use MySQL IF statements.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | DELIMITER $$    CREATE PROCEDURE GetCustomerLevel(      in  p\_customerNumber int(11),      out p\_customerLevel  varchar(10))  BEGIN      DECLARE creditlim double;        SELECT creditlimit INTO creditlim      FROM customers      WHERE customerNumber = p\_customerNumber;        IF creditlim > 50000 THEN      SET p\_customerLevel = 'PLATINUM';      ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN          SET p\_customerLevel = 'GOLD';      ELSEIF creditlim < 10000 THEN          SET p\_customerLevel = 'SILVER';      END IF;    END$$ |

We pass customer number to the stored procedure to get customer level based on credit limit. We use IF ELSEIF and ELSE statement to check customer credit limit against multiple values.

The following is the flow chart  that demonstrates the logic of determining customer level.

[](http://www.mysqltutorial.org/wp-content/uploads/2013/01/mysql-if-statement-flow-chart.png?44591f)

In this tutorial, you have learned how to use MySQL IF statement to execute a block of SQL code based on conditions.

MySQL CASE Statement

**Summary**: in this tutorial, you will learn how to use **MySQL CASE** statements to construct complex conditionals.

Besides the [IF statement](http://www.mysqltutorial.org/mysql-if-statement/), MySQL also provides an alternative conditional statement called MySQL CASE. The MySQL CASE statement makes the code more readable and efficient.

There are two forms of the CASE statements: simple and searched CASE statements.

Simple CASE statement

Let’s take a look at the syntax of the simple CASE statement:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CASE  case\_expression     WHEN when\_expression\_1 THEN commands     WHEN when\_expression\_2 THEN commands     ...     ELSE commands  END CASE; |

You use the simple CASE statement to check the value of an expression against a set of unique values.

The*case\_expression*can be any valid expression. We compare the value of thecase\_expression with  when\_expression in each WHEN clause e.g.,when\_expression\_1, when\_expression\_2, etc. If the value of the case\_expressionand when\_expression\_n are equal, the  commands in the corresponding WHEN branch executes.

In case none of the when\_expression in the WHEN clause matches the value of thecase\_expression*,*the commands in the ELSE clause will execute. The ELSE clause is optional. If you omit the ELSE clause and no match found, MySQL will raise an error.

The following example illustrates how to use the simple CASE statement:

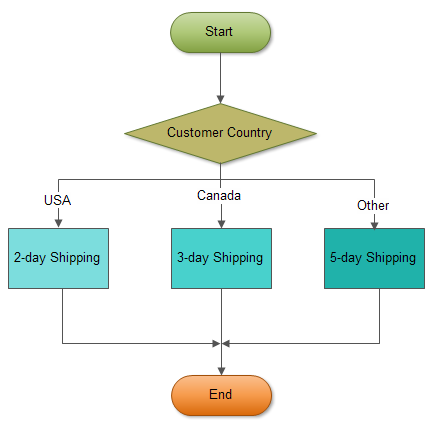


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | DELIMITER $$    CREATE PROCEDURE GetCustomerShipping(          in  p\_customerNumber int(11),          out p\_shiping        varchar(50))  BEGIN      DECLARE customerCountry varchar(50);        SELECT country INTO customerCountry      FROM customers      WHERE customerNumber = p\_customerNumber;        CASE customerCountry          WHEN  'USA' THEN             SET p\_shiping = '2-day Shipping';          WHEN 'Canada' THEN             SET p\_shiping = '3-day Shipping';          ELSE             SET p\_shiping = '5-day Shipping';      END CASE;    END$$ |

How the stored procedure works.

* The GetCustomerShipping stored procedure accepts customer number as an IN[parameter](http://www.mysqltutorial.org/stored-procedures-parameters.aspx)and returns shipping period based on the country of the customer.
* Inside the stored procedure, first we get the country of the customer based on the input customer number. Then we use the simple CASE statement to compare the country of the customer to determine the shipping period. If the customer locates in USA, the shipping period is 2-day shipping. If the customer is in Canada, the shipping period is 3-day shipping. The customers from other countries have 5-day shipping.

The following flowchart demonstrates the logic of determining shipping period.

[](http://www.mysqltutorial.org/wp-content/uploads/2013/01/mysql-case-statement.png?44591f)

The following is the test script for the stored procedure above:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | SET @customerNo = 112;    SELECT country into @country  FROM customers  WHERE customernumber = @customerNo;    CALL GetCustomerShipping(@customerNo,@shipping);    SELECT @customerNo AS Customer,         @country    AS Country,         @shipping   AS Shipping; |

[MySQL CASE - Simple CASE statement output](http://www.mysqltutorial.org/wp-content/uploads/2013/01/mysql-case-simple.png?44591f)

Searched CASE statement

The simple CASE statement only allows you match a value of an expression against a set of distinct values. In order to perform more complex matches such as ranges you use the *searched*CASE statement. The searched CASE statement is equivalent to the IF statement, however its construct is much more readable.

The following illustrates the syntax of the searched CASE statement:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CASE      WHEN condition\_1 THEN commands      WHEN condition\_2 THEN commands      ...      ELSE commands  END CASE; |

MySQL evaluates each condition in the WHEN clause until it finds a condition whose value isTRUE, then corresponding commands in the THEN clause will execute.

If no condition is TRUE , the command in the ELSE clause will execute. If you don’t specify the ELSE clause and no condition is TRUE, MySQL will issue an error message.

MySQL does not allow you to have empty commands in the THEN or ELSE clause. If you don’t want to handle the logic in the ELSE clause while preventing MySQL raise an error, you can put an empty BEGIN END block in the ELSE clause.

The following example demonstrates using searched CASE statement to find customer levelSILVER, GOLD or PLATINUM based on customer’s credit limit.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | DELIMITER $$    CREATE PROCEDURE GetCustomerLevel(      in  p\_customerNumber int(11),      out p\_customerLevel  varchar(10))  BEGIN      DECLARE creditlim double;        SELECT creditlimit INTO creditlim      FROM customers      WHERE customerNumber = p\_customerNumber;        CASE          WHEN creditlim > 50000 THEN             SET p\_customerLevel = 'PLATINUM';          WHEN (creditlim <= 50000 AND creditlim >= 10000) THEN             SET p\_customerLevel = 'GOLD';          WHEN creditlim < 10000 THEN             SET p\_customerLevel = 'SILVER';      END CASE;    END$$ |

If the credit limit is

* greater than 50K, then the customer is PLATINUM customer
* less than 50K and greater than 10K, then the customer is GOLD customer
* less than 10K, then the customer is SILVER customer.

We can test our stored procedure by executing the following test script:



|  |  |
| --- | --- |
| 1  2 | CALL GetCustomerLevel(112,@level);  SELECT @level AS 'Customer Level'; |

[MySQL Searched CASE output](http://www.mysqltutorial.org/wp-content/uploads/2013/01/mysql-searched-case.png?44591f)

In this tutorial, we’ve shown you how to use two forms of the MySQL CASE statements including simple CASE statement and searched CASE statement.

Hints for Choosing Between IF and CASE Statements

**Summary**: in this tutorial, we will give you some hints so that you can choose between [IF](http://www.mysqltutorial.org/mysql-if-statement/)and[CASE](http://www.mysqltutorial.org/mysql-case-statement/)statement in stored procedures.

MySQL provides both IF and CASE statements to enable you to execute a block of SQL code based on certain conditions, which is known as flow control. So what statement should you use? For the most developers, choosing between IF and CASE is just a matter of personal preference. However when you decide to use IF or CASE,  you should take the following points into the consideration:

* A [simple CASE statement](http://www.mysqltutorial.org/mysql-case-statement/) is more readable than the [IF statement](http://www.mysqltutorial.org/mysql-if-statement/) when you compare a single expression against a range of unique values.  In addition, the simple CASE statement is more efficient than the IF statement.
* When you check complex expressions based on multiple values, the IF statement is easier to understand.
* If you choose to use the CASE statement, you have to make sure that at least one of the CASE condition is matched. Otherwise you need to define an error handle to catch the error. Recall that you don’t have to do this with the IF statement.
* In most organization, there is always something called development guidelines document that provides developers with naming convention and guidelines on programming style. You should refer to this document and follow the development practices.
* In some situations, mixing between IF and CASE make your stored procedure more readable and efficient.

Loop in Stored Procedures

**Summary***:* in this tutorial, you will learn how to use various loop statements in MySQL includingWHILE, REPEAT and LOOP to run a block of code repeatedly based on a condition.

MySQL provides loop statements that allow you to execute a block of SQL code repeatedly based on a condition. There are three loop statements in MySQL: WHILE, REPEAT andLOOP.

We will examine each statement in more detail in the following section.

WHILE loop

The syntax of the WHILE statement is as follows:



|  |  |
| --- | --- |
| 1  2  3 | WHILE expression DO     Statements  END WHILE |

The WHILE loop checks the expression at the beginning of each iteration. If  the expressionevaluates to TRUE,  MySQL will executes statements between WHILE and END WHILE untilthe expression evaluates to FALSE. The WHILE loop is called pretest loop because it checksthe expression before the statements execute.

Here is an example of using the WHILE loop statement in stored procedure:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | DELIMITER $$  DROP PROCEDURE IF EXISTS WhileLoopProc$$  CREATE PROCEDURE WhileLoopProc()         BEGIN                 DECLARE x  INT;                 DECLARE str  VARCHAR(255);                 SET x = 1;                 SET str =  '';                 WHILE x  <= 5 DO                             SET  str = CONCAT(str,x,',');                             SET  x = x + 1;                 END WHILE;                 SELECT str;         END$$     DELIMITER ; |

In the stored procedure above:

* First, we build str string repeatedly until the value of the x [variable](http://www.mysqltutorial.org/variables-in-stored-procedures.aspx)is greater than 5.
* Then, we display the final string using the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx).

Notice that if we don’t initialize x variable, its default value is NULL. Therefore the condition in the WHILE loop statement is always TRUE and you will have a indefinite loop, which is not expected.

REPEAT loop

The syntax of the REPEAT loop statement is as follows:



|  |  |
| --- | --- |
| 1  2  3  4 | REPEAT  Statements;  UNTIL expression  END REPEAT |

First MySQL executes the statements, and then it evaluates the expression. If theexpression evaluates to TRUE, MySQL executes the statements repeatedly until theexpression evaluates to FALSE.

Because the REPEAT loop statement checks the expression after the execution ofstatements therefore the REPEAT loop statement is also known as post-test loop.

We can rewrite the stored procedure that uses WHILE loop statement above using the REPEATloop statement:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | DELIMITER $$  DROP PROCEDURE IF EXISTS RepeatLoopProc$$  CREATE PROCEDURE RepeatLoopProc()         BEGIN                 DECLARE x  INT;                 DECLARE str  VARCHAR(255);                 SET x = 1;                 SET str =  '';                 REPEAT                             SET  str = CONCAT(str,x,',');                             SET  x = x + 1;                 UNTIL x  > 5                 END REPEAT;                 SELECT str;         END$$  DELIMITER ; |

It is noticed that there is no delimiter semicolon (;) in the UNTIL expression.

LOOP, LEAVE and ITERATE Statements

The LEAVE statement allows you to exit the loop immediately without waiting for checking the condition. The LEAVE statement works like the  break statement in other languages such as PHP, C/C++, Java, etc.

The ITERATE statement allows you to skip the entire code under it and start a new iteration. The ITERATE statement is similar to the continuestatement in PHP, C/C++, Java, etc.

MySQL also gives you a LOOP statement that allows you to execute a block of code repeatedly with an additional flexibility of using a loop label.

The following is an example of using the LOOP loop statement.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | DELIMITER $$  DROP PROCEDURE IF EXISTS LOOPLoopProc$$  CREATE PROCEDURE LOOPLoopProc()         BEGIN                 DECLARE x  INT;                 DECLARE str  VARCHAR(255);                 SET x = 1;                 SET str =  '';                 loop\_label:  LOOP                             IF  x > 10 THEN                                 LEAVE  loop\_label;                             END  IF;                             SET  x = x + 1;                             IF  (x mod 2) THEN                                 ITERATE  loop\_label;                             ELSE                                 SET  str = CONCAT(str,x,',');                             END  IF;                   END LOOP;                 SELECT str;         END$$  DELIMITER ; |

* The stored procedure only constructs string with even numbers e.g., 2, 4, 6, etc.
* We put a loop\_labelloop label before the LOOP statement.
* If the value of  x is greater than 10, the loop is terminated because of the LEAVEstatement.
* If the value of the x is an odd number, the   ITERATE statement ignores everything below it and starts a new iteration.
* If the value of the x is an even number, the block in the ELSE statement will build the string with even numbers.

In this tutorial, you have learned various MySQL loop statements to execute a block of code repeatedly based on a condition.

MySQL Cursor

**Summary**: in this tutorial, you will learn how to use **MySQL cursor** in stored procedures to iterate through a result set returned by a SELECT statement.

Introduction to MySQL cursor

To handle a result set inside a [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), you use a cursor. A cursor allows you to [iterate](http://www.mysqltutorial.org/stored-procedures-loop.aspx)a set of rows returned by a query and process each row accordingly.

MySQL cursor is read only, non-scrollable and asensitive.

* **Read only**: you cannot update data in the underlying table through the cursor.
* **Non-scrollable**: you can only fetch rows in the order determined by the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx). You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.
* **Asensitive**: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore it is safer if you don’t update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

You can use MySQL cursors in [stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), stored functions and [triggers](http://www.mysqltutorial.org/mysql-triggers.aspx).

Working with MySQL cursor

First, you have to declare a cursor by using the DECLARE statement:



|  |  |
| --- | --- |
| 1 | DECLARE cursor\_name CURSOR FOR SELECT\_statement; |

The cursor declaration must be after any [variable](http://www.mysqltutorial.org/variables-in-stored-procedures.aspx)declaration. If you declare a cursor before variables declaration, MySQL will issue an error. A cursor must always be associated with aSELECT statement.

Next, you open the cursor by using the OPEN statement. The OPEN statement initializes the result set for the cursor therefore you must call the OPEN statement before fetching rows from the result set.



|  |  |
| --- | --- |
| 1 | OPEN cursor\_name; |

Then, you use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.



|  |  |
| --- | --- |
| 1 | FETCH cursor\_name INTO variables list; |

After that, you can check to see if there is any row available before fetching it.

Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it as follows:



|  |  |
| --- | --- |
| 1 | CLOSE cursor\_name; |

When the cursor is no longer used, you should close it.

When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row. Because each time you call the FETCHstatement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

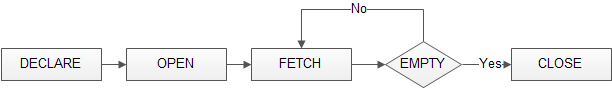
To declare a NOT FOUND handler, you use the following syntax:



|  |  |
| --- | --- |
| 1 | DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1; |

Where finished is a variable to indicate that the cursor has reached the end of the result set. Notice that the handler declaration must appear after variable and cursor declaration inside the stored procedures.

The following diagram illustrates how MySQL cursor works.



MySQL Cursor Example

We are going to develop a stored procedure that builds an email list of all employees in theemployees table in the [MySQL sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx).

First, we declare some variables, a cursor for looping over the emails of employees, and a NOT FOUND handler:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | DECLARE finished INTEGER DEFAULT 0;  DECLARE email varchar(255) DEFAULT "";    -- declare cursor for employee email  DEClARE email\_cursor CURSOR FOR      SELECT email FROM employees;    -- declare NOT FOUND handler  DECLARE CONTINUE HANDLER  FOR NOT FOUND SET finished = 1; |

Next, we open the email\_cursor by using the OPEN statement:



|  |  |
| --- | --- |
| 1 | OPEN email\_cursor; |

Then, we iterate the email list, and concatenate all emails where each email is separated by a semicolon(;):



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | get\_email: LOOP      FETCH email\_cursor INTO v\_email;      IF v\_finished = 1 THEN          LEAVE get\_email;      END IF;      -- build email list      SET email\_list = CONCAT(v\_email,";",email\_list);  END LOOP get\_email; |

After that, inside the loop we used the  v\_finished variable to check if there is any email in the list to terminate the loop.

Finally, we close the cursor using the CLOSE statement:



|  |  |
| --- | --- |
| 1 | CLOSE email\_cursor; |

The build\_email\_list stored procedure is as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | DELIMITER $$    CREATE PROCEDURE build\_email\_list (INOUT email\_list varchar(4000))  BEGIN        DECLARE v\_finished INTEGER DEFAULT 0;          DECLARE v\_email varchar(100) DEFAULT "";        -- declare cursor for employee email      DEClARE email\_cursor CURSOR FOR          SELECT email FROM employees;        -- declare NOT FOUND handler      DECLARE CONTINUE HANDLER          FOR NOT FOUND SET v\_finished = 1;        OPEN email\_cursor;        get\_email: LOOP            FETCH email\_cursor INTO v\_email;            IF v\_finished = 1 THEN              LEAVE get\_email;          END IF;            -- build email list          SET email\_list = CONCAT(v\_email,";",email\_list);        END LOOP get\_email;        CLOSE email\_cursor;    END$$    DELIMITER ; |

You can test the build\_email\_list stored procedure using the following script:



|  |  |
| --- | --- |
| 1  2  3 | SET @email\_list = "";  CALL build\_email\_list(@email\_list);  SELECT @email\_list; |

In this tutorial, we have shown you how to use MySQL cursor to iterate a result set and process each row accordingly.

Listing Stored Procedures in a MySQL Database

**Summary**: in this tutorial, we will show you how to list all stored procedures in a MySQL database**,**and introduce you to a very useful statement that displays stored procedure’s source code.

MySQL provides us with several useful statements that help us to manage stored procedures effectively. Those statements include listing stored procedures and showing the stored procedure’s source code.

Displaying stored procedures characteristics

To display characteristics of a stored procedure, you use the following statement:



|  |  |
| --- | --- |
| 1 | SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE expr]; |

The SHOW PROCEDURE STATUS statement is a MySQL extension to SQL standard. This statement gives you the stored procedure’s characteristics including database, stored procedure name, type, creator and so on.

You can use [LIKE](http://www.mysqltutorial.org/sql-like-mysql.aspx)or [WHERE clause](http://www.mysqltutorial.org/mysql-where/) to filter out the stored procedure based on various criteria.

To list all stored procedures of the databases that you have the privilege to access, you use the SHOW PROCEDURE STATUS statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW PROCEDURE STATUS; |

If you want to show just stored procedure in a particular database, you can use the WHEREclause in the  SHOW PROCEDURE STATUS statement:



|  |  |
| --- | --- |
| 1 | SHOW PROCEDURE STATUS WHERE db = 'classicmodels'; |

If you want to show stored procedures that have a particular pattern e.g., its name containsproduct, you can use the [LIKE operator](http://www.mysqltutorial.org/mysql-like) as the following command:



|  |  |
| --- | --- |
| 1 | SHOW PROCEDURE STATUS WHERE name LIKE '%product%' |

Displaying stored procedure’s source code

To display source code of a particular stored procedure, you use the  SHOW CREATE PROCEDURE statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW CREATE PROCEDURE stored\_procedure\_name |

You specify the name of the stored procedure after the  SHOW CREATE PROCEDURE keywords. For example, to display the code of the GetAllProducts stored procedure, you use the following statement:



|  |  |
| --- | --- |
| 1 | SHOW CREATE PROCEDURE GetAllProducts |

In this tutorial, you have learned some useful statements including  SHOW PROCEDURE STATUSand  SHOW CREATE PROCEDURE statements to list stored procedures in a database and get thesource code of the stored procedure.

MySQL Error Handling in Stored Procedures

This tutorial shows you how to use MySQL handler to handle exceptions or errors encountered in stored procedures.

When an error occurs inside a stored procedure, it is important to handle it appropriately, such as continuing or exiting the current code block’s execution, and issuing a meaningful error message.

MySQL provides an easy way to define handlers that handle from general conditions such as warnings or exceptions to specific conditions e.g., specific error codes.

Declaring a handler

To declare a handler, you use the  DECLARE HANDLER statement as follows:



|  |  |
| --- | --- |
| 1 | DECLARE action HANDLER FOR condition\_value statement; |

If a condition whose value matches the  condition\_value, MySQL will execute thestatement and continue or exit the current code block based on the action.

The action accepts one of the following values:

* CONTINUE:  the execution of the enclosing code block ( BEGIN … END) continues.
* EXIT: the execution of the enclosing code block, where the handler is declared, terminates.

The  condition\_value specifies a particular condition or a class of conditions that activates the handler. The  condition\_value accepts one of the following values:

* A MySQL error code.
* A standard SQLSTATE value. Or it can be an SQLWARNING, NOTFOUND orSQLEXCEPTION condition, which is shorthand for the class of SQLSTATE values. TheNOTFOUND condition is used for a [cursor](http://www.mysqltutorial.org/mysql-cursor/)or  SELECT INTO variable\_list statement.
* A named condition associated with either a MySQL error code or SQLSTATE value.

The statement could be a simple statement or a compound statement enclosing by theBEGIN and END keywords.

MySQL error handling examples

Let’s look into several examples of declaring handlers.

The following handler means if an error occurs, set the value of the  has\_error variable to 1 and continue the execution.



|  |  |
| --- | --- |
| 1 | DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET has\_error = 1; |

The following is another handler; it means that in case any error occurs, rollback the previous operation, issue an error message and exit the current code block. If you declare it inside the  BEGIN END block of a stored procedure, it will terminate stored procedure immediately.



|  |  |
| --- | --- |
| 1  2  3  4  5 | DECLARE EXIT HANDLER FOR SQLEXCEPTION  BEGIN  ROLLBACK;  SELECT 'An error has occurred, operation rollbacked and the stored procedure was terminated';  END; |

If there are no more rows to fetch, in case of a [cursor](http://www.mysqltutorial.org/mysql-cursor/)or [SELECT INTO](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx)statement, set the value of the  no\_row\_found variable to 1 and continue execution.



|  |  |
| --- | --- |
| 1 | DECLARE CONTINUE HANDLER FOR NOT FOUND SET no\_row\_found = 1; |

If a duplicate key error occurs, MySQL error 1062 is issued. The following handler issues an error message and continues execution.



|  |  |
| --- | --- |
| 1  2 | DECLARE CONTINUE HANDLER FOR 1062  SELECT 'Error, duplicate key occurred'; |

MySQL handler example in stored procedures

First, we create a new table named  article\_tags for the demonstration:



|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE article\_tags(      article\_id INT,      tag\_id     INT,      PRIMARY KEY(article\_id,tag\_id)  ); |

The  article\_tags table stores the relationships between articles and tags. Each article may have many tags and vice versa. For the sake of simplicity, we don’t create articles andtags tables, as well as the [foreign keys](http://www.mysqltutorial.org/mysql-foreign-key/) in the  article\_tags table.

Second, we create a stored procedure that inserts a pair of ids of article and tag into the article\_tags table:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | DELIMITER $$    CREATE PROCEDURE insert\_article\_tags(IN article\_id INT, IN tag\_id INT)  BEGIN        DECLARE CONTINUE HANDLER FOR 1062      SELECT CONCAT('duplicate keys (',article\_id,',',tag\_id,') found') AS msg;        -- insert a new record into article\_tags      INSERT INTO article\_tags(article\_id,tag\_id)      VALUES(article\_id,tag\_id);        -- return tag count for the article      SELECT COUNT(\*) FROM article\_tags;  END |

Third, we add tag id 1, 2 and 3 for the article 1 by calling the  insert\_article\_tags stored procedure as follows:



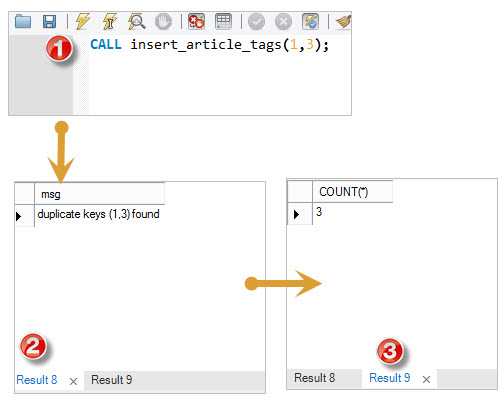
|  |  |
| --- | --- |
| 1  2  3 | CALL insert\_article\_tags(1,1);  CALL insert\_article\_tags(1,2);  CALL insert\_article\_tags(1,3); |

Fourth, let’s try to insert a duplicate key to see if the handler is really invoked.



|  |  |
| --- | --- |
| 1 | CALL insert\_article\_tags(1,3); |

We got an error message. However, because we declared the handler as a CONTINUE handler, the stored procedure continued execution. As the result, we got the tag count for the article as well.



If we change the CONTINUE in the handler declaration to EXIT, we will get only the error message.

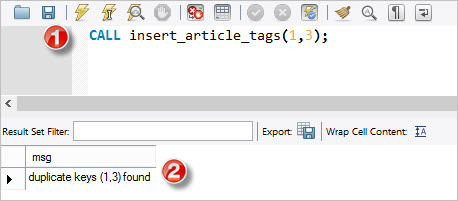


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | DELIMITER $$    CREATE PROCEDURE insert\_article\_tags\_2(IN article\_id INT, IN tag\_id INT)  BEGIN        DECLARE EXIT HANDLER FOR SQLEXCEPTION      SELECT 'SQLException invoked';        DECLARE EXIT HANDLER FOR 1062          SELECT 'MySQL error code 1062 invoked';        DECLARE EXIT HANDLER FOR SQLSTATE '23000'      SELECT 'SQLSTATE 23000 invoked';        -- insert a new record into article\_tags      INSERT INTO article\_tags(article\_id,tag\_id)         VALUES(article\_id,tag\_id);        -- return tag count for the article      SELECT COUNT(\*) FROM article\_tags;  END |

Now, we can try to add a duplicate key to see the effect.



|  |  |
| --- | --- |
| 1 | CALL insert\_article\_tags\_2(1,3); |



MySQL handler precedence

In case there are multiple handlers that are eligible for handling an error, MySQL will call the most specific handler to handle the error.

An error always maps to one MySQL error code so a MySQL it is the most specific. An SQLSTATEmay map to many MySQL error codes therefore it is less specific. An SQLEXCPETION or anSQLWARNING is the shorthand for a class of SQLSTATES values so it is the most generic.

Based on the handler precedence’s rules,  MySQL error code handler, SQLSTATE handler andSQLEXCEPTION takes the first, second and third precedence.

Suppose we declare three handlers in the  insert\_article\_tags\_3 stored procedure as follows:



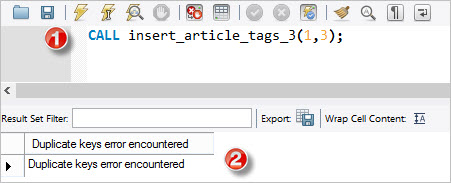
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | DELIMITER $$    CREATE PROCEDURE insert\_article\_tags\_3(IN article\_id INT, IN tag\_id INT)  BEGIN        DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered';      DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered';      DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000';        -- insert a new record into article\_tags      INSERT INTO article\_tags(article\_id,tag\_id)      VALUES(article\_id,tag\_id);        -- return tag count for the article      SELECT COUNT(\*) FROM article\_tags;  END |

We now try to insert a duplicate key into the article\_tags table by calling the stored procedure:



|  |  |
| --- | --- |
| 1 | CALL insert\_article\_tags\_3(1,3); |

As you see the MySQL error code handler is called.



Using named error condition

Let’s start with an error handler declaration.



|  |  |
| --- | --- |
| 1  2 | DECLARE EXIT HANDLER FOR 1051 SELECT 'Please create table abc first';  SELECT \* FROM abc; |

What does the number 1051 really mean? Imagine you have a big stored procedure polluted with those numbers all over places; it will become a nightmare for the maintenance developers.

Fortunately, MySQL provides us with the  DECLARE CONDITION statement that declares a named error condition, which associates with a condition. The syntax of the  DECLARE CONDITION statement is as follows:



|  |  |
| --- | --- |
| 1 | DECLARE condition\_name CONDITION FOR condition\_value; |

The  condition\_value can be a MySQL error code such as 1015 or a SQLSTATE value. The  condition\_value is represented by the condition\_name.

After declaration, you can refer to the  condition\_name instead of the condition\_value.

So we can rewrite the code above as follows:



|  |  |
| --- | --- |
| 1  2  3 | DECLARE table\_not\_found CONDITION for 1051;  DECLARE EXIT HANDLER FOR  table\_not\_found SELECT 'Please create table abc first';  SELECT \* FROM abc; |

This code is obviously more readable than the previous one.

Notice that the condition declaration must appear before handler or cursor declarations.

Raising Error Conditions with SIGNAL / RESIGNAL Statements

In this tutorial, you will learn how to use SIGNAL and RESIGNAL statements to raise error conditions inside stored procedures.

MySQL SIGNAL statement

You use the SIGNAL statement to return an error or warning condition to the caller from a stored program e.g., stored procedure, [trigger](http://www.mysqltutorial.org/mysql-triggers.aspx)or [event](http://www.mysqltutorial.org/mysql-triggers/modifying-mysql-events/). The SIGNAL statement provides you with control over which information to return such as SQLSTATE value and message.

The following illustrates syntax of the SIGNAL statement:



|  |  |
| --- | --- |
| 1  2  3 | SIGNAL SQLSTATE | condition\_name  SET condition\_information\_item\_name\_1 = value\_1,      condition\_information\_item\_name\_1 = value\_2, etc; |

Following the SIGNAL keyword is an SQLSTATE value or a condition name declared by the DECLARE CONDITION statement. Notice that the SIGNAL statement must always specify anSQLSTATE value or a named condition that defined with an  SQLSTATE value.

To provide the caller with information, you use the SET clause. If you want to return multiple condition information item names with values, you need to separate each name/value pair by a comma.

The  condition\_information\_item\_name can be MESSAGE\_TEXT, MYSQL\_ERRORNO,CURSOR\_NAME, etc.

The following stored procedure adds an order line item into an existing sales order. It issues an error message if the order number does not exist.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | DELIMITER $$    CREATE PROCEDURE AddOrderItem(                   in orderNo int,               in productCode varchar(45),               in qty int,                           in price double,                           in lineNo int )  BEGIN      DECLARE C INT;        SELECT COUNT(orderNumber) INTO C      FROM orders      WHERE orderNumber = orderNo;        -- check if orderNumber exists      IF(C != 1) THEN          SIGNAL SQLSTATE '45000'              SET MESSAGE\_TEXT = 'Order No not found in orders table';      END IF;      -- more code below      -- ...  END |

First, it counts the orders with the input order number that we pass to the stored procedure.

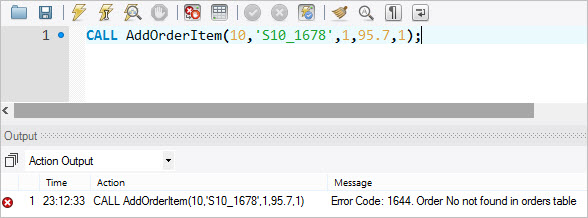
Second, if the number of order is not 1, it raises an error with  SQLSTATE 45000 along with an error message saying that order number does not exist in the orders table.

Notice that 45000 is a generic SQLSTATE value that illustrates an unhandled user-defined exception.

If we call the stored procedure  AddOrderItem() and pass a nonexistent order number, we will get an error message.



|  |  |
| --- | --- |
| 1 | CALL AddOrderItem(10,'S10\_1678',1,95.7,1); |



MySQL RESIGNAL statement

Besides the SIGNAL statement, MySQL also provides the RESIGNAL statement that is used to raise a warning or error condition.

The RESIGNAL statement is similar to SIGNAL statement in term of functionality and syntax, except that:

* You must use the RESIGNAL statement within an error or warning handler, otherwise you will get an error message saying that “RESIGNAL when handler is not active”. Notice that you can use SIGNAL statement anywhere inside a stored procedure.
* You can omit all attributes of the RESIGNAL statement, even the SQLSTATE value.

If you use the RESIGNAL statement alone, all attributes are the same as the ones passed to the condition handler.

The following stored procedure changes the error message before issuing it to the caller.

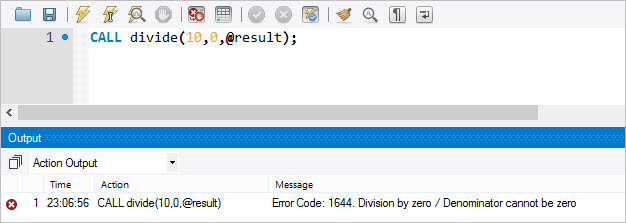


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | DELIMITER $$    CREATE PROCEDURE Divide(IN numerator INT, IN denominator INT, OUT result double)  BEGIN      DECLARE division\_by\_zero CONDITION FOR SQLSTATE '22012';        DECLARE CONTINUE HANDLER FOR division\_by\_zero      RESIGNAL SET MESSAGE\_TEXT = 'Division by zero / Denominator cannot be zero';      --      IF denominator = 0 THEN          SIGNAL division\_by\_zero;      ELSE          SET result := numerator / denominator;      END IF;  END |

Let’s call the  Divide() stored procedure.



|  |  |
| --- | --- |
| 1 | CALL Divide(10,0,@result); |



In this tutorial, we have shown you how to raise error conditions inside stored procedure usingSIGNAL and  RESIGNAL statements

MySQL Stored Function

In this tutorial, you will learn how to create stored functions using CREATE FUNCTION statement.

A stored function is a special kind stored program that returns a single value. You use stored functions to encapsulate common formulas or business rules that may be reusable among SQL statements or stored programs.

Different from a [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), you can use a stored function in SQL statements wherever an expression is used. This helps improve the readability and maintainability of the procedural code.

MySQL stored function syntax

The following illustrates the simplest syntax for creating a new stored function:



|  |  |
| --- | --- |
| 1  2  3  4 | CREATE FUNCTION function\_name(param1,param2,…)      RETURNS datatype     [NOT] DETERMINISTIC  statements |

First, you specify the name of the stored function after  CREATE FUNCTION keywords.

Second, you list all parameters of the stored function. By default, all parameters are implicitlyIN parameters. You cannot specify IN, OUT or INOUT modifiers to the parameters.

Third, you must specify the data type of the return value in the RETURNS statement. It can be any valid [MySQL data types](http://www.mysqltutorial.org/mysql-data-types.aspx).

Fourth, for the same input parameters, if the stored function returns the same result, it is considered deterministic and not deterministic otherwise.

You have to decide whether a stored function is deterministic or not. If you declare it incorrectly, the stored function may produced an unexpected result, or the available optimization is not used which degrade the performance.

Fifth, you write the code in the statements section. It can be a single statement or a compound statement. Inside the statements section, you have to specify at least one RETURN statement.

The RETURN statement returns a value to the caller. Whenever the RETURN statement is reached, the stored function’s execution is terminated immediately.

MySQL stored function example

The following example is a function that returns level of customer based on credit limit.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | DELIMITER $$    CREATE FUNCTION CustomerLevel(p\_creditLimit double) RETURNS VARCHAR(10)      DETERMINISTIC  BEGIN      DECLARE lvl varchar(10);        IF p\_creditLimit > 50000 THEN          SET lvl = 'PLATINUM';      ELSEIF (p\_creditLimit <= 50000 AND p\_creditLimit >= 10000) THEN          SET lvl = 'GOLD';      ELSEIF p\_creditLimit < 10000 THEN          SET lvl = 'SILVER';      END IF;        RETURN (lvl);  END |

Now we can call the CustomerLevel() in an [SQL SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT customerName,         CustomerLevel(creditLimit)  FROM customers; |

We also rewrite the  GetCustomerLevel() stored procedure that we developed in the [MySQL IF statement](http://www.mysqltutorial.org/mysql-if-statement/) tutorial as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | DELIMITER $$    CREATE PROCEDURE GetCustomerLevel(      IN  p\_customerNumber INT(11),      OUT p\_customerLevel  varchar(10)  )  BEGIN      DECLARE creditlim DOUBLE;        SELECT creditlimit INTO creditlim      FROM customers      WHERE customerNumber = p\_customerNumber;        SELECT CUSTOMERLEVEL(creditlim)      INTO p\_customerLevel;    END |

As you can see, the  GetCustomerLevel() stored procedure is much more readable when using the  CustomerLevel() stored function.

Notice that a stored function returns a single value only. If you include a SELECT statement without INTO clause, you will get an error.

In addition, if a stored function contains SQL statements, you should not use it inside other SQL statements; otherwise the stored function will cause the performance of the SQL statements to degrade.

MySQL Triggers

In this section, you will learn how to work with **MySQL triggers**. By definition, a trigger or database trigger is a stored program that is executed automatically to respond to a specific event associated with table e.g.,  insert, update or delete.

Database trigger is powerful tool for protecting the integrity of the data in your MySQL databases. Database triggers are very useful to automate some database operations such as audit logging.

Introduction to SQL Trigger

**Summary:** in this tutorial, we will  give you a brief overview of **SQL trigger**, its advantages and disadvantages.

A SQL trigger is a set of  SQL statements stored in the database catalog. A SQL trigger is executed or fired whenever an event associated with a table occurs e.g.,  insert, update or delete.

A SQL trigger is a special type of [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx). It is special because it is not called directly like a stored procedure. The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.

It is important to understand SQL trigger’s advantages and disadvantages so that you can use it appropriately. In the following sections, we will discuss about the advantages and disadvantages of using SQL triggers.

Advantages of using SQL triggers

* SQL triggers provide an alternative way to check the integrity of data.
* SQL triggers can catch errors in business logic in the database layer.
* SQL triggers provide an alternative way to run scheduled tasks. By using SQL triggers, you don’t have to wait to run the scheduled tasks because the triggers are invoked  automatically *before* or *after* a change  is made to the data in tables.
* SQL triggers are very useful to audit the changes of data in tables.

Disadvantages of using SQL triggers

* SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer. For example, you can validate user’s inputs in the client side by using JavaScript or in the server side using server side scripting languages such as JSP, PHP, ASP.NET, Perl, etc.
* SQL triggers are invoked and executed invisibly from client-applications therefore it is difficult to figure out what happen in the database layer.
* SQL triggers may increase the overhead of the database server.

Triggers or [stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx)? It is recommended that if you have no way to get the work done with [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), think about SQL trigger

MySQL Triggers Implementation

**Summary**: in this tutorial, you will learn about **MySQL triggers implementation**. In addition, we will show you how MySQL stores trigger definitions and the limitations of triggers in MySQL.

Introduction to MySQL triggers

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table. A trigger can be defined to be invoked either before or after the data is changed by [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx), [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx)or [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx)statements. MySQL allows you to define maximum six triggers for each table.

* BEFORE INSERT – activated before data is inserted into the table.
* AFTER INSERT- activated after data is inserted into the table.
* BEFORE UPDATE – activated before data in the table is updated.
* AFTER UPDATE - activated after data in the table is updated.
* BEFORE DELETE – activated before data is removed from the table.
* AFTER DELETE – activated after data is removed from the table.

When you use a statement that makes change to the table but does not use INSERT, DELETEor UPDATE statement, the trigger is not invoked. For example, the TRUNCATE statement removes the whole data of a table but does not invoke the trigger associated with that table.

There are some statements that use the INSERT statement behind the scenes such as REPLACEstatement and LOAD DATA statement. If you use these statements, the corresponding triggers associated with the tables if available will be invoked.

Triggers defined for a table must have a unique name. You can have the same trigger name that defines for different tables but it is not recommended. In practice, the names of triggers follow the following naming convention:



|  |  |
| --- | --- |
| 1 | (BEFORE | AFTER)\_tableName\_(INSERT| UPDATE | DELETE) |

MySQL Triggers Storage

MySQL stores triggers in a data directory e.g., /data/classicmodels/ with the files namedtablename.TRG and triggername.TRN:

* The tablename.TRG file maps the trigger to the corresponding table.
* the triggername.TRN file contains the trigger definition.

You can back up the MySQL triggers by copying the trigger files to the backup folder. You can also backup the triggers using the *mysqldump*tool*.*

MySQL Trigger Limitations

MySQL triggers have all features in standard SQL however there are some limitations that you should know before using them in your applications.

MySQL triggers cannot:

* Use SHOW, LOAD DATA, LOAD TABLE, BACKUP DATABASE, RESTORE, FLUSH andRETURN statements.
* Use statements that commit or rollback implicitly or explicitly such as COMMIT,ROLLBACK, START TRANSACTION, LOCK/UNLOCK TABLES, ALTER, CREATE, DROP,RENAME, etc.
* Use [prepared statements](http://www.mysqltutorial.org/mysql-prepared-statement.aspx) such as PREPARE, EXECUTE, etc.
* Use dynamic SQL statements.
* Call a [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) or stored function.

In this tutorial, we have shown you how triggers are implemented in MySQL. We also discussed about trigger’s storage as well as trigger’s limitations in MySQL..

Create Trigger in MySQL

**Summary**: in this tutorial, you will learn how to create **trigger in MySQL** by using the **CREATE TRIGGER** statement.

You should follow the [introduction to SQL triggers](http://www.mysqltutorial.org/sql-triggers.aspx) and t[rigger implementation in MySQL](http://www.mysqltutorial.org/mysql-trigger-implementation.aspx)first before going forward with this tutorial.

MySQL trigger syntax

In order to create a trigger you use the CREATE TRIGGER statement. The following illustrates the syntax of the CREATE TRIGGER statement:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE TRIGGER trigger\_name trigger\_time trigger\_event  ON table\_name  FOR EACH ROW  BEGIN  ...  END |

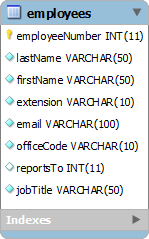
Let’s examine the syntax above in more detail.

* You put the trigger name after the CREATE TRIGGER statement. The trigger name should follow the naming convention [trigger time]\_[table name]\_[trigger event], for example before\_employees\_update*.*
* Trigger activation time can be BEFORE or AFTER. You must specify the activation time when you define a trigger. You use BEFORE keyword if you want to process action prior to the change is made on the table and AFTER if you need to process action after the change is made.
* Trigger event can be INSERT, UPDATE or DELETE. This event causes trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, you have to define multiple triggers, one for each event.
* A trigger must be associated with a specific table. Without a table trigger would not exist therefore you have to specify the table name after the ON keyword.
* The SQL statements are placed between BEGIN and END block.
* The OLD and NEW keywords are very handy. The OLD keyword refers to the existing record before you change the data and the NEW keyword refers to the new row after you change the data.

MySQL trigger example

Let’s start creating a trigger in MySQL to audit the changes of the employees table.

First, we have employees table in our [MySQL sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx) as follows:

[](http://www.mysqltutorial.org/wp-content/uploads/2009/12/employees-table.png?44591f)

Second, we create a new table named employees\_audit to keep the changes of the employee records. The following script creates the employee\_audit table.



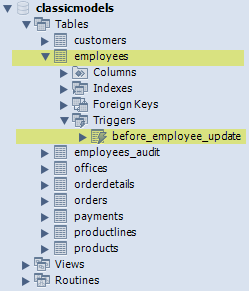
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE employees\_audit (      id int(11) NOT NULL AUTO\_INCREMENT,      employeeNumber int(11) NOT NULL,      lastname varchar(50) NOT NULL,      changedon datetime DEFAULT NULL,      action varchar(50) DEFAULT NULL,      PRIMARY KEY (id)  ) |

Third, we create a BEFORE UPDATE trigger to be invoked before a change is made to theemployees table.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | DELIMITER $$  CREATE TRIGGER before\_employee\_update      BEFORE UPDATE ON employees      FOR EACH ROW BEGIN        INSERT INTO employees\_audit      SET action = 'update',           employeeNumber = OLD.employeeNumber,          lastname = OLD.lastname,          changedon = NOW();  END$$  DELIMITER ; |

If you take a look at the schema, you will see before\_employee\_update trigger under theemployees table as follows:

[](http://www.mysqltutorial.org/wp-content/uploads/2009/12/MySQL-Trigger-Example.png?44591f)

Now it’s time to update an employee record to test if the trigger is really invoked.



|  |  |
| --- | --- |
| 1  2  3 | UPDATE employees  SET lastName = 'Phan'  WHERE employeeNumber = 1056 |

To check if the trigger was invoked by the UPDATE statement, we can query theemployees\_audit table by using the following query:



|  |  |
| --- | --- |
| 1  2 | SELECT \*  FROM employees\_audit |

The following is the output of the query:

[MySQL Trigger Audit Table](http://www.mysqltutorial.org/wp-content/uploads/2009/12/MySQL-Trigger-Audit-Table.png?44591f)

As you see, our trigger was really invoked so that we have a new record in theemployees\_audit table.

In this tutorial, you have learned how to create a trigger in MySQL. We also shown you how to develop a trigger to audit the changes of the employees table.

Managing Trigger in MySQL

**Summary**: in this tutorial, you will learn how to manage triggers including listing, modifying and removing triggers in MySQL databases.

After creating a trigger, you can display its definition in the data folder, which contains trigger definition file. A trigger is stored as plain text file in the following database folder:



|  |  |
| --- | --- |
| 1 | /data\_folder/database\_name/table\_name.trg |

MySQL provides you with an alternative way to display the trigger by using the following SQL statement:



|  |  |
| --- | --- |
| 1  2  3 | SELECT \* FROM Information\_Schema.Triggers  WHERE Trigger\_schema = 'database\_name' AND        Trigger\_name = 'trigger\_name'; |

The statement allows you to view both content of the trigger and its metadata such as associated table name and definer (name of MySQL user who created the trigger).

If you want to retrieve all triggers in a particular database, just executing the following SQL statement:



|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Information\_Schema.Triggers  WHERE Trigger\_schema = 'database\_name'; |

To find all triggers associated with a particular table, you use the following query:



|  |  |
| --- | --- |
| 1  2  3 | SELECT \* FROM Information\_Schema.Triggers  WHERE Trigger\_schema = 'database\_name' AND        Event\_object\_table = 'table\_name'; |

MySQL Drop Trigger

In MySQL, you can not only to view the trigger but also remove an existing one. To remove a trigger, you use DROP TRIGGER statement as follows:



|  |  |
| --- | --- |
| 1 | DROP TRIGGER table\_name.trigger\_name |

For example, if you want to remove *before\_employees\_update* trigger associated with the table*employees,*you can perform the following query:



|  |  |
| --- | --- |
| 1 | DROP TRIGGER employees.before\_employees\_update |

To modify a trigger, you have to delete it first and recreate it with the new code. There is no such ALTER TRIGGER statement exists in MySQL, and you cannot modify an existing trigger like modifying other database objects such as tables and stored procedures.

In this tutorial, you have learned how to manage triggers in MySQL. You have also learned how to remove an existing trigger using the DROP TRIGGER statement.

Working with MySQL Scheduled Event

In this tutorial, you will learn about MySQL event scheduler and how to create MySQL events to automate database tasks.

A MySQL event is a task that runs based on a predefined schedule therefore sometimes it is referred to as a scheduled event. MySQL event is also known as “temporal trigger” because it is triggered by time, not by table update like a [trigger](http://www.mysqltutorial.org/mysql-triggers.aspx). A MySQL event is similar to a cron job in UNIX or a task scheduler in Windows.

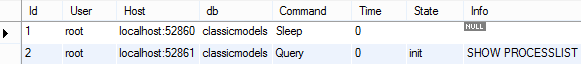
You can use MySQL events in many cases such as optimizing database tables, cleaning up logs, archiving data, or generate complex reports during off-peak time.

MySQL event scheduler configuration

MySQL uses a special thread called event schedule thread to execute all scheduled events. You can see the status of event scheduler thread by executing the following command:



|  |  |
| --- | --- |
| 1 | SHOW PROCESSLIST; |



By default, the event scheduler thread is not enabled. To enable and start the event scheduler thread, you need to execute the following command:

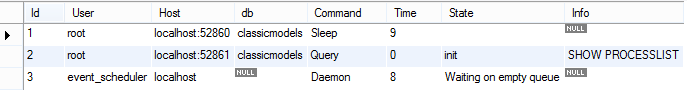


|  |  |
| --- | --- |
| 1 | SET GLOBAL event\_scheduler = ON; |

Now to see the status of event scheduler thread, you execute the  SHOW PROCESSLISTcommand again.



|  |  |
| --- | --- |
| 1 | SHOW PROCESSLIST; |



To disable and stop the event the event scheduler thread, you execute the SET GLOBAL command with value of the event\_scheduler is OFF:



|  |  |
| --- | --- |
| 1 | SET GLOBAL event\_scheduler = OFF; |

 Creating new MySQL events

Creating an event is similar to creating other database objects such as stored procedures or triggers. An event is a named object that contains SQL statements.

A [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) is only executed when it is invoked directly; a [trigger](http://www.mysqltutorial.org/mysql-triggers.aspx)is executed when an event associated with a table such as [insert](http://www.mysqltutorial.org/mysql-insert-statement.aspx), [update](http://www.mysqltutorial.org/mysql-update-data.aspx), or [delete](http://www.mysqltutorial.org/mysql-delete-statement.aspx)event occurs, while an event can be executed at once or more regular intervals.

To create and schedule a new event, you use the  CREATE EVENT statement as follows:



|  |  |
| --- | --- |
| 1  2  3  4 | CREATE EVENT [IF NOT EXIST]  event\_name  ON SCHEDULE schedule  DO  event\_body |

Let’s examine the statement in more detail.

* First, you specify the event name after the  CREATE EVENT clause. The event name must be unique within a database schema.
* Second, you put a schedule after the  ON SCHEDULE clause. If the event is a one-time event, you use the syntax: AT timestamp [+ INTERVAL]. If the event is a recurring event, you use the EVERY clause: EVERY interval STARTS timestamp [+INTERVAL] ENDS timestamp [+INTERVAL].
* Third, you place the SQL statements after the DO keyword. It is important to notice that you can call a stored procedure inside the body of the event. In case you have compound SQL statements, you can wrap them in a  BEGIN END block.

Let’s look at few examples of creating events to understand the syntax above.

To create and schedule a new one-time event that inserts a message into a table called messages you do the following steps:.

First, create a new table named messages by using the  CREATE TABLE statement as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE IF NOT EXISTS messages (      id INT PRIMARY KEY AUTO\_INCREMENT,      message VARCHAR(255) NOT NULL,      created\_at DATETIME NOT NULL  ); |

Second, create an event by using the  CREATE EVENT statement:



|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE EVENT IF NOT EXISTS test\_event\_01  ON SCHEDULE AT CURRENT\_TIMESTAMP  DO    INSERT INTO messages(message,created\_at)    VALUES('Test MySQL Event 1',NOW()); |

Third, check the messages table; you will see that we have 1 record. It means the event was executed when it is created.



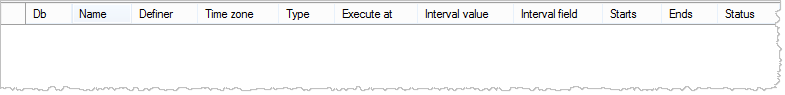
|  |  |
| --- | --- |
| 1 | SELECT \* FROM messages; |

mysql event log entry

To shows all events of a database schema, you use the following statement:



|  |  |
| --- | --- |
| 1 | SHOW EVENTS FROM classicmodels; |



We don’t see any row returned because an event is automatically dropped when it is expired. In our case, it is one-time event and expired when its execution completed.

To change this behavior, you can use the  ON COMPLETION PRESERVE clause. The following statement creates another one-time event that is executed after its creation time 1 minute and not dropped after execution.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE EVENT test\_event\_02  ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 1 MINUTE  ON COMPLETION PRESERVE  DO     INSERT INTO messages(message,created\_at)     VALUES('Test MySQL Event 2',NOW()); |

Wait for 1 minute, check the messages table, another record was added:



|  |  |
| --- | --- |
| 1 | SELECT \* FROM messages; |

message table records

If we execute the  SHOW EVENTS statement again, we see the event is there because the effect of the  ON COMPLETION PRESERVE clause:



|  |  |
| --- | --- |
| 1 | SHOW EVENTS FROM classicmodels; |

mysql event list

The following statement creates a recurring event that executes every minute and is expired in 1 hour from its creation time:

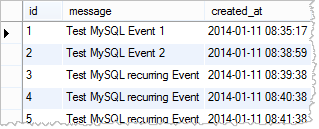


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE EVENT test\_event\_03  ON SCHEDULE EVERY 1 MINUTE  STARTS CURRENT\_TIMESTAMP  ENDS CURRENT\_TIMESTAMP + INTERVAL 1 HOUR  DO     INSERT INTO messages(message,created\_at)     VALUES('Test MySQL recurring Event',NOW()); |

Notice that we used STARTS and ENDS clauses to define expiration period for the event. You can test this recurring event by waiting for few minutes and check the messages table.



|  |  |
| --- | --- |
| 1 | SELECT \* FROM messages; |



Drop MySQL events

To remove an existing event, you use the  DROP EVENT statement as follows:



|  |  |
| --- | --- |
| 1 | DROP EVENT [IF EXIST] event\_name; |

For example, to drop the  test\_event\_03 event, you use the following statement:



|  |  |
| --- | --- |
| 1 | DROP EVENT [IF EXIST] test\_event\_03; |

In this tutorial, you have learned about MySQL events, how to create and delete events from a database schema. In the next tutorial, we will show you how to change existing event.

Modifying MySQL Events

This tutorial shows you how modify existing [MySQL event](http://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/)s by using the  ALTER EVENTstatement. After the tutorial, you will know how to modify schedule of an event, how to enable or disable an event, and how to rename an event.

MySQL allows you to change various attributes of an existing event. To change existing events, you use the  ALTER EVENT statement as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | ALTER EVENT event\_name  ON SCHEDULE schedule  ON COMPLETION [NOT] PRESERVE  RENAME TO new\_event\_name  ENABLE | DISABLE  DO    event\_body |

Notice that the ALTER EVENT statement is only applied to an existing event. If you try to modify a nonexistent event, MySQL will issue an error message therefore you should always use the  SHOW EVENTS statement to check the event for its existence before changing it.



|  |  |
| --- | --- |
| 1 | SHOW EVENTS FROM classicmodels; |

ALTER EVENT examples

Let’s create a sample event to demonstrate various features of the  ALTER EVENT statement.

The following statement creates an event that inserts a new record every minute into themessages table.



|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE EVENT test\_event\_04  ON SCHEDULE EVERY 1 MINUTE  DO     INSERT INTO messages(message,created\_at)     VALUES('Test ALTER EVENT statement',NOW()); |

Changing schedules

To make the event run every 2 minutes, you use the following statement:



|  |  |
| --- | --- |
| 1  2 | ALTER EVENT test\_event\_04  ON SCHEDULE EVERY 2 MINUTE; |

Changing event body

You can also change the event’s body by specifying the new logic as follows:



|  |  |
| --- | --- |
| 1  2  3  4 | ALTER EVENT test\_event\_04  DO     INSERT INTO messages(message,created\_at)     VALUES('Message from event',NOW()); |

You can wait for 2 minutes and check the messages table again:



|  |  |
| --- | --- |
| 1 | SELECT \* FROM messages; |

Disable events

To disable the event, you use the following statement:



|  |  |
| --- | --- |
| 1  2 | ALTER EVENT test\_event\_04  DISABLE; |

You can check the status of the event by using the  SHOW EVENTS statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW EVENTS FROM classicmodels; |

Enable events

To enable a disabled event, you use the ENABLE keyword after the  ALTER EVENT statement as follows:



|  |  |
| --- | --- |
| 1  2 | ALTER EVENT test\_event\_04  ENABLE; |

Rename events

MySQL does not provide you with the  RENAME EVENT statement. Fortunately, you can use the ALTER EVENT to rename an existing event as follows:



|  |  |
| --- | --- |
| 1  2 | ALTER EVENT test\_event\_04  RENAME TO test\_event\_05; |

Move events to another database

You can move an event from a database to another database by using the  RENAME TO clause as follows:



|  |  |
| --- | --- |
| 1  2 | ALTER EVENT classicmodels.test\_event\_05  RENAME TO newdb.test\_event\_05 |

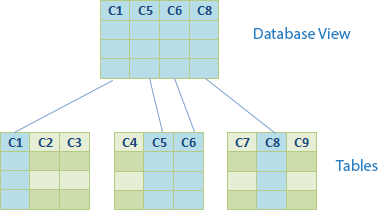
It is assumed that the newdb database is available in your MySQL database server.

In this tutorial, we have shown you how to change various attributes of a MySQL event by using the  ALTER EVENT statement.

PART 4: MySQL Views

Introduction to Database View

**Summary**: in this tutorial, you will learn about a new database object called **database view**. We will discuss about the advantages and disadvantages of using database views.

A database view is a virtual table or logical table which is defined as a [SQL SELECT query](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) with [joins](http://www.mysqltutorial.org/mysql-inner-join.aspx). Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it. Most database management systems, including MySQL, allows you to update data in the underlying tables through the database view with some prerequisites.

A database view is dynamic because it is not related to the physical schema. The database system stores database views as a [SQL SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement with joins. When the data of the tables changes, the view reflects that changes as well.

Advantages of database view

The following are advantages of using database views.

* A database view allows you to simplify complex queries: a database view is defined by an SQL statement that associates with many underlying tables. You can use database view to hide the complexity of underlying tables to the end-users and external applications. Through a database view, you only have to use simple SQL statements instead of complex ones with many joins.
* A database view helps limit data access to specific users. You may not want a subset of sensitive data can be queryable by all users. You can use database views to expose only non-sensitive data to a specific group of users.
* A database view provides extra security layer. Security is a vital part of any relational database management system. Database views provides extra security for a database management system. A database view allows you to create only read-only view to expose read-only data to specific users. Users can only retrieve data in read-only view but cannot update it.
* A database view enables computed columns. A database table should not have calculated columns however a database view should. Suppose in the orderDetails table you have quantityOrder (the number of ordered products) and priceEach (price per product item) columns. However the orderDetails table does not have computed column to store total sales for each line item of the order. If it has, the database schema would not be a good design. In this case, you can create a computed column namedtotal*,*which is a product of quantityOrder and priceEach to store the computed result. When you query data from the database view, the data of the computed column is calculated on fly.
* Database view enables backward compatibility. Suppose you have a central database, which many applications are using it. One day you decided to redesign the database to adapt with the new business requirements. You remove some tables and create several new tables, and you don’t want the changes affect other applications. In this scenario, you can create database views with the same schema as the legacy tables that you’ve removed.

Disadvantages of database view

Besides the advantages above, there are several disadvantages of using database views:

* Performance: querying data from a database view can be slow especially if the view is created based on other views.
* Tables dependency: you create view based on underlying tables of the a database. Whenever you change the structure of those tables that view associates with, you have to change the view as well.

In this tutorial, you have learned what a database view is. We also discussed about the advantages and disadvantages of using database views so that you can apply them effectively in your database design.

Views in MySQL

**Summary**: in this tutorial, you are going to learn about **MySQL View**. We will explain how MySQL implements views.

MySQL supports database views or views since version 5.X. In MySQL, almost features of views conform to the SQL: 2003 standard. MySQL process queries to the views in two ways:

* MySQL creates a temporary table based on the view definition statement and then executes the incoming query on this temporary table.
* First, MySQL combines the incoming query with the query defined the view into one query. Then, MySQL executes the combined query.

MySQL supports version system for views. Each time when the view is altered or replaced, a copy of the existing view is back up in arc (archive) folder which resides in a specific database folder. The name of back up file is view\_name.frm-00001. If you then change the view again, MySQL will create a new backup file named view\_name.frm-00002.

MySQL also allows you to create a view of views. In the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) of view definition, you can refer to another views.

If the cache is enabled, the query against a view is stored in the cache. As the result, it increases the performance of the query by pulling data from the cache instead of querying data from the underlying tables.

Creating Views in MySQL

**Summary**: in this tutorial, you will learn how to **create views in MySQL** by using the CREATE VIEW statement.

Introducing to CREATE VIEW statement

The syntax of creating a view in MySQL is as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE     [ALGORITHM = {MERGE  | TEMPTABLE | UNDEFINED}]  VIEW [database\_name].[view\_name]  AS  [SELECT  statement] |

Algorithms

The algorithm attribute allows you to control which mechanism is used when creating a view. MySQL provides the MERGE, TEMPTABLE and UNDEFINED algorithms.

* MERGE means the input query will be combined with the [SELECT statement](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) of the view definition. MySQL will execute the combined query to return the result set. This mechanism is more efficient than TEMPTABLE (temporary table) but MERGE only allowed when the rows in the view represent a one-to-one relationship with the rows in the underlying table. In case the MERGE is not allowed, MySQL will switch the algorithm toUNDEFINED. The combination of input query and query in view definition into one query sometimes refers as view resolution.
* TEMPTABLE means that MySQL first creates a temporary table based on the SELECTstatement of the view definition, and then it executes the input query against this temporary table. Because MySQL has to create temporary table to store the result set and move the data from the physical tables to the temporary table, the TEMPTABLE algorithmis less efficient than the MERGE algorithm. In addition, a view that usesTEMPTABLE algorithm is not [updateable](http://www.mysqltutorial.org/create-sql-updatable-views.aspx).
* UNDEFINED is the default algorithm when you create a view without specifying an explicit algorithm. The UNDEFINED algorithm allows MySQL to make a decision whether to use MERGE or TEMPTABLE. MySQL prefers MERGE to TEMPTABLE, which is more efficient.

View name

Each view is associated with a specific database therefore you can have database name prefix with the view name. Names of views share the same domain with tables therefore they cannot be the same names as tables in a database.

SELECT statement

In the SELECT statement, you can query data from any table or view that exists in the database. There are several rules that the SELECT statement must follow:

* The SELECT statement can contain a [subquery](http://www.mysqltutorial.org/mysql-subquery/)in [WHERE clause](http://www.mysqltutorial.org/mysql-where/) but not in the FROMclause.
* The SELECT statement cannot refer to any variable including local variable, user variable or session variable.
* The SELECT statement cannot refer to the parameters of [prepared statements](http://www.mysqltutorial.org/mysql-prepared-statement.aspx).

MySQL create view examples

Create a simple view

Let’s take a look at the orderDetails table. We can create a view that represents total sales per order.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE VIEW SalePerOrder     AS    SELECT orderNumber,    SUM  (quantityOrdered \* priceEach) total    FROM orderDetails    GROUP by orderNumber    ORDER BY total DESC |

If you want to query total sales for each sales order, you just need to execute a simple SELECTstatement against the SalePerOrder view as follows:



|  |  |
| --- | --- |
| 1  2  3 | SELECT total  FROM salePerOrder  WHERE orderNumber = 10102 |

Create view with JOIN

The following is an example of creating a view with an [INNER JOIN statement](http://www.mysqltutorial.org/mysql-inner-join.aspx). The view contains *order number*, *customer name* and *total sales*per order.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE VIEW customerOrders AS  SELECT  D.orderNumber,           customerName,           SUM(quantityOrdered \* priceEach) total  FROM orderDetails D  INNER JOIN orders O ON O.orderNumber = D.orderNumber  INNER JOIN customers C ON O.customerNumber =  C.customerNumber  GROUP BY D.orderNumber  ORDER BY total DESC |

Create view with subquery

The following illustrates how to create a view with [subquery](http://www.mysqltutorial.org/mysql-subquery/). The view contains products whose buy prices are higher than average price of all products.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE VIEW vwProducts  AS  SELECT productCode,          productName,          buyPrice  FROM products  WHERE buyPrice > (        SELECT AVG  (buyPrice)        FROM  products  )  ORDER BY buyPrice DESC |

In this tutorial, we have shown you how to create views by using the CREATE VIEW statement.

Managing Views in MySQL

**Summary**: in this tutorial, you will learn how to manage views in MySQL including displaying, modifying and removing views.

Show view definition in MySQL

MySQL provides the SHOW CREATE VIEW statement that helps you show view definition. The following is the syntax of the SHOW CREATE VIEW statement:



|  |  |
| --- | --- |
| 1 | SHOW CREATE VIEW [database\_name].[view\_ name]; |

To display the definition of a view, you just need to specify its name after the SHOW CREATE VIEW keywords.

Let’s [create a view](http://www.mysqltutorial.org/create-sql-views-mysql.aspx) for the demonstration.

First, we create a simple view against the employees table that displays the company’s organization structure:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE VIEW organization     AS      SELECT CONCAT (E.lastname,E.firstname) AS Employee,             CONCAT  (M.lastname,M.firstname) AS Manager      FROM employees AS E      INNER JOIN employees AS M          ON M.employeeNumber = E.ReportsTo      ORDER BY Manager |

To display the view’s definition, you use the SHOW CREATE VIEW statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW CREATE VIEW organization |

You can also display the definition of the view by using any plain text editor such as notepad to open the view definition file in the database folder.

For example, to open the organization view definition, you can find the view definition file with the following path: \data\classicmodels\organization.frm

Modifying views

Once a view is defined, you can modify it by using the ALTER VIEW statement. The syntax of the ALTER VIEW statement is similar to the CREATE VIEW statement except the CREATEkeyword is replaced by the ALTER keyword.



|  |  |
| --- | --- |
| 1  2  3  4  5 | ALTER  [ALGORITHM =  {MERGE | TEMPTABLE | UNDEFINED}]    VIEW [database\_name].  [view\_name]     AS  [SELECT  statement] |

The following query modifies the organization view by adding an addition email field*.*



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | ALTER VIEW organization    AS    SELECT CONCAT(E.lastname,E.firstname) AS Employee,           E.email AS  employeeEmail,           CONCAT(M.lastname,M.firstname) AS Manager    FROM employees AS E    INNER JOIN employees AS M      ON M.employeeNumber = E.ReportsTo    ORDER BY Manager |

To verify the change, you can query data from the organization view:



|  |  |
| --- | --- |
| 1 | SELECT \* FROM Organization |

MySQL drop views

Once a view created, you can remove it by using the DROP VIEW statement. The following illustrates the syntax of the DROP VIEW statement:



|  |  |
| --- | --- |
| 1 | DROP VIEW [IF EXISTS] [database\_name].[view\_name] |

The IF EXISTS is the optional element of the statement, which allows you to check whether the view exists or not. It helps you avoid an error of removing a non-existent view.

For example, if you want to remove the organization view*,* you can use the DROP VIEWstatement as follows:



|  |  |
| --- | --- |
| 1 | DROP VIEW IF EXISTS organization |

Each time you modify or remove a view, MySQL makes a back up of the view definition file to the /database\_name/arc/ folder. In case you modify or remove a view by accident, you can get a back up from there.

In this tutorial, you have learned how to manage views in MySQL including displaying, modifying and removing views.

MySQL Functions

This section gives you the most commonly used **MySQL functions** including aggregate functions, string functions, date time functions, control flow functions, etc.

MySQL aggregate functions

* [MySQL aggregate function](http://www.mysqltutorial.org/mysql-aggregate-functions.aspx) - provides a brief overview of the most commonly used MySQL aggregate functions
* [AVG](http://www.mysqltutorial.org/mysql-avg/)- calculates the average value of a set of values or an expression.
* [COUNT](http://www.mysqltutorial.org/mysql-count/)- count the number rows in a table.
* [SUM](http://www.mysqltutorial.org/mysql-sum/)- calculates the sum of a set of values or an expression.
* [MIN](http://www.mysqltutorial.org/mysql-min/)- finds the minimum value in a set of values
* [MAX](http://www.mysqltutorial.org/mysql-max-function/)- finds the maximum value in a set of values
* [GROUP\_CONCAT](http://www.mysqltutorial.org/mysql-group_concat/) – concatenates strings from a group into a string with various options such as DISTINCT, ORDER BY and SEPARATOR.
* [MySQL standard deviation functions](http://www.mysqltutorial.org/mysql-standard-deviation/) – shows you how to compute population standard deviation and sample standard deviation by using functions: STD, STDDEV,STDDEV\_POP and STDDEV\_SAMP. In addition, the population variance and sample variance functions such as VAR\_POP, VARIANCE and VAR\_SAMP are also covered.

MySQL string functions

* [CONCAT](http://www.mysqltutorial.org/sql-concat-in-mysql.aspx)- combines two or more strings into one string.
* [LENGTH and CHAR\_LENGTH](http://www.mysqltutorial.org/mysql-string-length/)- get the length of strings in bytes and in characters.
* [REPLACE](http://www.mysqltutorial.org/mysql-string-replace-function.aspx)- searches and replaces a substring in a string.
* [SUBSTRING](http://www.mysqltutorial.org/mysql-substring.aspx)-  extracts a substring starting from a position with a specific length.

MySQL control flow functions

* [IF](http://www.mysqltutorial.org/mysql-if-function.aspx)- returns a value based on a given condition
* [IFNULL](http://www.mysqltutorial.org/mysql-ifnull/)- returns the first argument if it is not NULL, otherwise returns the second argument.
* [NULLIF](http://www.mysqltutorial.org/mysql-nullif/)- returns NULL if the first argument is equal to the second argument, otherwise returns the first argument.

MySQL date and time functions

* [DATEDIFF](http://www.mysqltutorial.org/mysql-datediff.aspx)- calculates the number of days between two DATE or DATETIME values.
* [DATE\_FORMAT](http://www.mysqltutorial.org/mysql-date_format/) – formats a date value based on a specified date format.
* [STR\_TO\_DATE](http://www.mysqltutorial.org/mysql-str_to_date/)- converts a string into a date and time value based on a specified format.
* [NOW](http://www.mysqltutorial.org/mysql-now/)- returns the current date and time at which the statement executed.

Other MySQL functions

* [LAST\_INSERT\_ID](http://www.mysqltutorial.org/mysql-last_insert_id.aspx) - obtains the last generated sequence number of the last inserted record.
* [CAST](http://www.mysqltutorial.org/mysql-cast/) – converts a value of any type into a value with a specified type.

**Share this:**

MySQL Administration

In this section, you will find a lot of useful MySQL administration tutorials including MySQL server startup and shutdown, MySQL server security, MySQL database maintenance, backup and replication.

[Getting Started with MySQL Access Control System](http://www.mysqltutorial.org/getting-started-with-mysql-access-control-system.aspx)

[](http://www.mysqltutorial.org/getting-started-with-mysql-access-control-system.aspx)

MySQL implemented a sophisticated access control and privilege system to allow you to create comprehensive access rules for handling client operations and prevent unauthorized clients from accessing the database system.

[MySQL Create User](http://www.mysqltutorial.org/mysql-create-user.aspx)

[](http://www.mysqltutorial.org/mysql-create-user.aspx)

In this tutorial, you will learn how to create user in MySQL by using CREATE USER and INSERT statements.

[MySQL Changing Password for Accounts](http://www.mysqltutorial.org/mysql-changing-password.aspx)

[](http://www.mysqltutorial.org/mysql-changing-password.aspx)

In this tutorial you will learn how to change or reset password for MySQL account in various ways by using the UPDATE, SET PASSWORD and GRANT statements.

[How to Use MySQL GRANT to Grant Privileges to Account](http://www.mysqltutorial.org/mysql-grant.aspx)

[](http://www.mysqltutorial.org/mysql-grant.aspx)

In this tutorial, you will learn how to use MySQL GRANT statement to grant privileges to accounts.

[Revoking Privileges from Users Using MySQL REVOKE](http://www.mysqltutorial.org/mysql-revoke.aspx)

[Revoking Privileges from Users Using MySQL REVOKE](http://www.mysqltutorial.org/mysql-revoke.aspx)

In this tutorial, you will learn how to use MySQL REVOKE statement to revoke privileges from MySQL accounts.

[Maintaining MySQL Database Tables](http://www.mysqltutorial.org/mysql-database-table-maintenance-statements.aspx)

[](http://www.mysqltutorial.org/mysql-database-table-maintenance-statements.aspx)

MySQL provides several useful statements that allows you to maintain database tables to increase the efficiency of table access. Those statements includes analyze, optimize, check, and repair tables.

[How to Backup Databases Using mysqldump Tool](http://www.mysqltutorial.org/how-to-backup-database-using-mysqldump.aspx)

In this tutorial you will learn how to use mysqldump tool back up MySQL databases.

Getting Started with MySQL Access Control System

**Summary**: in this tutorial, we will introduce you to **MySQL Access Control System** and various privilege-related tables in MySQL.

MySQL implements a sophisticated access control and privilege system that allows you to create comprehensive access rules for handling client operations and effectively preventing unauthorized clients from accessing the database system.

The MySQL access control has two stages when a client connects to the server:

* **Connection verification**: a client, which connects to the MySQL database server, needs to have a valid username and password. In addition, the host from which the client connects has to match with the host in the MySQL grant table.
* **Request verification**: once a connection is established successfully, for each statement issued by the client, MySQL checks whether the client has sufficient privileges to execute that particular statement. MySQL has ability to check a privilege at database, table, and field level.

There is a database named mysql created automatically by MySQL installer. The mysqldatabase contains five main grant tables. You often manipulate these tables indirectly through the statements such as [GRANT](http://www.mysqltutorial.org/mysql-grant.aspx)and [REVOKE](http://www.mysqltutorial.org/mysql-revoke.aspx).

* user: contains user account and global privileges columns. MySQL uses the user table to either accept or reject a connection from a host. A privilege granted in user table is effective to all databases on the MySQL server.
* db: contains database level privileges. MySQL uses the db table to determine which database a user can access and from which host. A privilege granted at database level in the db table applies to the database and all objects belong to that database e.g., tables,[triggers](http://www.mysqltutorial.org/mysql-triggers.aspx" \o "MySQL Triggers), [views](http://www.mysqltutorial.org/mysql-views-tutorial.aspx), [stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), etc.
* table\_priv and columns\_priv: contains table-level and column-level privileges. A privilege granted in the table\_priv table applies to the table and its columns while a privilege granted in columns\_priv table applies only to a specific column of a table.
* procs\_priv: contains stored functions and [stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) privileges

MySQL makes use of those tables to control the privileges in MySQL database server. Understanding those table is very important before you can implement your own flexible access control system.

In this tutorial, you have learned how the MySQL access control system works and explored the grant tables in MySQL.

MySQL Create User

**Summary:** in this tutorial, you will learn how to **create user** in MySQL by using the CREATE USER and INSERT statements.

In MySQL, you can specify not only who can connect to a database server but also from where. Therefore an account in MySQL consists of username and host name where user connects from, that separated by @. For example, if the root user connects from themysqltutorial.org host to the database server the account name would beroot@mysqltutorial.org.

This allows you to setup multiple accounts with the same name but connects from different hosts and have different privileges. In addition, you have control to setup account to connect from either specific or broad a set of hosts. The username and host are stored in the grant table named  user*.*

Create new users using CREATE USER statement

MySQL provides the CREATE USER statement to allow you to create a new user in a database server. The syntax of the CREATE USER statement is as follows:



|  |  |
| --- | --- |
| 1 | CREATE USER user IDENTIFIED BY password |

User account in the format 'username'@'hostname' is followed by the CREATE USER.

The password is specified after the IDENTIFIED BY clause. The password must be in plain text. MySQL will encrypt the password for you when it saves the user into the user table.

For example, to create a new user dbadmin that connects from localhost with the password CrEate-User , you use the CREATE USER statement as follows:



|  |  |
| --- | --- |
| 1  2 | CREATE USER dbadmin@localhost  IDENTIFIED BY 'CrEate-User' |

To allow user to connect from any host you use the % wildcard, which means any host.



|  |  |
| --- | --- |
| 1  2 | CREATE USER superadmin@'%'  IDENTIFIED BY 'Secured' |

The percentage wildcard  %  has same effect as it is used in the [LIKE](http://www.mysqltutorial.org/mysql-like) operator e.g., to allowmysqladmin user to connect from any host to the  mysqltutorial.org you use percentage wildcard % as follows:



|  |  |
| --- | --- |
| 1  2 | CREATE USER mysqladmin@'%.mysqltutorial.org'  IDENTIFIED by 'SecurePass704'; |

Notice that you can also use the underscrore wildcard \_  in the CREATE USER statement.

If you omit the hostname part of the account, MySQL still accepts it and allows user to connect from any host. The quote is very important especially when the account contains special characters such as - or %.

If you accidentally quote the account name like 'username@hostname', MySQL will create a user with username@hostname username and allow the user to connect from any host, which may not be what you expected.

The MySQL CREATE USER statement only creates a new user and does not grant any privilege to that user. If you want to [grant privileges to the user](http://www.mysqltutorial.org/mysql-grant.aspx) you use the MySQL GRANT statement.

Create new user by using INSERT statement

A less common way to create user is using [INSERT statement](http://www.mysqltutorial.org/mysql-insert-statement.aspx) to insert a new record into the grant table named  user*.*By doing this way, you need to use the PASSWORD function to encrypt the password before inserting the user record into the table. See the following example:



|  |  |
| --- | --- |
| 1  2 | INSERT INTO user (host,user,password)  VALUES('localhost','dbadmin',PASSWORD('CrEate-User')); |

In this tutorial, you have learned various ways to create user in MySQL using the  CREATE USER statement and INSERT statement.

MySQL Changing Password for Accounts

**Summary**: in this tutorial you will learn how to change or reset password for MySQL account in various ways by using the UPDATE, SET PASSWORD and GRANT statements.

Before changing the password, there are some points that you should consider:

* Which user do you want to change the password?
* What host, which the user connects to, you want to change the password?
* Have you been aware of the related applications that are currently using the MySQL account whose password is being changed?If you change the password without proper planning, those applications will lose the connection to the database and stop functioning properly.

MySQL provides several statements that you can use to change or reset password of a MySQL account including [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx)statement, SET PASSWORD statement and GRANT USAGE statement.

MySQL changing password using UPDATE statement

The first way to change the password is to use the UPDATE statement to update the Userand Host column of the user table in the mysql database. After executing the UPDATEstatement, you need to execute the FLUSH PRIVILEGES statement to reload privileges from the grant table in the mysql database.

Suppose you want to change the password for mysqltutorialuser that connect from the mysqltutorial.orghost to Secret1970, you need to execute the following statements:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | USE mysql;    UPDATE user  SET password = PASSWORD('Secret1970')  WHERE user = 'mysqltutorial' AND        host = 'mysqltutorial.org';    FLUSH PRIVILEGES; |

It is important to note that we’ve used the PASSWORD() function to encrypt the plain text password. The following example demonstrates how the PASSWORD() function encrypts a plain text password.



|  |  |
| --- | --- |
| 1 | SELECT PASSWORD('Secret1970') AS encrypt\_password |



|  |  |
| --- | --- |
| 1 | \*16A54B5EE15C823362FAE7F64409A8F8C8501DD6 |

MySQL changing password using SET PASSWORD statement

The second way to change the password is by using the SET PASSWORD statement. You use the MySQL account in  user@host format to update the password. If you need to change password for other accounts, you need to have at least UPDATE privilege. You don’t need to execute theFLUSH PRVILILEGES statement to reload privileges from grant table.

The following statement change password of mysqltutorial account by using the SET PASSWORD statement.



|  |  |
| --- | --- |
| 1 | SET PASSWORD FOR 'mysqltutorail'@'mysqltutorial.org' = PASSWORD('Secret1970') |

MySQL changing password using GRANT USAGE statement

The third way to change the password is by using the GRANT USAGE statement with the IDENTIFIED BY clause. In this way, you specify the password in plain text instead of the encrypted from.

The following statement changes the password of mysqltutorial account by using theGRANT USAGE statement:



|  |  |
| --- | --- |
| 1 | GRANT USAGE ON \*.\* TO mysqltutorial@mysqltutorial.org IDENTIFIED BY Secret1970 |

In case you want to reset the MySQL root account’s password, you need to force the MySQL database server to stop and restart without using grant table validation.

In this tutorial, we introduced you to some handy MySQL statements that allows you to changeMySQL accounts password.

How to Use MySQL GRANT to Grant Privileges to Account

**Summary**: in this tutorial, you will learn how to use **MySQL GRANT**statement to grant privileges to MySQL accounts.

To going forward with this tutorial, we are highly recommend that you follow the [Getting started with MySQL access control system](http://www.mysqltutorial.org/getting-started-with-mysql-access-control-system.aspx) and [How to create a user in MySQL](http://www.mysqltutorial.org/mysql-create-user.aspx) first.

MySQL GRANT Statement Syntax

MySQL provides you with the MySQL GRANTstatement that allows you to grant access privileges to database accounts. The following illustrates the GRANT statement syntax:



|  |  |
| --- | --- |
| 1  2  3  4  5 | GRANT privileges (column\_list)  ON [object\_type] privilege\_level  TO account [IDENTIFIED BY 'password']  [REQUIRE encryption]  WITH with\_options |

We will examine the MySQL GRANT statement in greater detail:

* privileges indicates the privileges that you assign to the account. For example, theCREATE privilege allows an account to [create databases](http://www.mysqltutorial.org/mysql-create-drop-database.aspx) and [create tables](http://www.mysqltutorial.org/mysql-create-table/). You can grant multiple privileges using single GRANT statement; the privileges are separated by commas.
* column\_list specifies the columns to which a privilege applies. The columns are separated by commas and listed within parentheses. The column\_list is optional element.
* privilege\_level specifies the level at which the privileges apply. You can use global privileges, database-specific privileges, table-specific privileges, column-specific privileges, etc.
* account specifies which account is being granted the privileges.
* password specifies the password to assign to the account*.* If the account exists, theGRANT statement replaces the old password by the new one. Like the CREATE USERstatement, you use plain text password followed by the IDENTIFIED BY clause. TheIDENTIFIED BY clause is optional.
* After the REQUIRE clause, you specifies whether the account has to connect to the database server over secure connection using SSL.
* If you want the account to have the privilege that can grant its own privileges to other accounts, you need to use the WITH clause with GRANT OPTION clause. In addition, you can use the WITH clause to allocate MySQL database server’s resource e.g., to set how many connections or statements that an account can use per hour. This is very helpful in shared environments such as MySQL shared hosting.
* If the account that you specify in the GRANT statement after the TO clause exists, theGRANT statement modifies its privileges, otherwise, the GRANT statement creates a new account with the specified privileges.

Besides granting privileges, the GRANT statement also assigns other characteristics to an account such as:

* Limit account on access to the database server resource.
* Force account to use secure connection when connecting to the database server.

If you want to grant a particular privilege to an account, you must have at least that privilege and GRANT OPTION privilege.

You often use the MySQL GRANT statement together with the [CREATE USER statement](http://www.mysqltutorial.org/mysql-create-user.aspx). You use the CREATE USER statement to create a new account first and then use the MySQL GRANTstatement to grant privileges to the created account.

MySQL GRANT examples

Let’s practice with some examples of using MySQL GRANT statement to have a better understanding.

If you want to create a super account that can do anything including being able to grant privileges to other users, you can use the following statements:



|  |  |
| --- | --- |
| 1  2  3 | CREATE USER 'super'@'localhost' IDENTIFIED BY 'SecurePass1';    GRANT ALL ON \*.\* TO 'super'@'localhost' WITH GRANT OPTION; |

The ON \*.\* clause means all databases and all objects in the databases. The only limitation of the super user is that it can only connect to the database server from the localhost, which makes the MySQL server more secure.

To create a user that has all access in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx) and can connect from any host you use the following statements:



|  |  |
| --- | --- |
| 1  2  3 | CREATE USER 'super2'@'%' IDENTIFIED BY 'SecurePass2';    GRANT ALL classicmodels.\* TO 'super2'@'%' WITH GRANT OPTION; |

You can grant multiple privileges using a single GRANT statement. For example, you can create a user that can execute the [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx)and [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx)statements against theclassicmodels sample database using the following statements:



|  |  |
| --- | --- |
| 1  2  3 | CREATE USER 'rfc'@'%' IDENTIFIED BY 'SecurePass3';    GRANT SELECT, UPDATE, DELETE ON  classicmodels.\* TO 'rfc'@'%'; |

Available privileges to use with MySQL GRANT

The following table illustrates all privileges available in MySQL.

| **Privilege** | **Description** |
| --- | --- |
| ALL [PRIVILEGES] | Grant all privileges at specified access level except GRANT OPTION |
| ALTER | Allow to use of ALTER TABLE statement |
| ALTER ROUTINE | Allow user to alter or drop stored routine |
| CREATE | Allow user to create database and table |
| CREATE ROUTINE | Allow user to create stored routine |
| CREATE TABLESPACE | Allow user to create, alter or drop tablespaces and log file groups |
| CREATE TEMPORARY TABLES | Allow user to create temporary table by using CREATE TEMPORARY TABLE |
| CREATE USER | Allow user to use the CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES statements. |
| CREATE VIEW | Allow user to create or modify view |
| DELETE | Allow user to use DELETE |
| DROP | Allow user to drop database, table and view |
| EVENT | Allow user to schedule events in Event Scheduler |
| EXECUTE | Allow user to execute stored routines |
| FILE | Allow user to read any file in the database directory. |
| GRANT OPTION | Allow user to have privileges to grant or revoke privileges from other accounts |
| INDEX | Allow user to create or remove indexes. |
| INSERT | Allow user to use INSERT statement |
| LOCK TABLES | Allow user to use LOCK TABLES on tables for which you have the SELECT privilege |
| PROCESS | Allow user to see all processes with SHOW PROCESSLIST statement. |
| PROXY | Enable user proxying |
| REFERENCES | Not implemented |
| RELOAD | Allow user to use FLUSH operations |
| REPLICATION CLIENT | Allow user to query to see where master or slave servers are |
| REPLICATION SLAVE | Allow user to use replicate slaves to read binary log events from the master. |
| SELECT | Allow user to use SELECT statement |
| SHOW DATABASES | Allow user to show all databases |
| SHOW VIEW | Allow user to use SHOW CREATE VIEW statement |
| SHUTDOWN | Allow user to use mysqladmin shutdown command |
| SUPER | Allow user to use other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin command |
| TRIGGER | Allow user to use TRIGGER operations. |
| UPDATE | Allow user to use UPDATE statement |
| USAGE | Equivalent to “no privileges” |

*Table 1.1*

In this tutorial, you have learned how to use the MySQL GRANT statement to grant privileges to accounts.

Revoking Privileges from Users Using MySQL REVOKE

**Summary***:*in this tutorial, you will learn how to use **MySQL REVOKE**statement to revoke privileges from MySQL accounts*.*

We are highly recommend that you follow the tutorials below to have a better understanding of how MySQL REVOKE works:

* [Getting Started with MySQL Access Control System](http://www.mysqltutorial.org/getting-started-with-mysql-access-control-system.aspx)
* [MySQL Create User](http://www.mysqltutorial.org/mysql-create-user.aspx)
* [How to Use MySQL GRANT to Grant Privileges to Account](http://www.mysqltutorial.org/mysql-grant.aspx)

MySQL REVOKE Syntax

In order to revoke privileges from an account, you use the MySQL REVOKE statement. The syntax of MySQL REVOKE statement is as follows:



|  |  |
| --- | --- |
| 1  2  3 | REVOKE   privilege\_type [(column\_list)]      [, priv\_type [(column\_list)]]...  ON [object\_type] privilege\_level  FROM user [, user]... |

Let’s examine the MySQL REVOKE statement in more detail.

* You specify a list of privileges that you want to revoke from an account right after the REVOKE keyword. You need to separate privileges by comma.
* ON clause specifies the privilege level at that privileges are to be revoked.
* After FROM keyword, you specify the account that you want to revoke the privileges. You can specify multiple accounts in the FROM clause. You separate the accounts by comma.

In order to revoke privileges from an account, you must have GRANT OPTION privilege and privileges that you are revoking. To revoke all privileges, you use the following MySQL REVOKE syntax:



|  |  |
| --- | --- |
| 1 | REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user]… |

To execute the above command, you must have the global CREATE USER privilege or the UPDATE privilege for the *mysql* database.

To revoke proxy user, you use the REVOKE PROXY command as follows:



|  |  |
| --- | --- |
| 1 | REVOKE PROXY ON user FROM user [, user]... |

A proxy user is a valid user in MySQL who can impersonate as another user therefore the proxy user has all privileges of the user that it impersonates.

Before revoking privileges of a user, it is good practice to check if the user has the privileges by using the SHOW GRANTS statement as follows:



|  |  |
| --- | --- |
| 1 | SHOW GRANTS FOR user; |

MySQL REVOKE examples

Suppose *rfc* account has privileges SELECT, UPDATE and DELETE in the *classicmodels*[sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx) . If you want to revoke UPDATE and DELETE privileges from the *rfc*  account, you can do so as follows:

First, you check the privileges of *rfc* account using SHOW GRANTS statement:



|  |  |
| --- | --- |
| 1 | SHOW GRANTS FOR 'rfc'@'localhost'; |



|  |  |
| --- | --- |
| 1 | GRANT SELECT, UPDATE, DELETE ON 'classicmodels'.\* TO 'rfc'@'localhost' |

If you have not followed the previous tutorial on [granting privileges to user](http://www.mysqltutorial.org/mysql-grant.aspx), you can first grant the SELECT, UPDATE and DELETE privileges for *rfc* account that connects from *localhost* to the *classicmodels* database as follows:



|  |  |
| --- | --- |
| 1 | GRANT SELECT, UPDATE, DELETE ON  classicmodels.\* TO 'rfc'@'localhost'; |

Second, you can revoke the UPDATE and DELETE privileges from the *rfc* account:



|  |  |
| --- | --- |
| 1 | REVOKE UPDATE, DELETE ON classicmodels.\*  FROM 'rfc'@'localhost'; |

Third, you can check the privileges of the *rfc* account again using the SHOW GRANTS command.



|  |  |
| --- | --- |
| 1 | SHOW GRANTS FOR 'rfc'@'localhost'; |



|  |  |
| --- | --- |
| 1 | GRANT SELECT ON 'classicmodels'.\* TO 'rfc'@'localhost' |

If you want to revoke all privileges of the *rfc* account, you run the following command:



|  |  |
| --- | --- |
| 1 | REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'rfc'@'localhost'; |

If you check the privileges of the *rfc* account again, you will see the *rfc* account has no privilege.



|  |  |
| --- | --- |
| 1 | SHOW GRANTS FOR 'rfc'@'localhost'; |



|  |  |
| --- | --- |
| 1 | GRANT USAGE ON \*.\* TO 'rfc'@'localhost' |

Note that USAGE privilege means no privileges in MySQL.

When MySQL REVOKE takes effect

The effect of MySQL REVOKE statement depends on the privilege level as follows:

* Changes that are made to the global privileges only take effect when the client connects to the MySQL in the subsequent sessions. The changes are not applied to all current connected accounts.
* The change of database privileges is applied after the next *USE* statement.
* Table and column privilege’s changes are applied to all queries that are issued after the changes are made.

In this tutorial, you’ve learned how to use MySQL REVOKE statement to revoke privileges from MySQL accounts.

Maintaining MySQL Database Tables

**Summary**: in this tutorial, we will introduce you to some very useful statements that allow you to maintain database tables in MySQL.

MySQL provides several useful statement that allows you to maintain database tableseffectively. Those statements enable you to analyze, optimize, check, and repair database tables.

Analyze table statement

MySQL query optimizer is an important component of the MySQL server that creates an optimal query execution plan for a query. For a particular query, the query optimizer uses the stored key distribution and other factors to decide the order in which tables should be joined when you performing join, and which indexes should be used for a specific table.

However, the key distributions can be sometimes inaccurate e.g., after you have done a lot of data changes in the table including insert, delete or update. If the key distribution is not accurate, the query optimizer may pick a bad query execution plan that may cause a severe performance issue.

To solve this problem, you can run the ANALYZE TABLE statement for the table e.g., the following statement analyze the payments table in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx).



|  |  |
| --- | --- |
| 1 | ANALYZE TABLE payments |

mysql analyze table statement

If there is no change to the table since the ANALYZE TABLE statement ran, MySQL will not analyze the table again. If you run the above statement again:



|  |  |
| --- | --- |
| 1 | ANALYZE TABLE payments |

mysql analyze table again

It is saying that the table is already up to date.

Optimize table statement

While working with the database, you do a lot of changes such as [insert](http://www.mysqltutorial.org/mysql-insert-statement.aspx), [update](http://www.mysqltutorial.org/mysql-update-data.aspx)and [delete](http://www.mysqltutorial.org/mysql-delete-statement.aspx)data in the table that may cause the physical storage of the table fragmented. As a result, the performance of database server is degraded.

MySQL provides you with a statement that allows you to optimize the table to avoid this defragmenting problem. The following illustrates how to optimize a table:



|  |  |
| --- | --- |
| 1 | OPTIMIZE TABLE table\_name |

It is recommended that you execute this statement for the tables that are updated frequently. For example, if you want to optimize the orders table to defragment it, you can perform the following statement:



|  |  |
| --- | --- |
| 1 | OPTIMIZE TABLE orders |

mysql optimize table statement

Check table statement

Something wrong can happen to the database server e.g., the server was shutdown unexpectedly, error while writing data to the hard disk, etc. These situations could make the database operate incorrectly and in the worst case it can be crashed.

MySQL allows you to check the integrity of database tables by using the CHECK TABLEstatement. The following illustrates the syntax of the CHECK TABLE statement:



|  |  |
| --- | --- |
| 1 | CHECK TABLE table\_name |

The CHECK TABLEstatement checks both table and its indexes. For example, you can use theCHECK TABLE statement to check the orderstable as follows:



|  |  |
| --- | --- |
| 1 | CHECK TABLE orders |

mysql check table statement

The CHECK TABLE statement only detects problems in a database table but it does not repair them. To repair the table, you use the REPAIR TABLE statement.

Repair table statement

The REPAIR TABLE statement allows you to repair some errors occurred in database tables. MySQL does not guarantee that the REPAIR TABLE statement can repair all errors that the tables may have. The following is the syntax of the REPAIR TABLE statement:



|  |  |
| --- | --- |
| 1 | REPAIR TABLE table\_name |

Suppose there are some errors in the orderstable and you need to fix them, you can use theREPAIR TABLE statement as the following query:



|  |  |
| --- | --- |
| 1 | REPAIR TABLE employees |

MySQL returns what it has done to the table and shows you whether the table was repaired or not.

mysql repair statement

In this tutorial, you have learned some very handy statements to maintain database tables in MySQL.

How to Backup Databases Using mysqldump Tool

**Summary***: in this tutorial you will learn how to use****mysqldump****tool back up MySQL databases.*

MySQL GUI tools such as phpMyAdmin, SQLyog and etc often provide features for backup MySQL databases with ease. However if your database is big, the backup process could be very slow because the backup file need to be transferred across the network to your client PC. As the result, the backup process increases locking time therefore MySQL unavailability.

MySQL provides a very useful tool for backup or dump MySQL databases locally on server very fast. The backup file is stored in the file system in the server so you just need to download it when needed. The tool to backup MySQL databases is mysqldump. It is located in the root/bin folder of MySQL installation folder. The mysqldump is a program provided by MySQL that can be used to dump databases for backup or transfer database to another database server. The dump file contains a set of SQL statements to create and populate tables. In addition, the mysqldump can be used to generate CSV, delimited or XML files. In this tutorial, we will focus only on how to backup MySQL database by using mysqldump tool.

How to Backup a MySQL Database

To backup a MySQL database, the database first has to exist in the database server and you have access to that server as well. You can use SSH or Telnet to login to the remote server if you do not have remote desktop to it. The command to backup a MySQL database as follows:



|  |  |
| --- | --- |
| 1 | mysqldump -u [username] –p[password] [database\_name] > [dump\_file.sql] |

The parameter of the command above as follows:

* [username]: valid MySQL username.
* [password]: valid password for the user. Note that there is no space between –p and the password.
* [database\_name]: database name you want to backup
* [dump\_file.sql]: dump file you want to generate.

By executing the above command all database structure and data will be exported into a single[dump\_file.sql] dump file. For example, in order to back our sample database classicmodels, we use the following command:



|  |  |
| --- | --- |
| 1 | mysqldump -u mysqltutorial –psecret  classicmodels > c:\temp\backup001.sql |

How to Backup MySQL Database Structure Only

If you only want to backup database structure only you just need to add an option –no-data to tell mysqldump that only database structure need to export as follows:



|  |  |
| --- | --- |
| 1 | mysqldump -u [username] –p[password] –no-data [database\_name] > [dump\_file.sql] |

For example to backup our sample database with structure only, you use the following command:



|  |  |
| --- | --- |
| 1 | mysqldump -u mysqltutorial –psecret  -no-data classicmodels > c:\temp\backup002.sql |

How to Backup MySQL Database Data Only

There is a case that you want to refresh data in staging and development system so the data in those systems are the same as production system. In this case you just need to export data only from production system and import it to staging and development system. In order to backup data only, you use option –no-create-info of mysqldump as follows:



|  |  |
| --- | --- |
| 1 | mysqldump -u [username] –p[password] –no-create-info [database\_name] > [dump\_file.sql] |

For example to backup our sample database with data only, you use the following command:



|  |  |
| --- | --- |
| 1 | mysqldump –u mysqltutorial –psecret –no-create-info classicmodels > c:\temp\backup002.sql |

How to Backup Multiple MySQL Databases into a Single File

If you want to backup multiple database just separate database name by command in the [database\_name]. If you want to back up all databases in the database server use the option **–all-database.**



|  |  |
| --- | --- |
| 1  2  3 | mysqldump -u [username] –p[password]  [dbname1,dbname2,…] > [dump\_file.sql]    mysqldump -u [username] –p[password] –all-database > [dump\_file.sql] |