How to Start MySql Server

Download latest version of XAMPP From <https://www.apachefriends.org/download.html>

Install XAMPP and open XAMPP Control Panel

From Control panel you need to start two services apache, mysql

Now open browser and enter URL <http://localhost/phpmyadmin/> or press admin button from XAMPP control panel.

Now you are connected with MySql server home page

**MySQL Tutorial**

MySQL is a widely used relational database management system (RDBMS).

MySQL is free and open-source.

MySQL is ideal for both small and large applications.

# Introduction to MySQL

MySQL is a very popular open-source relational database management system (RDBMS).

## What is MySQL?

* MySQL is a relational database management system
* MySQL is open-source
* MySQL is free
* MySQL is ideal for both small and large applications
* MySQL is very fast, reliable, scalable, and easy to use
* MySQL is cross-platform
* MySQL is compliant with the ANSI SQL standard
* MySQL was first released in 1995
* MySQL is developed, distributed, and supported by **Oracle Corporation**
* MySQL is named after co-founder Monty Widenius's daughter: My

## Who Uses MySQL?

* Huge websites like Facebook, Twitter, Airbnb, Booking.com, Uber, GitHub, YouTube, etc.
* Content Management Systems like WordPress, Drupal, Joomla!, Contao, etc.
* A very large number of web developers around the world

## Show Data On Your Web Site

To build a web site that shows data from a database, you will need:

* An RDBMS database program (like MySQL)
* A server-side scripting language, like PHP
* To use SQL to get the data you want
* To use HTML / CSS to style the page

# MySQL RDBMS

## What is RDBMS?

RDBMS stands for Relational Database Management System.

RDBMS is a program used to maintain a relational database.

RDBMS is the basis for all modern database systems such as MySQL, Microsoft SQL Server, Oracle, and Microsoft Access.

RDBMS uses [SQL queries](https://www.w3schools.com/sql/default.asp) to access the data in the database.

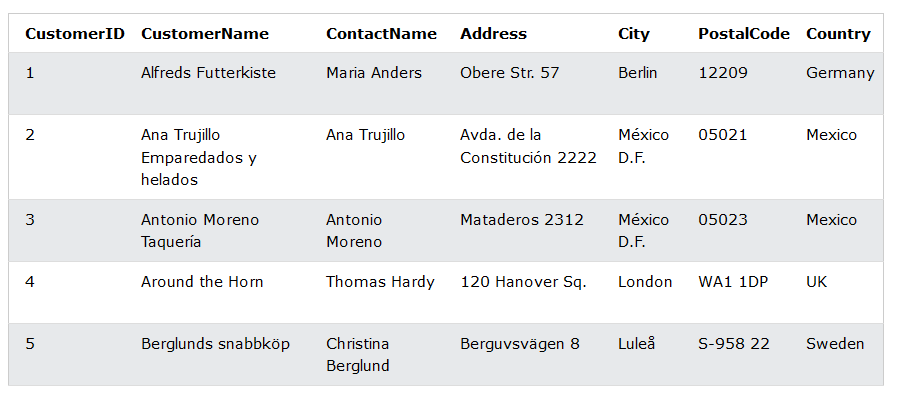
## What is a Database Table?

A table is a collection of related data entries, and it consists of columns and rows.

A column holds specific information about every record in the table.

A record (or row) is each individual entry that exists in a table.

Look at a selection from the Northwind "Customers" table:



The columns in the "Customers" table above are: CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country. The table has 5 records (rows).

## What is a Relational Database?

A relational database defines database relationships in the form of tables. The tables are related to each other - based on data common to each.

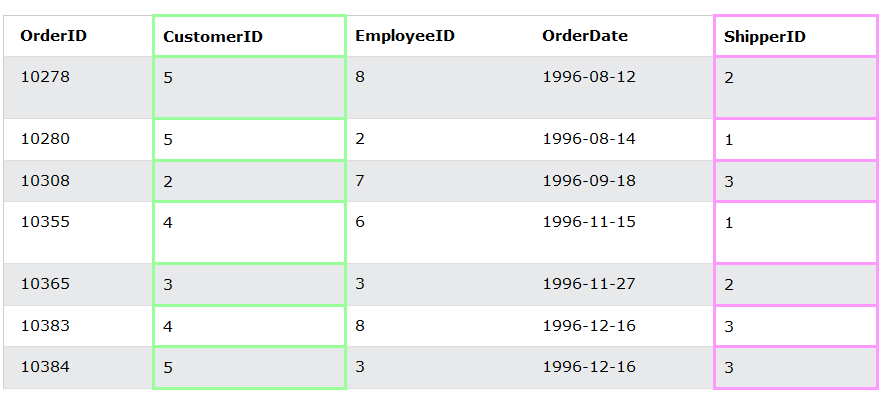
Look at the following three tables "Customers", "Orders", and "Shippers" from the Northwind database:

### Customers Table



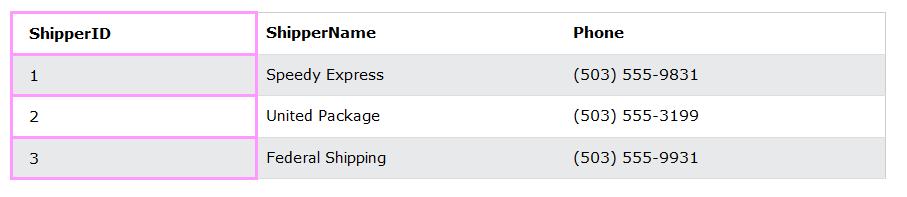
The relationship between the "Customers" table and the "Orders" table is the CustomerID column:

### Orders Table



The relationship between the "Orders" table and the "Shippers" table is the ShipperID column:

### Shippers Table



# MySQL SQL

## What is SQL? (Structured Query Language)

SQL is the standard language for dealing with Relational Databases.

SQL is used to insert, search, update, and delete database records.

## How to Use SQL

The following SQL statement selects all the records in the "Customers" table:

SELECT \* FROM Customers;

## Keep in Mind That...

* SQL keywords are NOT case sensitive: select is the same as SELECT

In this tutorial we will write all SQL keywords in upper-case.

## Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

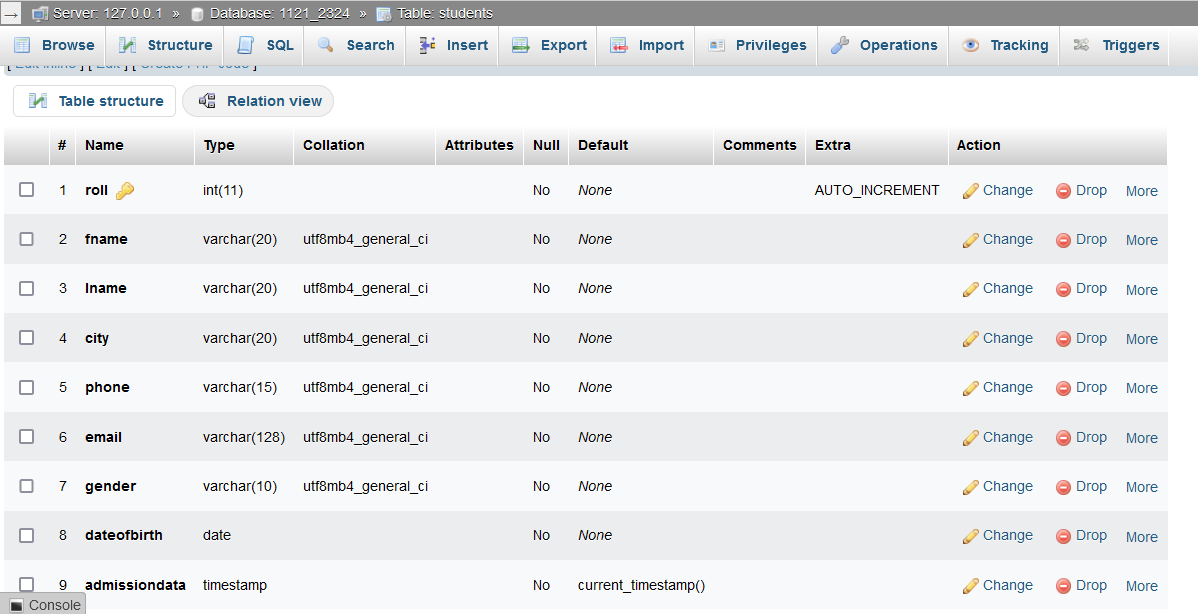
Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

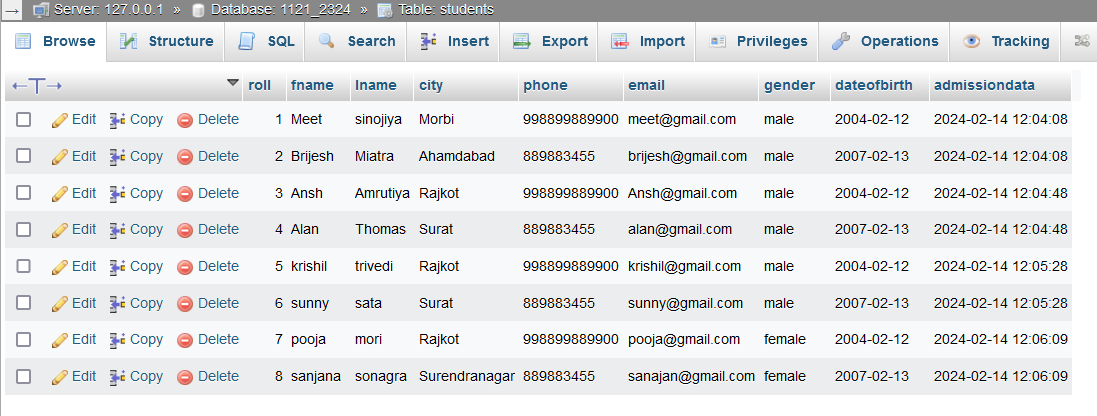
In this tutorial, we will use semicolon at the end of each SQL statement.

## Some of The Most Important SQL Commands

* SELECT - extracts data from a database
* UPDATE - updates data in a database
* DELETE - deletes data from a database
* INSERT INTO - inserts new data into a database
* CREATE DATABASE - creates a new database
* ALTER DATABASE - modifies a database
* CREATE TABLE - creates a new table
* ALTER TABLE - modifies a table
* DROP TABLE - deletes a table
* CREATE INDEX - creates an index (search key)
* DROP INDEX - deletes an index

Before continue with MySql you need to create sample database and add some data in it.





# MySQL SELECT Statement

## The MySQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

### SELECT Syntax

SELECT column1, column2, ... FROM table\_name;

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT \* FROM table\_name;

SELECT roll, fname, lname, city from students

SELECT \* from students;

## The MySQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

### SELECT DISTINCT Syntax

SELECT DISTINCT column1, column2, ... FROM table\_name;

SELECT city from students;

SELECT DISTINCT city from students;

SELECT count(DISTINCT city) from students;

SELECT roll, fname, lname, dateofbirth from students

# MySQL WHERE Clause

## The MySQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

### WHERE Syntax

SELECT column1, column2, ... FROM table\_nameWHERE condition;

SELECT roll, fname, lname, dateofbirth from students WHERE roll = 1;

SELECT roll, fname, lname, dateofbirth from students WHERE roll > 5;

SELECT roll, fname, lname, dateofbirth from students WHERE not roll > 5;

**Note:** The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

SELECT \* from students WHERE city = 'rajkot'

SELECT \* from students WHERE not city = 'rajkot';

SELECT \* from students WHERE city <> "rajkot";

## Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

SELECT \* from students WHERE not city = rajkot;

SELECT \* from students WHERE roll BETWEEN 1 and 5

# MySQL AND, OR and NOT Operators

## The MySQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.
* The NOT operator displays a record if the condition(s) is NOT TRUE.

### AND Syntax

SELECT column1, column2, ... FROM table\_name WHERE condition1 AND condition2 AND condition3 ...;

SELECT \* from students WHERE roll = 1

SELECT \* from students WHERE roll = 1 and city = 'Rajkot';

### OR Syntax

SELECT column1, column2, ... FROM table\_name WHERE condition1 OR condition2 OR condition3 ...;

SELECT \* from students WHERE city = 'surat' or city = 'Rajkot';

### NOT Syntax

SELECT column1, column2, ... FROM table\_name WHERE NOT condition;

SELECT \* from students WHERE not (city = 'surat' or city = 'Rajkot');

## Combining AND, OR and NOT

You can also combine the AND, OR and NOT operators.

SELECT \* from students WHERE roll = 1 and (city = 'surat' or city = 'Rajkot' or city = 'morbi');

SELECT \* from students WHERE roll = 1 or roll = 5 and city = 'Rajkot';

# MySQL ORDER BY Keyword

## The MySQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

### ORDER BY Syntax

SELECT column1, column2, ...  
FROM table\_name  
ORDER BY column1, column2, ... ASC|DESC;

SELECT \* FROM students

SELECT \* FROM students ORDER by fname

## ORDER BY DESC Example

SELECT \* FROM students ORDER by fname desc

## ORDER BY Several Columns Example

SELECT \* FROM students ORDER by fname, city

SELECT \* FROM students ORDER by fname, city desc

SELECT \* FROM students ORDER by fname asc, city desc

# MySQL INSERT INTO Statement

## The MySQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

### INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

INSERT into students (fname, lname, city, phone, email, gender, dateofbirth) values ('demo', 'text', 'example', '9900009900', 'demo@example.com', 'male', '2001-02-01')

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table\_name*  
VALUES (*value1*, *value2*, *value3*, ...);

INSERT into students values ('demo', 'text', 'example', '9900009900', 'demo@example.com', 'male', '2001-02-01')

#1136 - Column count doesn't match value count at row 1

INSERT into students values (null,'demo', 'text', 'example', '9900009900', 'demo@example.com', 'male', '2001-02-01', null)

**Did you notice that we did not insert any number into the CustomerID field?**  
The CustomerID column is an [auto-increment](https://www.w3schools.com/mysql/mysql_autoincrement.asp) field and will be generated automatically when a new record is inserted into the table.

## Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

INSERT into students (fname, lname, city) VALUES ('another', 'example', 'of insert')

# MySQL NULL Values

## What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

**Note:** A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

If you allow to NULL values in your table first you need to modify column to accept NULL values.

Go to structure section of our table then select specific column to allow null values click on change and select null check box and click save.

INSERT into students (fname, lname, city) VALUES ('another', 'example', 'of insert')

## How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

### IS NULL Syntax

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NULL;

### IS NOT NULL Syntax

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

SELECT \* from students WHERE gender = 'NULL'

SELECT \* from students WHERE gender = NULL;

SELECT \* from students WHERE gender = '';

SELECT \* from students WHERE gender is null;

SELECT \* from students WHERE gender is not null;

# MySQL UPDATE Statement

## The MySQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

### UPDATE Syntax

UPDATE table\_name SET column1 = value1, column2 = value2, ... WHERE condition;

**Note:** Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

UPDATE students set city = 'bhuj' WHERE roll = 1

UPDATE students set phone = '9090908080' WHERE roll = 1

UPDATE students set city = 'Gandhinagar'

## UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

UPDATE students set city = 'rajkot' WHERE roll >= 1 and roll <= 5

UPDATE students set city = 'ahamdabad' WHERE roll >= 6 and roll <= 10;

## Update Warning!

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

UPDATE students set fname = 'KRISHIL', city = 'baroda', phone = '9998889990' WHERE roll = 5

# MySQL LIMIT Clause

## The MySQL LIMIT Clause

The LIMIT clause is used to specify the number of records to return.

The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

### LIMIT Syntax

SELECT column\_name(s) FROM table\_nameWHERE condition LIMIT number;

SELECT \* FROM students

SELECT \* FROM students LIMIT 5;

SELECT \* FROM students where city = 'rajkot' LIMIT 5;

SELECT \* FROM students LIMIT 5 OFFSET 5;

SELECT \* FROM students LIMIT 5 OFFSET 10;

MySQL provides a way to handle this: by using OFFSET.

The SQL query below says "return only 3 records, start on record 4 (OFFSET 3)":

SELECT \* FROM Customers LIMIT 3 OFFSET 3;

SELECT \* FROM students LIMIT 5 OFFSET 10;

SELECT \* FROM students LIMIT 10, 5; --10 offset 5 limit

# MySQL MIN() and MAX() Functions

## MySQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

### MIN() Syntax

SELECT MIN(column\_name)  
FROM table\_name  
WHERE condition;

### MAX() Syntax

SELECT MAX(column\_name)  
FROM table\_name  
WHERE condition;

SELECT max(dateofbirth) FROM students

SELECT min(dateofbirth) FROM students;

SELECT max(roll) FROM students

SELECT min(roll) FROM students

# MySQL COUNT(), AVG() and SUM() Functions

## MySQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

### COUNT() Syntax

SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

The AVG() function returns the average value of a numeric column.

### AVG() Syntax

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

The SUM() function returns the total sum of a numeric column.

### SUM() Syntax

SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;

SELECT COUNT(roll) FROM students

SELECT COUNT(roll) FROM students WHERE city = 'rajkot';

SELECT COUNT(roll) FROM students WHERE not city = 'rajkot';

SELECT sum(roll) FROM students

SELECT avg(roll) FROM students;

SELECT sum(roll), avg(roll) FROM students;

# MySQL LIKE Operator

## The MySQL LIKE Operator

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

There are two wildcards often used in conjunction with the LIKE operator:

* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

The percent sign and the underscore can also be used in combinations!

SELECT \* from students

SELECT \* from students WHERE fname like 'a%';

SELECT \* from students WHERE fname like '%a%';

SELECT \* from students WHERE fname like '%a';

SELECT \* from students WHERE fname like '\_a%';

### LIKE Syntax

SELECT column1, column2, ... FROM table\_name WHERE columnN LIKE pattern;

**Tip:** You can also combine any number of conditions using AND or OR operators.

SELECT \* from students WHERE fname like 'a\_%';

SELECT \* from students WHERE fname like 's%a';

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that start with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that end with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_%' | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE CustomerName LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

# MySQL Wildcards

## MySQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the [LIKE](https://www.w3schools.com/mysql/mysql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

### Wildcard Characters in MySQL

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| \_ | Represents a single character | h\_t finds hot, hat, and hit |

The wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_%\_%' | Finds any values that starts with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a" and ends with "o" |

# MySQL IN Operator

## The MySQL IN Operator

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

The IN operator is a shorthand for multiple **OR** conditions.

### IN Syntax

SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1, value2, ...);

or:

SELECT column\_name(s) FROM table\_name WHERE column\_name IN (*SELECT* STATEMENT);

SELECT \* from students WHERE city = 'rajkot' or city = 'ahamdabad' or city = 'bhuj'

SELECT \* from students WHERE city in ('rajkot', 'ahamdabad', 'baroda')

## IN Operator Examples

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

SELECT \* FROM Customers WHERE Country IN ('Germany', 'France', 'UK');

SELECT \* from students WHERE city in ('rajkot', 'ahamdabad', 'baroda')

SELECT \* from students WHERE city not in('rajkot', 'ahamdabad', 'baroda');

# MySQL BETWEEN Operator

## The MySQL BETWEEN Operator

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

The BETWEEN operator is inclusive: begin and end values are included.

### BETWEEN Syntax

SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2;

SELECT \* FROM students WHERE roll BETWEEN 1 and 5

SELECT \* FROM students WHERE roll not BETWEEN 1 and 5;

## BETWEEN Text Values Example

SELECT \* FROM students WHERE fname BETWEEN 'ansh' and 'krishil'

SELECT \* FROM students WHERE dateofbirth BETWEEN '2000-01-01' and '2005-12-31'

SELECT \* FROM students WHERE dateofbirth not BETWEEN '2000-01-01' and '2005-12-31';

# MySQL DELETE Statement

## The MySQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

### DELETE Syntax

DELETE FROM table\_name WHERE condition;

**Note:** Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

If you try delete query on your table and you need to data back must create shadow copy of your current table to prevent any data loss

CREATE TABLE studentsBackup as SELECT \* FROM students

## SQL DELETE Example

DELETE from students WHERE Roll = 1

DELETE from students WHERE city = 'gandhinagar'

DELETE from students WHERE roll > 5

DELETE from students

Restore data from backup table

INSERT into students SELECT \* from studentsbackup

Delete from students (empty students table)

INSERT into students (fname, lname, city, phone, email, gender, dateofbirth) values ('demo', 'text', 'example', '9900009900', 'demo@example.com', 'male', '2001-02-01')

Notice the roll number started from where we leave before delete

Select \* from students

If you need to reset whole table with data and data structure then you need to run truncate table

TRUNCATE TABLE students

INSERT into students (fname, lname, city, phone, email, gender, dateofbirth) values ('demo', 'text', 'example', '9900009900', 'demo@example.com', 'male', '2001-02-01')

Select \* from students

# MySQL Aliases

## MySQL Aliases

Aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

### Alias Column Syntax

SELECT column\_name AS alias\_name  
FROM table\_name;

### Alias Table Syntax

SELECT column\_name(s)  
FROM table\_name AS alias\_name;

SELECT \* from students

SELECT roll, fname, lname, city, phone, email, gender, dateofbirth from students;

SELECT roll as "Roll Number", fname as "First Name", lname as "Last Name", city as "Home Town", phone as "Phone Number", email as "Email Address", gender as "Student Gender", dateofbirth as "Date of Birth" from students;

SELECT roll "Roll Number", fname "First Name", lname "Last Name", city "Home Town", phone "Phone Number", email "Email Address", gender "Student Gender", dateofbirth "Date of Birth" from students;

SELECT roll RollNumber, fname FirstName, lname LastName, city HomeTown, phone PhoneNumber, email EmailAddress, gender StudentGender, dateofbirth DateofBirth from students;

The following SQL statement creates two aliases, one for the CustomerName column and one for the ContactName column. **Note:** Single or double quotation marks are required if the alias name contains spaces:

SELECT concat\_ws("\_", roll, fname, lname, city, email, phone, gender, dateofbirth, admissiondata) FROM students

SELECT concat\_ws("\_", roll, fname, lname, city, email, phone, gender, dateofbirth, admissiondata) as "Student Information" FROM students;

SELECT concat\_ws(" \* ", roll, fname, lname, city, email, phone, gender, dateofbirth, admissiondata) as "Student Information" FROM students;

## Alias for Tables Example

Without alias

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.gender, students.dateofbirth, students.admissiondata, attendance.absents, attendance.presents from students, attendance WHERE students.roll = 1 and students.roll = attendance.roll

With alias

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.email, s.gender, s.dateofbirth, s.admissiondata, a.absents, a.presents from students as s, attendance as a WHERE s.roll = 1 and s.roll = a.roll;

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.email, s.gender, s.dateofbirth, s.admissiondata, a.absents, a.presents from students s, attendance a WHERE s.roll = 1 and s.roll = a.roll;

Aliases can be useful when:

* There are more than one table involved in a query
* Functions are used in the query
* Column names are big or not very readable
* Two or more columns are combined together

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.email, s.gender, s.dateofbirth, s.admissiondata, a.absents, a.presents, sum(a.absents+a.presents) from students s, attendance a WHERE s.roll = 1 and s.roll = a.roll;

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.email, s.gender, s.dateofbirth, s.admissiondata, a.absents, a.presents, sum(a.absents+a.presents) "Total Days" from students s, attendance a WHERE s.roll = 1 and s.roll = a.roll;

# MySQL Joins

## MySQL Joining Tables

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

SELECT students.roll,students.fname, students.lname, students.city, students.phone, students.email, students.gender, students.dateofbirth, students.admissiondata, marks.total, marks.result from students inner JOIN marks on students.roll = marks.roll

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.email, s.gender, s.dateofbirth, s.admissiondata, m.total, m.result from students s inner JOIN marks m on s.roll = m.roll;

## Supported Types of Joins in MySQL

* INNER JOIN: Returns records that have matching values in both tables
* LEFT JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT JOIN: Returns all records from the right table, and the matched records from the left table
* CROSS JOIN: Returns all records from both tables
*       

# MySQL INNER JOIN Keyword

## MySQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.



### INNER JOIN Syntax

SELECT column\_name(s) FROM table1 INNER JOIN table2ON table1.column\_name = table2.column\_name;

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.email, s.gender, s.dateofbirth, s.admissiondata, m.total, m.result from students s inner JOIN marks m on s.roll = m.roll;

SELECT students.\*, attendance.absents, attendance.presents FROM students INNER join attendance on students.roll = attendance.roll;

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.gender, students.dateofbirth, students.admissiondata, attendance.absents, attendance.presents from students INNER join attendance on students.roll = attendance.roll

## JOIN Three Tables

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.gender, students.dateofbirth, students.admissiondata, attendance.absents, attendance.presents, marks.total, marks.result from students INNER join attendance on students.roll = attendance.roll INNER join marks on students.roll = marks.roll;

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.email, s.gender, s.dateofbirth, s.admissiondata, a.absents, a.presents, m.total, m.result from students s INNER join attendance a on s.roll = a.roll inner join marks m on s.roll = m.roll

# MySQL LEFT JOIN Keyword

## MySQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).



### LEFT JOIN Syntax

SELECT column\_name(s) FROM table1 LEFT JOIN table2ON table1.column\_name = table2.column\_name;

// inner join

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.gender, students.dateofbirth, students.admissiondata, attendance.absents, attendance.presents from students INNER join attendance on students.roll = attendance.roll

// left join

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.gender, students.dateofbirth, students.admissiondata, attendance.absents, attendance.presents from students left join attendance on students.roll = attendance.roll;

# MySQL RIGHT JOIN Keyword

## MySQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



### RIGHT JOIN Syntax

SELECT column\_name(s) FROM table1 RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.gender, students.dateofbirth, students.admissiondata, attendance.absents, attendance.presents from students right join attendance on students.roll = attendance.roll;

# MySQL CROSS JOIN Keyword

## SQL CROSS JOIN Keyword

The CROSS JOIN keyword returns all records from both tables (table1 and table2).



### CROSS JOIN Syntax

SELECT column\_name(s) FROM table1 CROSS JOIN table2;

**Note:** CROSS JOIN can potentially return very large result-sets!

## MySQL CROSS JOIN Example

SELECT students.\* from students CROSS join attendance

Return large resultset (total records of students \* total records of attendance)

**Note:** The CROSS JOIN keyword returns all matching records from both tables whether the other table matches or not. So,

If you add a WHERE clause (if table1 and table2 has a relationship), the CROSS JOIN will produce the same result as the INNER JOIN clause:

SELECT students.\*, attendance.absents, attendance.presents from students CROSS join attendance WHERE students.roll = attendance.roll;

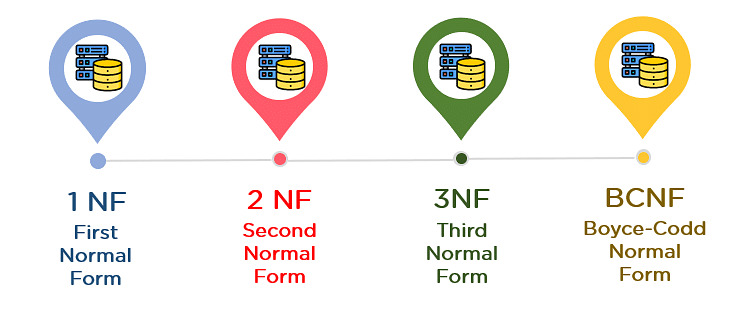
## What Is Normalization in SQL?

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies. It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.

In 1970 Edgar F. Codd defined the First Normal Form.

Now let's understand the types of Normal forms with the help of examples.

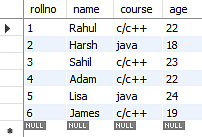


## 1st Normal Form (1NF)

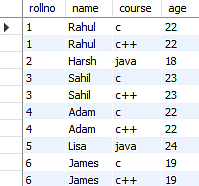
* A table is referred to as being in its First Normal Form if atomicity of the table is 1.
* Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute.
* The First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Now you will understand the First Normal Form with the help of an example.

Below is a students’ record table that has information about student roll number, student name, student course, and age of the student.



In the students record table, you can see that the course column has two values. Thus it does not follow the First Normal Form. Now, if you use the First Normal Form to the above table, you get the below table as a result.



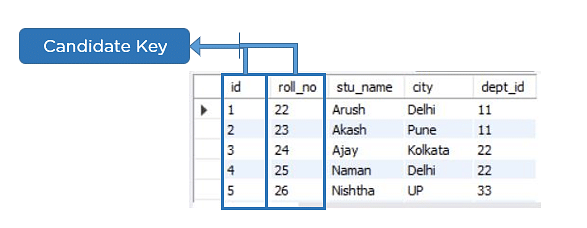
By applying the First Normal Form, you achieve atomicity, and also every column has unique values.

Before proceeding with the Second Normal Form, get familiar with Candidate Key and Super Key.

## Candidate Key

A candidate key is a set of one or more columns that can identify a record uniquely in a table, and YOU can use each candidate key as a [Primary Key.](https://www.simplilearn.com/tutorials/sql-tutorial/primary-key-in-sql)

Now, let’s use an example to understand this better.



### Super Key

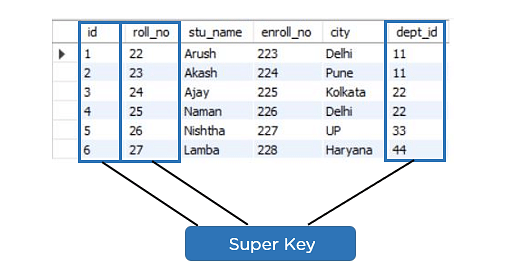
Super key is a set of over one key that can identify a record uniquely in a table, and the Primary Key is a subset of Super Key.

Let’s understand this with the help of an example.

### Super Key

Super key is a set of over one key that can identify a record uniquely in a table, and the Primary Key is a subset of Super Key.

Let’s understand this with the help of an example.

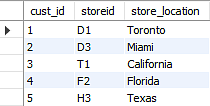


## Second Normal Form (2NF)

The first condition for the table to be in Second Normal Form is that the table has to be in First Normal Form. The table should not possess partial dependency. The partial dependency here means the proper subset of the candidate key should give a non-prime attribute.

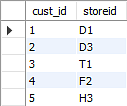
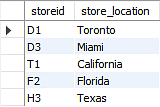
Now understand the Second Normal Form with the help of an example.

Consider the table Location:



The Location table possesses a composite primary key cust\_id, storeid. The non-key attribute is store\_location. In this case, store\_location only depends on storeid, which is a part of the primary key. Hence, this table does not fulfill the second normal form.

To bring the table to Second Normal Form, you need to split the table into two parts. This will give you the below tables:

As you have removed the partial functional dependency from the location table, the column store\_location entirely depends on the primary key of that table, storeid.

Now that you understood the 1st and 2nd Normal forms, you will look at the next part of this Normalization in SQL tutorial.

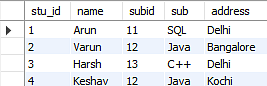
## Third Normal Form (3NF)

The first condition for the table to be in Third Normal Form is that the table should be in the Second Normal Form.

The second condition is that there should be no transitive dependency for non-prime attributes, which indicates that non-prime attributes (which are not a part of the candidate key) should not depend on other non-prime attributes in a table. Therefore, a transitive dependency is a functional dependency in which A → C (A determines C) indirectly, because of A → B and B → C (where it is not the case that B → A).

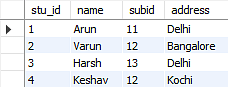
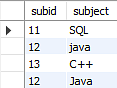
The third Normal Form ensures the reduction of data duplication. It is also used to achieve data integrity.

Below is a student table that has student id, student name, subject id, subject name, and address of the student as its columns.



In the above student table, stu\_id determines subid, and subid determines sub. Therefore, stu\_id determines sub via subid. This implies that the table possesses a transitive functional dependency, and it does not fulfill the third normal form criteria.

Now to change the table to the third normal form, you need to divide the table as shown below:

As you can see in both the tables, all the non-key attributes are now fully functional, dependent only on the primary key. In the first table, columns name, subid, and addresses only depend on stu\_id. In the second table, the sub only depends on subid.

## Boyce Codd Normal Form (BCNF)

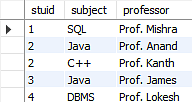
Boyce Codd Normal Form is also known as 3.5 NF. It is the superior version of 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to tackle certain types of anomalies which were not resolved with 3NF.

The first condition for the table to be in Boyce Codd Normal Form is that the table should be in the third normal form. Secondly, every Right-Hand Side (RHS) attribute of the functional dependencies should depend on the super key of that particular table.

For example :

You have a functional dependency X → Y. In the particular functional dependency, X has to be the part of the super key of the provided table.

Consider the below subject table:



The subject table follows these conditions:

Each student can enroll in multiple subjects.

Multiple professors can teach a particular subject.

For each subject, it assigns a professor to the student.

In the above table, student\_id and subject together form the primary key because using student\_id and subject; you can determine all the table columns

Another important point to be noted here is that one professor teaches only one subject, but one subject may have two professors.

Which exhibit there is a dependency between subject and professor, i.e. subject depends on the professor's name.

to

The table is in 1st Normal form as all the column names are unique, all values are atomic, and all the values stored in a particular column are of the same domain.

The table also satisfies the 2nd Normal Form, as there is no Partial Dependency.

And, there is no Transitive Dependency; hence, the table also satisfies the 3rd Normal Form.

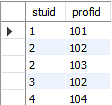
This table follows all the Normal forms except the Boyce Codd Normal Form.

As you can see stuid, and subject forms the primary key, which means the subject attribute is a prime attribute.

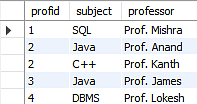
However, there exists yet another dependency - professor → subject

BCNF does not follow in the table as a subject is a prime attribute, the professor is a non-prime attribute.

To transform the table into the BCNF, you will divide the table into two parts. One table will hold stuid which already exists and the second table will hold a newly created column profid.



And in the second table will have the columns profid, subject, and professor, which satisfies the BCNF.



With this, you have reached the conclusion of the ‘Normalization in SQL’ tutorial.

# MySQL Self Join

A self join is a regular join, but the table is joined with itself.

### Self Join Syntax

SELECT column\_name(s) FROM table1 T1, table1 T2 WHERE condition;

T1 and T2 are different table aliases for the same table.

## MySQL Self Join Example

SELECT s1.fname as "Student s1", s2.fname as "Student s2", s1.city as "City s1" from students s1, students s2 WHERE s1.roll <> s2.roll and s1.city = s2.city;

# MySQL UNION Operator

## The MySQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

### UNION Syntax

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

CREATE TABLE students1 as SELECT \* from students

SELECT \* from students UNION SELECT \* from students1

### UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT \* from students UNION ALL SELECT \* from students1

SELECT \* from students WHERE city = 'Rajkot' UNION ALL SELECT \* from students1 WHERE city = 'Rajkot';

SELECT roll, fname, lname, city from students WHERE city = 'Rajkot'

UNION ALL

SELECT roll, fname, lname, city from students1 WHERE city = 'Rajkot';

# MySQL GROUP BY Statement

## The MySQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

### GROUP BY Syntax

SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)ORDER BY column\_name(s);

Select \* from students

SELECT DISTINCT(city) from students

SELECT city, COUNT(city) from students GROUP by(city)

SELECT city, COUNT(city) from students GROUP by(city) ORDER by COUNT(city);

SELECT city, COUNT(city) from students GROUP by(city) ORDER by COUNT(city) desc;

## GROUP BY With JOIN Example

SELECT students.roll, students.fname, students.lname, students.city, students.gender, students.phone, students.email, marks.total, marks.result from students INNER join marks on students.roll = marks.roll;

SELECT students.roll, students.fname, students.lname, students.city, students.gender, students.phone, students.email, marks.total, marks.result, count(marks.roll) as "Attempts" from students INNER join marks on students.roll = marks.roll GROUP by (marks.roll);

# MySQL HAVING Clause

## The MySQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

### HAVING Syntax

SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

SELECT students.roll, students.fname, students.lname, students.city, students.gender, students.phone, students.email, marks.total, marks.result, count(marks.roll) as "Attempts" from students INNER join marks on students.roll = marks.roll GROUP by (marks.roll) having count(marks.roll) > 1;

SELECT students.roll, students.fname, students.lname, students.city, students.gender, students.phone, students.email, marks.total, marks.result, count(marks.roll) as "Attempts" from students INNER join marks on students.roll = marks.roll GROUP by (marks.roll) having count(marks.roll) >= 3;

# MySQL EXISTS Operator

## The MySQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

SELECT column\_name(s) FROM table\_name WHERE EXISTS  
(SELECT column\_name FROM table\_name WHERE condition);

SELECT \* from students WHERE EXISTS (SELECT roll from marks WHERE students.roll = marks.roll and marks.result = 'pass')

SELECT \* from students WHERE not EXISTS (SELECT roll from marks WHERE students.roll = marks.roll and marks.result = 'pass');

# MySQL ANY and ALL Operators

## The MySQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

## The ANY Operator

The ANY operator:

* returns a boolean value as a result
* returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

### ANY Syntax

SELECT column\_name(s) FROM table\_name WHERE column\_name operator ANY (SELECT column\_name FROM table\_name  WHERE condition);

SELECT \* from students WHERE roll = any (SELECT roll FROM marks WHERE result = 'pass');

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

## The ALL Operator

The ALL operator:

* returns a boolean value as a result
* returns TRUE if ALL of the subquery values meet the condition
* is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

### ALL Syntax With SELECT

SELECT ALL column\_name(s) FROM table\_name WHERE condition;

SELECT all roll, fname, lname, city from students

### ALL Syntax With WHERE or HAVING

SELECT column\_name(s) FROM table\_name WHERE column\_name operator ALL (SELECT column\_name  FROM table\_name  WHERE condition);

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

SELECT \* from students WHERE roll = all (SELECT roll from marks WHERE result = 'pass')