The SQL LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.



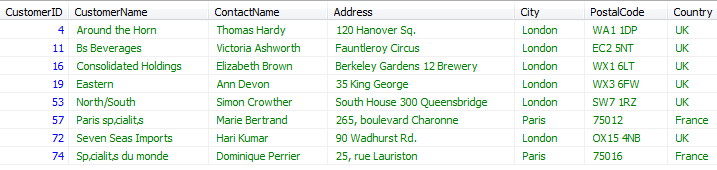
SELECT \* FROM customers where city like '%me%'



## The IN Operator

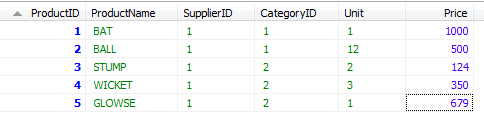
The IN operator allows you to specify multiple values in a WHERE clause.

SELECT \* FROM Customers WHERE City IN ('Paris','London');

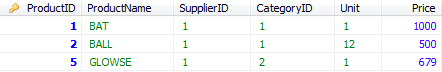


## The SQL BETWEEN Operator

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.



SELECT \* from products p WHERE p.Price BETWEEN 500 AND 1000



## NOT BETWEEN Operator Example

To display the products outside the range of the previous example, use NOT BETWEEN:

SELECT \* from products p WHERE p.Price NOT BETWEEN 500 AND 1000



BETWEEN Operator with Text Value Example

The following SQL statement selects all products with a ProductName beginning with any of the letter BETWEEN 'C' and 'M':

Example

SELECT \* FROM Products  
WHERE ProductName BETWEEN 'C' AND 'M';

BETWEEN Operator with Date Value Example

The following SQL statement selects all orders with an OrderDate BETWEEN '04-July-1996' and '09-July-1996':

Example

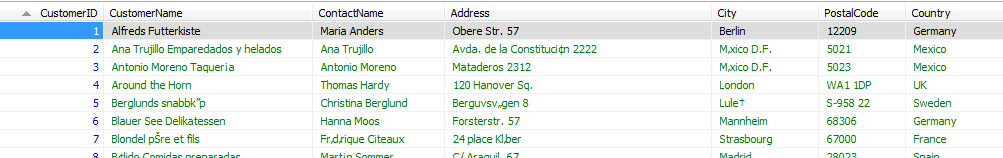
SELECT \* FROM Orders  
WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;

# SQL Aliases

SQL aliases are used to temporarily rename a table or a column heading

SQL aliases are used to give a database table, or a column in a table, a temporary name.

Basically aliases are created to make column names more readable.



SELECT o.OrderID, o.OrderDate, c.CustomerName as CUSTOMER FROM Customers AS c, Orders AS o WHERE c.CustomerID=o.CustomerID;



# SQL Joins

SQL joins are used to combine rows from two or more tables.

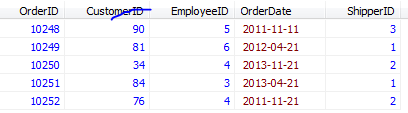
## SQL JOIN

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

***Notice that the "CustomerID" column in the "Orders" table refers to the customer in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.***

select \* from orders r



select \* from customers u



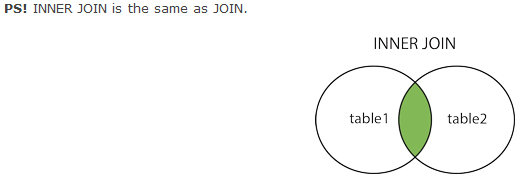
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

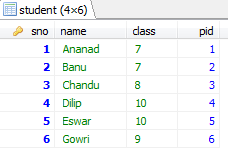


Different SQL JOINs

* **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables
* **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table
* **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table
* **FULL JOIN**: Return all rows when there is a match in ONE of the tables

# SQL INNER JOIN Keyword



Syntax:

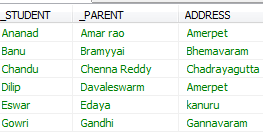
Select table1.col1, table2.col2, table3.col3 from

Table1 **INNER JOIN** Table2

**ON** table1.col1 = table2.col1 [condition]

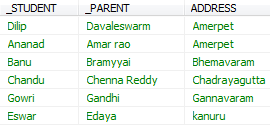
**ORDER BY** table.col name

select student.name as \_STUDENT, parent.pname as \_PARENT , parent.address as ADDRESS from student INNER JOIN parent ON student.pid = parent.pid



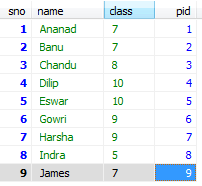
select student.name as \_STUDENT, parent.pname as \_PARENT , parent.address as ADDRESS from student INNER JOIN parent ON student.pid = parent.pid

ORDER BY parent.address

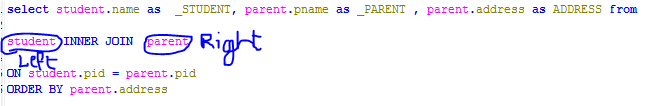


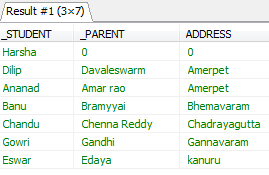
# SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

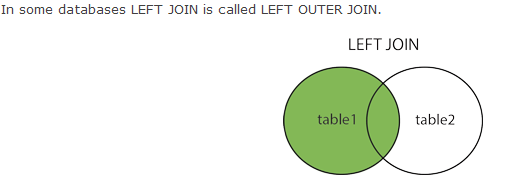
 

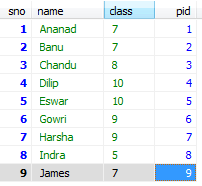
**INNER JOIN:**

****

****

## SQL LEFT JOIN

****

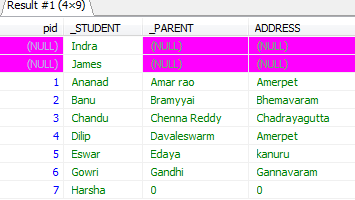
** **

select parent.pid , student.name as \_STUDENT, parent.pname as \_PARENT , parent.address as ADDRESS from

student LEFT JOIN parent

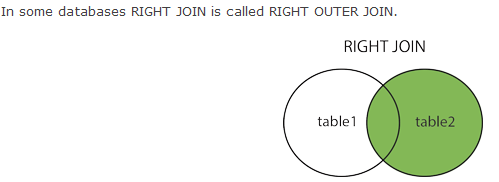
ON student.pid = parent.pid

ORDER BY parent.PID

****

## SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

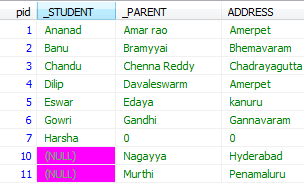


select parent.pid , student.name as \_STUDENT, parent.pname as \_PARENT , parent.address as ADDRESS from

student RIGHT JOIN parent

ON student.pid = parent.pid

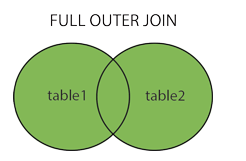
ORDER BY parent.PID

****

## SQL FULL OUTER JOIN Keyword

## The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

****

# SQL UNION Operator

The SQL UNION operator combines the result of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

# SQL SELECT INTO Statement

With SQL, you can copy information from one table into another.

The SELECT INTO statement copies data from one table and inserts it into a new table.

# CREATE DATABASE Statement

CREATE DATABASE dbname;

## SQL CREATE TABLE Example

CREATE TABLE Persons  
(  
PersonID int,  
LastName varchar(255),  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
);

# SQL Constraints

SQL constraints are used to specify rules for the data in a table.

 If there is any violation between the constraint and the data action, the action is aborted by the constraint

### SQL CREATE TABLE + CONSTRAINT Syntax

CREATE TABLE table\_name  
(  
column\_name1 data\_type(size) constraint\_name,  
column\_name2 data\_type(size) constraint\_name,  
column\_name3 data\_type(size) constraint\_name,  
....  
);

 In SQL, we have the following constraints:

* **NOT NULL** - Indicates that a column cannot store NULL value
* **UNIQUE** - Ensures that each row for a column must have a unique value
* **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
* **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
* **CHECK** - Ensures that the value in a column meets a specific condition
* **DEFAULT** - Specifies a default value when specified none for this column

## SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain unique values.

A primary key column cannot contain NULL values.

Each table should have a primary key, and each table can have only ONE primary key.

## PRIMARY KEY Constraint on ALTER TABLE

## ALTER TABLE Persons ADD PRIMARY KEY (P\_Id)

## To DROP a PRIMARY KEY Constraint

ALTER TABLE Persons  
DROP PRIMARY KEY

# SQL FOREIGN KEY Constraint

 FOREIGN KEY in one table points to a PRIMARY KEY in another table.

## SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CHECK (P\_Id>0)  
)

## SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

# SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes allow the database application to find data fast; without reading the whole table.

## Indexes

An index can be created in a table to find data more quickly and efficiently.

The users cannot see the indexes, they are just used to speed up searches/queries.

**Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So you should only create indexes on columns (and tables) that will be frequently searched against.

### SQL CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX PIndex  
ON Persons (LastName, FirstName)

# SQL Views

A view is a virtual table.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

### SQL CREATE VIEW Syntax

CREATE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition

## MySQL Date Functions

|  |  |
| --- | --- |
| [NOW()](http://www.w3schools.com/sql/func_now.asp) | Returns the current date and time |
| [CURDATE()](http://www.w3schools.com/sql/func_curdate.asp) | Returns the current date |
| [CURTIME()](http://www.w3schools.com/sql/func_curtime.asp) | Returns the current time |
| [DATE()](http://www.w3schools.com/sql/func_date.asp) | Extracts the date part of a date or date/time expression |
| [EXTRACT()](http://www.w3schools.com/sql/func_extract.asp) | Returns a single part of a date/time |
| [DATE\_ADD()](http://www.w3schools.com/sql/func_date_add.asp) | Adds a specified time interval to a date |
| [DATE\_SUB()](http://www.w3schools.com/sql/func_date_sub.asp) | Subtracts a specified time interval from a date |
| [DATEDIFF()](http://www.w3schools.com/sql/func_datediff_mysql.asp) | Returns the number of days between two dates |
| [DATE\_FORMAT()](http://www.w3schools.com/sql/func_date_format.asp) | Displays date/time data in different formats |

# SQL Functions

[« Previous](http://www.w3schools.com/sql/sql_datatypes.asp)

[Next Chapter »](http://www.w3schools.com/sql/sql_func_avg.asp)

SQL has many built-in functions for performing calculations on data.

## SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

* AVG() - Returns the average value
* COUNT() - Returns the number of rows
* FIRST() - Returns the first value
* LAST() - Returns the last value
* MAX() - Returns the largest value
* MIN() - Returns the smallest value
* SUM() - Returns the sum

## SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

* UCASE() - Converts a field to upper case
* LCASE() - Converts a field to lower case
* MID() - Extract characters from a text field
* LEN() - Returns the length of a text field
* ROUND() - Rounds a numeric field to the number of decimals specified
* NOW() - Returns the current system date and time
* FORMAT() - Formats how a field is to be displayed

# SQL Quick Reference From W3Schools

[« Previous](http://www.w3schools.com/sql/sql_func_format.asp)

[Next Chapter »](http://www.w3schools.com/sql/sql_hosting.asp)

|  |  |
| --- | --- |
| **SQL Statement** | **Syntax** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE | ALTER TABLE table\_name  ADD column\_name datatype  or  ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias) | SELECT column\_name AS column\_alias FROM table\_name  or  SELECT column\_name FROM table\_name  AS table\_alias |
| BETWEEN | SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, column\_name2 data\_type, ... ) |
| CREATE INDEX | CREATE INDEX index\_name ON table\_name (column\_name)  or  CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| DELETE | DELETE FROM table\_name WHERE some\_column=some\_value  or  DELETE FROM table\_name  (**Note:**Deletes the entire table!!)  DELETE \* FROM table\_name  (**Note:**Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name (SQL Server) DROP INDEX index\_name ON table\_name (MS Access) DROP INDEX index\_name (DB2/Oracle) ALTER TABLE table\_name DROP INDEX index\_name (MySQL) |
| DROP TABLE | DROP TABLE table\_name |
| GROUP BY | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name |
| HAVING | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name HAVING aggregate\_function(column\_name) operator value |
| IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2, value3,....)  *or*  INSERT INTO table\_name (column1, column2, column3,...) VALUES (value1, value2, value3,....) |
| INNER JOIN | SELECT column\_name(s) FROM table\_name1 INNER JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LEFT JOIN | SELECT column\_name(s) FROM table\_name1 LEFT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| RIGHT JOIN | SELECT column\_name(s) FROM table\_name1 RIGHT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| FULL JOIN | SELECT column\_name(s) FROM table\_name1 FULL JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern |
| ORDER BY | SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC] |
| SELECT | SELECT column\_name(s) FROM table\_name |
| SELECT \* | SELECT \* FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s) FROM table\_name |
| SELECT INTO | SELECT \* INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name |
| SELECT TOP | SELECT TOP number|percent column\_name(s) FROM table\_name |
| TRUNCATE TABLE | TRUNCATE TABLE table\_name |
| UNION | SELECT column\_name(s) FROM table\_name1 UNION SELECT column\_name(s) FROM table\_name2 |
| UNION ALL | SELECT column\_name(s) FROM table\_name1 UNION ALL SELECT column\_name(s) FROM table\_name2 |
| UPDATE | UPDATE table\_name SET column1=value, column2=value,... WHERE some\_column=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE column\_name operator value |

[**http://www.tutorialspoint.com/sql/**](http://www.tutorialspoint.com/sql/)

Transaction

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

Properties of Transactions:

Transactions have the following four standard properties, usually referred to by the acronym ACID:

* **Atomicity:** ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.
* **Consistency:** ensures that the database properly changes states upon a successfully committed transaction.
* **Isolation:** enables transactions to operate independently of and transparent to each other.
* **Durability:** ensures that the result or effect of a committed transaction persists in case of a system failure.

Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They can not be used while creating tables or dropping them because these operations are automatically commited in the database.

The COMMIT Command:

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.

The SAVEPOINT Command:

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

SQL> SAVEPOINT SP1;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=1;

1 row deleted.

SQL> SAVEPOINT SP2;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=2;

1 row deleted.

SQL> SAVEPOINT SP3;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=3;

1 row deleted.

SQL> ROLLBACK TO SP2;

Rollback complete.

The RELEASE SAVEPOINT Command:

The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for RELEASE SAVEPOINT is as follows:

Here is an example showing you usage of temporary table:

mysql> CREATE TEMPORARY TABLE SALESSUMMARY (

-> product\_name VARCHAR(50) NOT NULL

-> , total\_sales DECIMAL(12,2) NOT NULL DEFAULT 0.00

-> , avg\_unit\_price DECIMAL(7,2) NOT NULL DEFAULT 0.00

-> , total\_units\_sold INT UNSIGNED NOT NULL DEFAULT 0

);

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO SALESSUMMARY

-> (product\_name, total\_sales, avg\_unit\_price, total\_units\_sold)

-> VALUES

-> ('cucumber', 100.25, 90, 2);

# PL/SQL Tutorial

* PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
* PL/SQL provides support for Object-Oriented Programming.
* PL/SQL provides support for Developing Web Applications and Server Pages.
* PL/SQL is a block-structured language, meaning that PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts:

|  |  |
| --- | --- |
| **S.N.** | **Sections & Description** |
| 1 | **Declarations**  This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program. |
| 2 | **Executable Commands**  This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed. |
| 3 | **Exception Handling**  This section starts with the keyword **EXCEPTION**. This section is again optional and contains exception(s) that handle errors in the program. |

* Every PL/SQL statement ends with a semicolon **(;)**. PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**. Here is the basic structure of a PL/SQL block:
* DECLARE
* <declarations section>
* BEGIN
* <executable command(s)>
* EXCEPTION
* <exception handling>
* END;

[http://www.mysqltutorial.org/](http://www.mysqltutorial.org/#)

# 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

Before doing anything else with the data, you need to create a database. A database is a container of data. It stores contacts, vendors, customers or any kind of data that you can think of. In MySQL, a database is a collection of objects that are used to store and manipulate data such as tables, [database views](http://www.mysqltutorial.org/mysql-views-tutorial.aspx), [triggers](http://www.mysqltutorial.org/mysql-triggers.aspx), [stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), etc.

To create a database in MySQL, you use the CREATE DATABASE statement as follows:



|  |  |
| --- | --- |
| 1 | CREATE DATABASE [IF NOT EXISTS] database\_name; |

Let’s examine the CREATE DATABASE statement in greater detail:

* Followed by the CREATE DATABASE statement is database name that you want to create. It is recommended that the database name should be as meaningful and descriptive as possible.
* The IF NOT EXISTS is an optional element of the statement. The IF NOT EXISTS statement prevents you from an error of creating a new database that already exists in the database server. You cannot have 2 databases with the same name in a MySQL database server.

For example, to create classicmodels database, you can execute the CREATE DATABASE statement as follows:



|  |  |
| --- | --- |
| 1 | CREATE DATABASE classicmodels; |

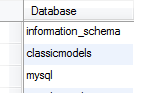
After executing the statement, MySQL returns a message to notify that the new database has been created successfully or not.

## Displaying Databases

The SHOW DATABASE statement displays all databases in the MySQL database server. You can use the SHOW DATABASE statement to check the database that you’ve created or to see all the databases on the database server before you create a new database, for example:



|  |  |
| --- | --- |
| 1 | SHOW DATABASES; |



We have thee databases in the MySQL database server. The information\_schema and mysqlare the default databases that are available when we install MySQL, and the classicmodels is the new database that we have created.

## Selecting a database to work with

Before working with a particular database, you must tell MySQL which database you want to work with by using the USE statement.



|  |  |
| --- | --- |
| 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; |

From now all operations such as [querying data](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [create new tables](http://www.mysqltutorial.org/mysql-create-table/) or [stored procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) which you perform, will take effects on the current database.

## Removing Databases

Removing database means you delete the database physically. All the data and related objects inside the database are permanently deleted and this cannot be undone, therefore it is very important to execute this query with extra cautions.

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



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

Followed the DROP DATABASE is the database name that you want to remove. Similar to the CREATE DATABASE statement, the IF EXISTS is an optional part of the statement to prevent you from removing a database that does not exist in the database server.

If you want to practice with the DROP DATABASE statement, you can create a new database, make sure that it is created and remove it. Take a look at the following queries:



|  |  |
| --- | --- |
| 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.
* Does the values of the data type can be indexed.
* How MySQL compares the value of a specific data type.

## Numeric Data Types

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

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

## String Data Types

In MySQL, string can hold anything from plain text to binary data such as images and files. String can be compared and searched based on pattern matching by using the [LIKE operator](http://www.mysqltutorial.org/sql-like-mysql.aspx) or [regular expression](http://www.mysqltutorial.org/mysql-regular-expression-regexp.aspx). The following table shows you the string data types in MySQL:

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

## Date and Time Data Types

MySQL provides types for date and time as well as a combination of date and time. In addition, MySQL also provides [timestamp](http://www.mysqltutorial.org/mysql-timestamp.aspx) data type for tracking the changes of a row in a table. If you just want to store the year without date and month, you can use YEAR data type. The following table illustrates the MySQL 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 kind 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.

The following illustrates the syntax of the CREATE TABLE statement in the simple form:



|  |  |
| --- | --- |
| 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 InnoDB by 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).
* The  NOT 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:

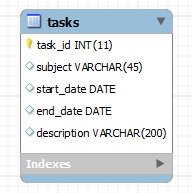


|  |  |
| --- | --- |
| 1 | PRIMARY KEY (col1,col2,...) |

## 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)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:



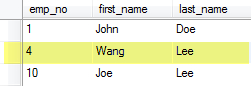
|  |  |
| --- | --- |
| 1  2  3  4 | UPDATE employees  SET first\_name = 'Joe',      emp\_no = 10  WHERE emp\_no = 3; |

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 inside parentheses followed the PRIMARY KEY keywords.



|  |  |
| --- | --- |
| 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.

### Defining MySQL PRIMARY KEY constraints using ALTER TABLE statement

If a table, for some reasons, does not have a primary key, you can use the [ALTER TABLE](http://www.mysqltutorial.org/mysql-alter-table.aspx)statement to add a column that has all necessary primary key’s characteristics to the primary key as the following statement:



|  |  |
| --- | --- |
| 1  2 | ALTER TABLE table\_name  ADD PRIMARY KEY(primary\_key\_column); |

The following example adds the id column to the primary key.

First, create t1 table  without defining the primary key.



|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TABLE t1(     id int,     title varchar(255) NOT NULL  ); |

Second, add the id column to primary key of the t1 table.



|  |  |
| --- | --- |
| 1  2 | ALTER TABLE t1  ADD PRIMARY KEY(id); |

## 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.

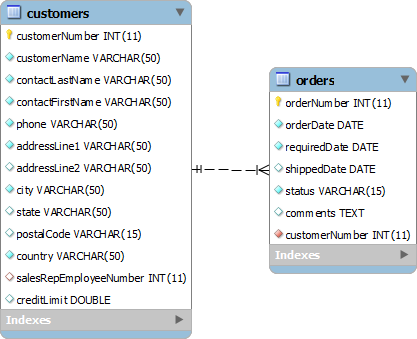
# MySQL Foreign Key

**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.

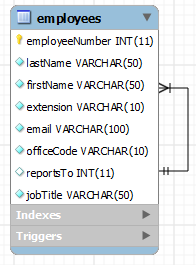
The customers table is called parent table or referenced table, and the orders table is known aschild table or referencing table.

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:

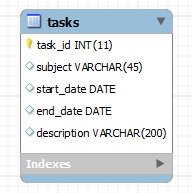
* First, you specify the table name that you want to change after the ALTER TABLE keywords.
* Second, you list a set of actions that you want to apply to the table. An action can be anything such as adding a new column, adding primary key, renaming table, etc. The  ALTER TABLE statement allows you to 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  tasks in 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 the value of the  task\_id column 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  description column 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:

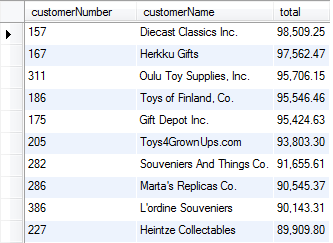


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TEMPORARY TABLE top10customers  SELECT p.customerNumber,         c.customerName,         FORMAT(SUM(p.amount),2) total  FROM payments p  INNER JOIN customers c ON c.customerNumber = p.customerNumber  GROUP BY p.customerNumber  ORDER BY total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

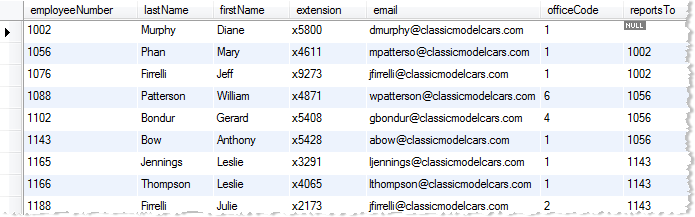
You will learn about each clause in more detail in the next tutorial. In this tutorial, we are going to focus on the simple form of the SELECT statement.

## MySQL SELECT Examples

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



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

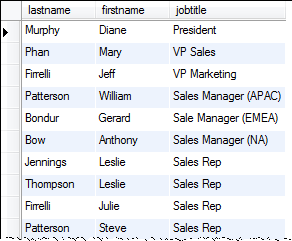


The asterisk (\*) notation is a shorthand of selecting all columns in the table.

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

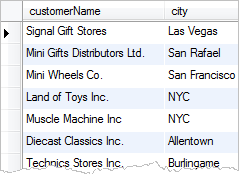
**Summary:** you will learn how to use **MySQL WHERE** clause to filter rows returned from the SELECT statement.

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:



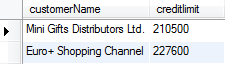
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT customerName, city  FROM customers  WHERE country = 'USA' AND        city    = 'NYC'; |



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

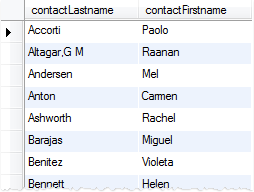
Let’s practice with some examples of using the ORDER BY clause.

## MySQL ORDER BY examples

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



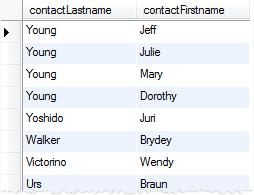
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT contactLastname,         contactFirstname  FROM customers  ORDER BY contactLastname; |



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:



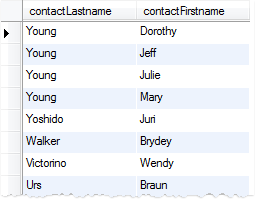
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT contactLastname,         contactFirstname  FROM customers  ORDER BY contactLastname DESC |



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; |



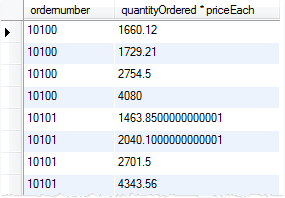
In the query above, the ORDER BY clause sorts the result set by  last name in descending order first, and then sorts the sorted result set by first name in ascending order to produce the final result set.

### 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 |



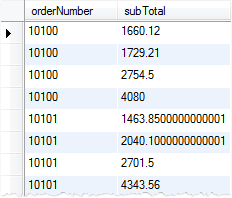
To make the result more readable, you can use a [column alias](http://www.mysqltutorial.org/mysql-alias/), and sort the result based on the column alias.



|  |  |
| --- | --- |
| 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

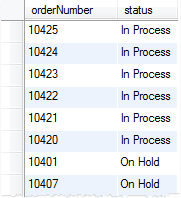
The ORDER BY clause enables you to define your own custom sort order for the values in a column using the FIELD() function. For example, if you want to sort the orders based on the following status by the following order:

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.

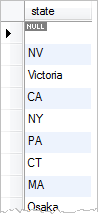
## MySQL DISTINCT and NULL values

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 |



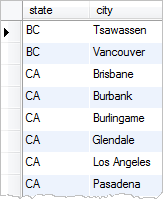
## MySQL DISTINCT with multiple columns

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:

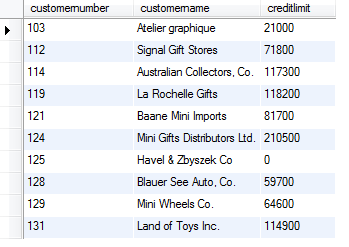


|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM tbl  LIMIT N |

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; |



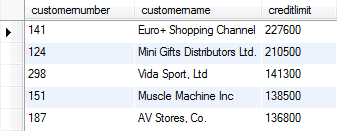
## Using MySQL LIMIT to get the highest and lowest values

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:



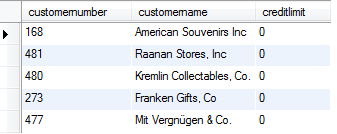
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT customernumber,         customername,         creditlimit  FROM customers  ORDER BY creditlimit DESC  LIMIT 5; |



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; |



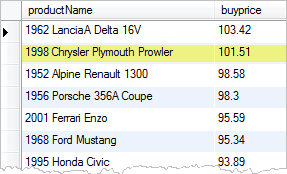
## Using MySQL LIMIT to get the N highest values

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), [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx), etc.

The IN operator returns 1 if the value of the  column or the 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.

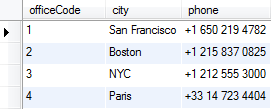
Let’s practice with some examples of using the IN operator.

## 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:



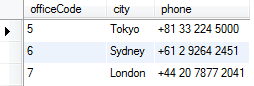
|  |  |
| --- | --- |
| 1  2  3 | SELECT officeCode, city, phone  FROM offices  WHERE country = 'USA' OR country = 'France' |

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') |

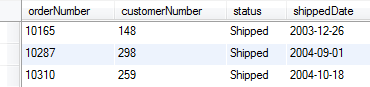


## MySQL IN with subquery

The IN operator is often used with a [subquery](http://www.mysqltutorial.org/mysql-subquery/). For example, if you want to find order whose 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, otherwise 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

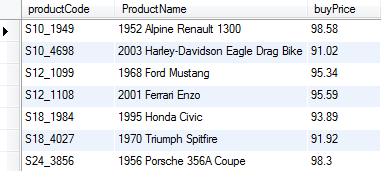
Let’s practice with some examples of using the BETWEEN operator.

### 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:

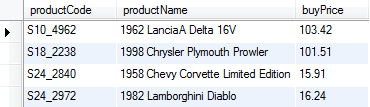


|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productCode,         productName,         buyPrice  FROM products  WHERE buyPrice >= 90 AND buyPrice <= 100 |

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.



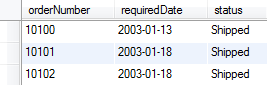
|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT productCode,         productName,         buyPrice  FROM products  WHERE buyPrice < 20 OR buyPrice > 100 |

### 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

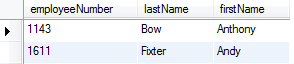
Let’s practice with some examples of how to use the LIKE operator.

### 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:



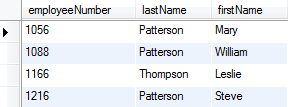
|  |  |
| --- | --- |
| 1  2  3 | SELECT employeeNumber, lastName, firstName  FROM employees  WHERE firstName LIKE 'a%' |

  
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:



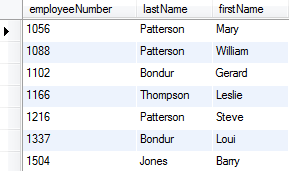
|  |  |
| --- | --- |
| 1  2  3 | SELECT employeeNumber, lastName, firstName  FROM employees  WHERE lastName LIKE '%on' |



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

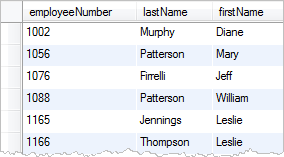
### MySQL LIKE operator with NOT operator

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:

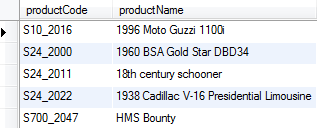


|  |  |
| --- | --- |
| 1  2  3 | SELECT productCode, productName  FROM products  WHERE productCode LIKE '%\\_20%' |

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:

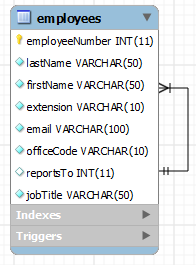


|  |  |
| --- | --- |
| 1  2 | SELECT [col1 | expression] AS `descriptive name`  FROM table\_name |

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.

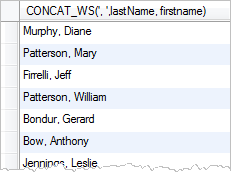
Let’s look at the employees table in the [sample database.](http://www.mysqltutorial.org/mysql-sample-database.aspx)



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.



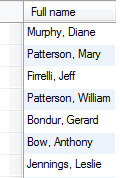
|  |  |
| --- | --- |
| 1  2 | SELECT CONCAT\_WS(', ',lastName, firstname)  FROM employees; |



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:



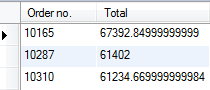
|  |  |
| --- | --- |
| 1  2  3 | SELECT CONCAT\_WS(', ',lastName, firstname) `Full name`  FROM employees  ORDER BY `Full name`; |



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:

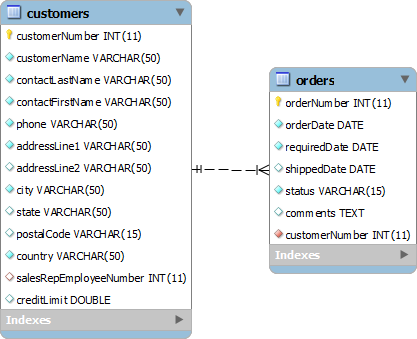


|  |  |
| --- | --- |
| 1 | table\_name AS table\_alias |

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/).

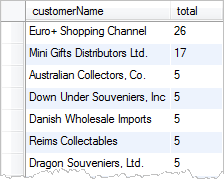
Let’s look at the customers and orders tables:



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

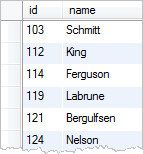
Let’s practice with an example of using MySQL UNION to get a better understanding.

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

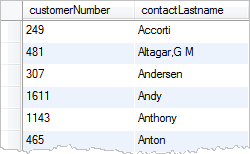
In the example above, we used the [column alias](http://www.mysqltutorial.org/mysql-alias/) for each column in the SELECT statements. What would be the output if we didn’t use the column alias? MySQL uses the names of columns in the firstSELECT statement as the labels for the output.

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*.*



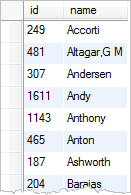
## MySQL UNION with ORDER BY

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; |

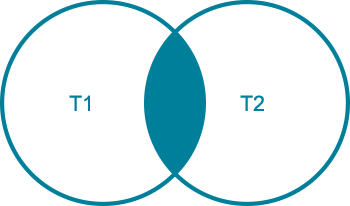
Let’s simplify the syntax above by assuming that we are joining two tables T1 and T2 using theINNER JOIN clause.

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**

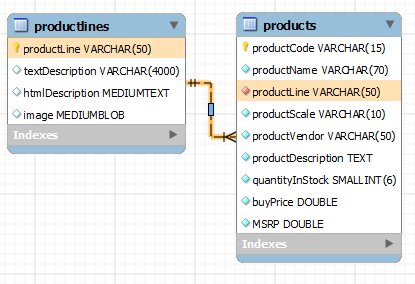
## Avoid ambiguous column error in MySQL INNER JOIN

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 C column 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; |

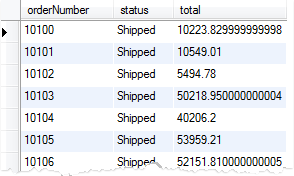
### MySQL INNER JOIN - Products Data Example

### 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.