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

## 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.

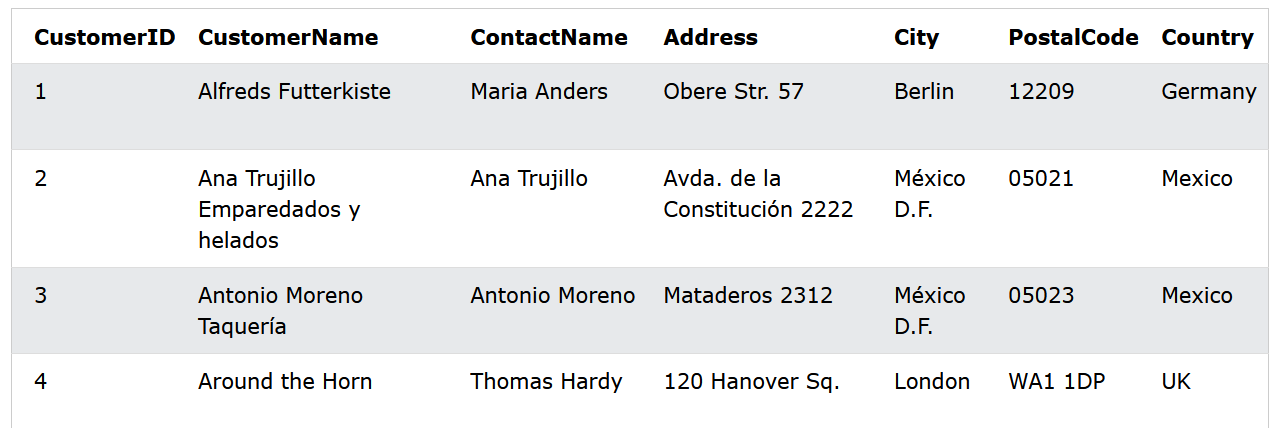
## What is a Database Table?

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:



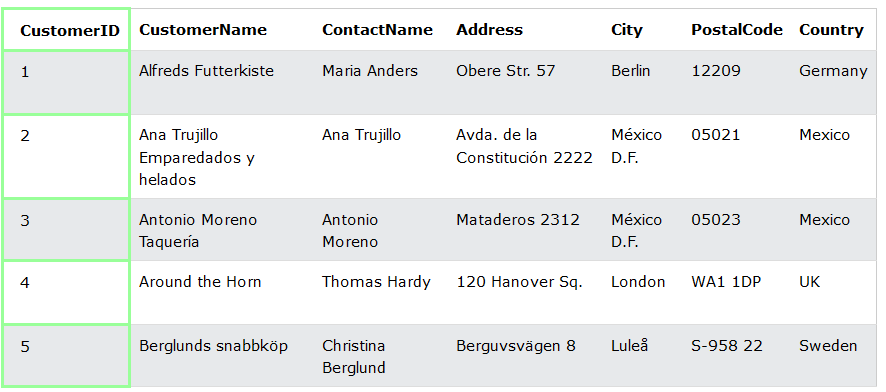
The columns in the "Customers" table above are: CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country. The table has 4 records (rows).

## 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.

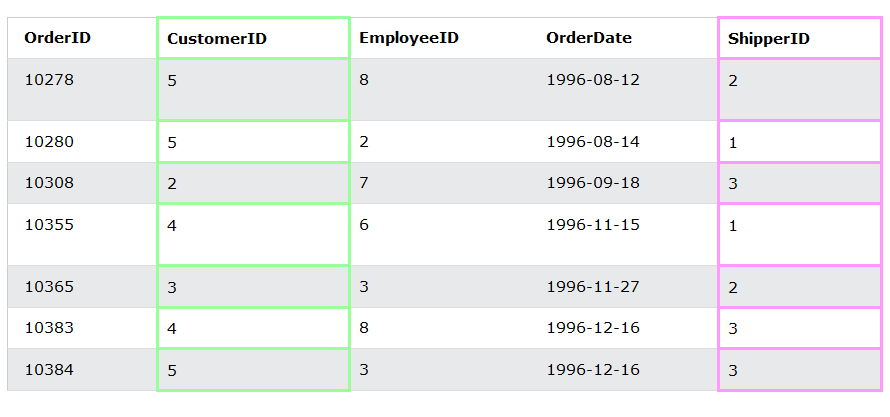
Look at the following three tables "Customers", "Orders", and "Shippers" from the Northwind database:

### Customers Table



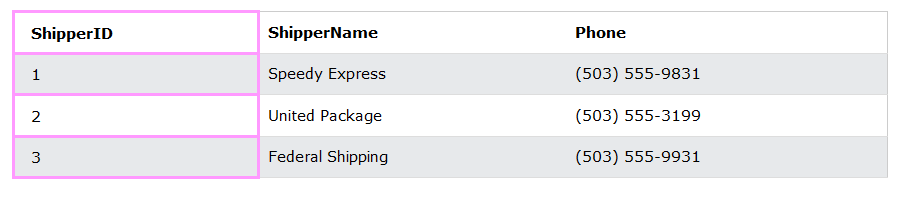
The relationship between the "Customers" table and the "Orders" table is the CustomerID column:

### 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.

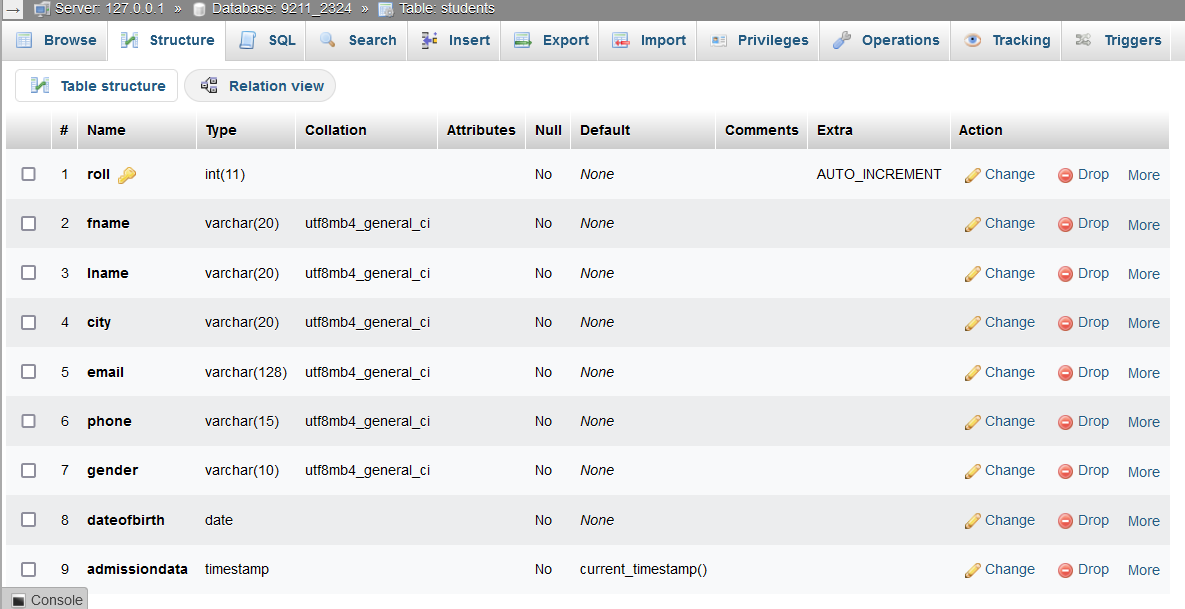
Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

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

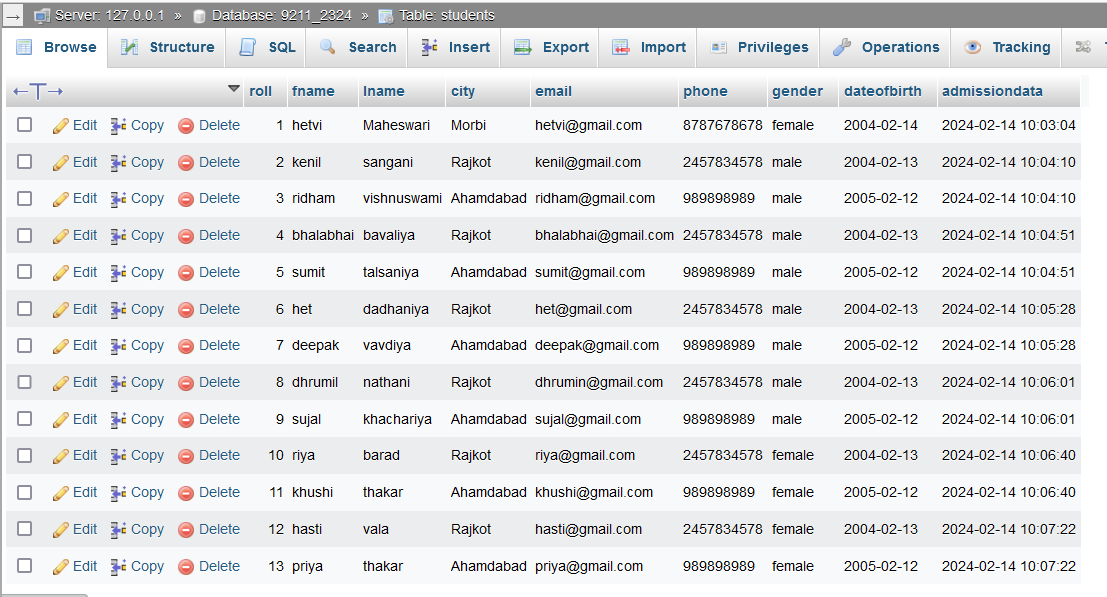
## Some of The Most Important SQL Commands

* 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

Create Database and Table as like following



And add some data as following



# 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 \* from students;

SELECT roll, fname, lname, city 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;

The following SQL statement counts and returns the number of different (distinct) city in the "Students" table:

SELECT count(DISTINCT city) 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\_name  
WHERE condition;

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

SELECT \* FROM students WHERE roll = 1

SELECT \* FROM students WHERE roll > 5;

SELECT \* FROM students WHERE not roll > 5;

SELECT \* FROM students WHERE city = 'rajkot';

SELECT \* FROM students WHERE city <> 'rajkot';

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 = 'morbi' or city = 'Rajkot';

### NOT Syntax

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

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

## Combining AND, OR and NOT

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

SELECT \* from students WHERE roll = 1 and city = 'rajkot' or city = 'morbi'

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

# 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

The following SQL statement selects all customers from the "students" table, sorted DESCENDING by the "fname" column:

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 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, ...);

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 (fname, lname, city, email, phone, gender, dateofbirth) VALUES ('Demo', 'text', 'example', 'demo@example.com', '9876543210', 'male', '2001-01-01')

**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, email, phone) VALUES ('Demo', 'text', 'example', 'demo@example.com', '9876543210')

# 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!

INSERT into students (fname, lname, city, email, phone) VALUES ('Demo', 'text', 'example', 'demo@example.com', '9876543210')

SELECT \* FROM `students` WHERE gender = '';

SELECT \* FROM `students` WHERE gender = 'NULL';

SELECT \* FROM `students` WHERE gender is null;

## 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;

SELECT \* FROM `students` WHERE gender is null;

### IS NOT NULL Syntax

SELECT column\_namesFROM table\_name WHERE column\_name IS NOT 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, ...

**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!WHERE condition;

UPDATE students set gender = 'male' WHERE roll = 15 or roll = 16

UPDATE students set city = 'rajkot'

UPDATE students set city = 'surat' WHERE roll > 10

UPDATE students set city = 'ahamdabad' WHERE roll > 5 and roll < 10;

## UPDATE Multiple Records

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

UPDATE students set gender = 'female', dateofbirth = '2001-01-15' WHERE roll >= 15

## Update Warning!

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

# 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;

What if we want to select records 6-10 (inclusive)?

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

The SQL query below says "return only 5 records, start on record 6 (OFFSET 5)":

SELECT \* FROM `students` limit 5 OFFSET 5;

SELECT \* FROM `students` limit 5 OFFSET 10;

SELECT \* FROM students WHERE city = 'rajkot' LIMIT 10 ;

SELECT \* from students LIMIT 5 OFFSET 10

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

# 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 min(roll) FROM students;

SELECT max(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 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!

Backup all the data of students table to studentsBackup table

create table studentsBackup as SELECT \* FROM students

## SQL DELETE Example

DELETE from students WHERE ROLL = 1

DELETE from students WHERE city = 'rajkot'

## Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM table\_name;

DELETE from students

// restore

INSERT into students SELECT \* from studentsbackup

// when you delete all the data from table the auto increment is not reset to zero

INSERT INTO `students` (`roll`, `fname`, `lname`, `city`, `email`, `phone`, `gender`, `dateofbirth`, `admissiondata`) VALUES (NULL, 'brijesh', 'sinojiya', 'Morbi', 'demo@gmail.com', '998899889900', 'male', '2004-02-13', current\_timestamp());

A new record get a roll number as you leave before delete all the data

Use truncate table to delete all the data from table and also reset all auto increments

TRUNCATE TABLE students

INSERT INTO `students` (`roll`, `fname`, `lname`, `city`, `email`, `phone`, `gender`, `dateofbirth`, `admissiondata`) VALUES (NULL, 'brijesh', 'sinojiya', 'Morbi', 'demo@gmail.com', '998899889900', 'male', '2004-02-13', current\_timestamp());

Tuncate table students

Insert into students select \* from studentsbackup

DROP TABLE studentsbackup

# 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!

### 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 'a%'

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 '\_\_\_\_';

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 = 'surat' or city = 'baroda'

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

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

# 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;

SELECT \* from students WHERE fname BETWEEN 'bhalabhai' and 'riya';

SELECT \* from students WHERE fname not BETWEEN 'bhalabhai' and 'riya';

SELECT \* FROM `students` WHERE dateofbirth BETWEEN '2001-01-01' and '2004-12-31'

SELECT \* FROM `students` WHERE dateofbirth not BETWEEN '2001-01-01' and '2004-12-31';

# 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 roll, fname, lname, email, city, phone, dateofbirth from students

SELECT roll as "Roll Number", fname as "First Name", lname as "Last Name", email as "Email Address", city as HomeTown, phone as 'Phone Number', dateofbirth from students;

SELECT roll "Roll Number", fname "First Name", lname "Last Name", email "Email Address", city HomeTown, phone 'Phone Number', dateofbirth from students;

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

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

**The following SQL statement is the same as above, but without aliases:**

SELECT students.roll, students.fname, students.lname, students.city, students.email, students.phone, students.gender, students.dateofbirth, students.admissiondata, marks.total, marks.result FROM students, marks WHERE students.roll = 1 and students.roll = marks.roll

**The following SQL statement is the same as above, but with aliases:**

SELECT s.roll, s.fname, s.lname, s.city, s.email, s.phone, s.gender, s.dateofbirth, s.admissiondata, m.total, m.result FROM students as s, marks as m WHERE s.roll = 1 and s.roll = m.roll;

SELECT s.roll, s.fname, s.lname, s.city, s.email, s.phone, s.gender, s.dateofbirth, s.admissiondata, m.total, m.result FROM students s, marks m WHERE s.roll = 1 and s.roll = m.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.email, s.phone, s.gender, s.dateofbirth, s.admissiondata, 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.email, s.phone, s.gender, s.dateofbirth, s.admissiondata, a.absents, a.presents, (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.email, s.phone, s.gender, s.dateofbirth, s.admissiondata, a.absents, a.presents, (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.

Let's look at a selection from the "Orders" table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

Then, look at a selection from the "Customers" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

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

## 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
*       

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

# 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 students.roll, students.fname, students.lname, students.dateofbirth, marks.total, marks.result from students INNER join marks on students.roll = marks.roll

Without alias

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

With alias

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

## JOIN Three Tables

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

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

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

SELECT s.roll, s.fname, s.lname, s.city, s.email, s.phone, s.gender, s.dateofbirth, m.total, m.result from students s left join marks m on **s.roll** = m.roll

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders) even no match found mysql replace required values with NULL keyword.

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

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

**Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders) even no match found mysql replace required values with NULL keyword.

.

# 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**!

SELECT \* FROM students CROSS join marks

**Note:** The CROSS JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

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 \* FROM students CROSS join marks WHERE students.roll = marks.roll;

# MySQL Self Join

## 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.

SELECT s1.roll, s1.fname, s1.lname, s1.city, s1.email, s1.phone FROM students s1, students s2 WHERE s1.roll <> s2.roll and s1.city = 'Rajkot';

SELECT s1.roll, s1.fname, s1.lname, s1.city, s2.roll, s2.fname, s2.lname, s2.city 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;

### UNION ALL Syntax

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

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

create table students1 as SELECT \* from students

SELECT \* FROM students

UNION

SELECT \* FROM students1

SELECT \* FROM students

UNION ALL

SELECT \* FROM students1

SELECT roll, fname, lname, city FROM students

UNION ALL

SELECT roll, fname, lname, city FROM students1;

SELECT roll, fname, lname, city FROM students

UNION ALL

SELECT roll, fname, lname FROM students1;

#1222 - The used SELECT statements have a different number of columns

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.

SELECT DISTINCT city FROM students

### GROUP BY Syntax

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

SELECT city, count(roll) from students GROUP by(city)

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

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

## GROUP BY With JOIN Example

SELECT students.roll, students.fname, students.lname, students.city, students.gender, students.phone, students.email, students.admissiondata, 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, students.admissiondata, marks.total, marks.result, count(marks.roll) as "Exam Attempt" 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.

SELECT students.roll, students.fname, students.lname, students.city, students.gender, students.phone, students.email, students.admissiondata, marks.total, marks.result, count(marks.roll) as "Exam Attempt" from students inner join marks on students.roll = marks.roll **GROUP by** (marks.roll) **where** count(marks.roll) > 2;

SELECT students.roll, students.fname, students.lname, students.city, students.gender, students.phone, students.email, students.admissiondata, marks.total, marks.result, count(marks.roll) as "Exam Attempt" from students inner join marks on students.roll = marks.roll GROUP by (marks.roll) having count(marks.roll) > 2;

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

SELECT city, COUNT(roll) from students GROUP by (city) HAVING COUNT(roll) > 3;

SELECT city, COUNT(roll) from students GROUP by (city) HAVING COUNT(roll) > 3 ORDER by (COUNT(roll));

# 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.

### EXISTS Syntax

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

SELECT roll from marks WHERE result = 'pass';

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

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

# 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);

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

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

## 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;

### 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 <=).

This will of course return FALSE because the Quantity column has many different values (not only the value of pass):

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

# MySQL INSERT INTO SELECT Statement

## The MySQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables matches.

**Note:** The existing records in the target table are unaffected.

### INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

INSERT INTO table2 SELECT \* FROM table1WHERE condition;

INSERT into students1 SELECT \* FROM students

INSERT into students1 SELECT \* FROM students WHERE City = 'rajkot'

Copy only some columns from one table into another table:

INSERT INTO *table2* (*column1*, *column2*, *column3*, ...) SELECT *column1*, *column2*, *column3*, ... FROM *table1* WHERE *condition*;

INSERT into students1 (Fname, lname, city) SELECT fname, lname, city from students

# MySQL CASE Statement

## The MySQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

## CASE Syntax

CASE  
    WHEN condition1 THEN result1  
    WHEN condition2 THEN result2  
    WHEN conditionN THEN resultN  
    ELSE result  
END;

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

SELECT roll, fname, lname, city, email, phone, gender, case

WHEN gender = 'male' then 'Blue'

WHEN gender = 'female' then 'Pink'

else 'Black'

end as "Bag Color",

dateofbirth from students

SELECT roll, fname, lname, city, case

WHEN city = 'Rajkot' THEN 'Home Town'

WHEN city = 'Ahamdabad' THEN 'Far Town'

WHEN city = 'Gandhinagar' THEN 'Far Town'

WHEN city = 'Bhuj' THEN 'Much Far Town'

WHEN city = 'Baroda' THEN 'Too Much Far Town'

WHEN city = 'Surat' THEN 'Remote Town'

end as "Distance",email, phone, gender,dateofbirth from students;

# MySQL NULL Functions

## MySQL IFNULL() and COALESCE() Functions

SELECT roll, absents, presents, (absents +presents) as "Total Days" FROM attendance GROUP by (roll);

## MySQL IFNULL() Function

The MySQL [IFNULL()](https://www.w3schools.com/mysql/func_mysql_ifnull.asp) function lets you return an alternative value if an expression is NULL.

The example below returns 0 if the value is NULL:

SELECT roll, absents, presents, (ifnull(absents, 0) + ifnull(presents, 0)) as "Total Days" FROM attendance GROUP by (roll);

## MySQL COALESCE() Function

SELECT roll, absents, presents, (COALESCE(absents, 0) + COALESCE(presents, 0)) as "Total Days" FROM attendance GROUP by (roll);

# MySQL Comments

## MySQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

## Single Line Comments

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

-- this is single line comment in MYSQL

SELECT \* FROM `students`

SELECT \* FROM `students`; -- this is another example of single line comment

## Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored.

The following example uses a multi-line comment as an explanation:

/\*Select all the columns  
of all the records  
in the Customers table:\*/  
SELECT \* FROM Customers;

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

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

# SQL Commands

* SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
* SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

## Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

### 1. Data Definition Language (DDL)

* DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

**CREATE** It is used to create a new table in the database.

**DROP:** It is used to delete both the structure and record stored in the table.

**ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table

### 2. Data Manipulation Language

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

**INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**UPDATE:** This command is used to update or modify the value of a column in the table.

**DELETE:** It is used to remove one or more row from a table.

### 3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**Grant:** It is used to give user access privileges to a database.

**Revoke:** It is used to take back permissions from the user.

### 4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**Commit:** Commit command is used to save all the transactions to the database.

**Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction

### 5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

* SELECT

**SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

# MySQL CREATE DATABASE Statement

## The MySQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

### Syntax

CREATE DATABASE databasename;

## CREATE DATABASE Example

CREATE DATABASE testDB;

show DATABASES

create DATABASE example9211

show DATABASES

# MySQL DROP DATABASE Statement

## The MySQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.

### Syntax

DROP DATABASE databasename;

**Note:** Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

show DATABASES;

drop DATABASE example9211

show DATABASES;

**Tip:** Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

# MySQL CREATE TABLE Statement

## The MySQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

### Syntax

CREATE TABLE table\_name ( column1 datatype, column2 datatype, column3 datatype, ...);

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

**Tip:** For an overview of the available data types, go to our complete [Data Types Reference](https://www.w3schools.com/mysql/mysql_datatypes.asp).

## MySQL CREATE TABLE Example

create TABLE persons (personid int AUTO\_INCREMENT PRIMARY key, fname varchar(20) not null, lname varchar(20), city varchar(20), gender varchar(10), email varchar(128), phone varchar(15), createdat timestamp DEFAULT CURRENT\_TIMESTAMP)

## Create Table Using Another Table

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

CREATE TABLE new\_table\_name AS  
    SELECT column1, column2,...  
    FROM existing\_table\_name  
    WHERE ....;

create TABLE persons\_backup as SELECT \* from persons

create TABLE persons\_backup1 as SELECT personid, fname, lname, city from persons

# MySQL DROP TABLE Statement

## The MySQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

### Syntax

DROP TABLE table\_name;

drop TABLE persons\_backup;

**Note:** Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

## MySQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

### Syntax

TRUNCATE TABLE table\_name;

create table studentsBackup as SELECT \* FROM students

TRUNCATE TABLE studentsbackup

\* truncate table reset auto increment field to 1

# MySQL ALTER TABLE Statement

## MySQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

## ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

ALTER TABLE table\_name  
ADD column\_name datatype;

ALTER TABLE persons add COLUMN dateofbirth date

## ALTER TABLE - MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype;

alter TABLE persons MODIFY COLUMN phone varchar(20)

## ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name  
DROP COLUMN column\_name;

ALTER TABLE persons DROP COLUMN dateofbirth

ALTER TABLE persons add COLUMN dateofbirth date AFTER city

# MySQL Constraints

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

## Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

### Syntax

CREATE TABLE table\_name (  
    column1 datatype *constraint*,  
    column2 datatype *constraint*,  
    column3 datatype *constraint*,  
    ....  
);

## MySQL Constraints

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

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

* [NOT NULL](https://www.w3schools.com/mysql/mysql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/mysql/mysql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/mysql/mysql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/mysql/mysql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/mysql/mysql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/mysql/mysql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/mysql/mysql_create_index.asp) - Used to create and retrieve data from the database very quickly

# MySQL NOT NULL Constraint

## MySQL NOT NULL Constraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

## NOT NULL on CREATE TABLE

CREATE table persons (person\_id int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), city varchar(20), email varchar(20))

INSERT into persons (fname, lname, city, email) values ('Ridham', 'Vishnuswami', 'Rajkot', 'Ridham@gmail.com')

INSERT into persons (fname, lname, city, email) values ('Ridham', 'Vishnuswami', null, null);

drop TABLE persons

CREATE table persons (person\_id int AUTO\_INCREMENT PRIMARY key, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null, email varchar(20) not null)

INSERT into persons (fname, lname, city, email) values ('Ridham', 'Vishnuswami', 'Rajkot', 'Ridham@gmail.com')

INSERT into persons (fname, lname, city, email) values ('Ridham', 'Vishnuswami', null, null);

#1048 - Column 'city' cannot be null

## NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

ALTER TABLE persons add COLUMN age int

ALTER TABLE persons MODIFY COLUMN age int not null

INSERT into persons (fname, lname, city, email, age) values ('Ridham', 'Vishnuswami', 'Rajkot', 'rv@gmail.com', null);

#1048 - Column 'age' cannot be null

# MySQL UNIQUE Constraint

## MySQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

## UNIQUE Constraint on CREATE TABLE

drop TABLE persons

CREATE table persons (person\_id int AUTO\_INCREMENT PRIMARY key, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null, email varchar(128) not null)

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'kenil', 'sangani', 'rajkot', 'kenil@gmail.com'), (NULL, 'ridham', 'vishnuswami', 'rajkot', 'ridham@gmail.com');

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', 'kenil@gmail.com'), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', 'ridham@gmail.com');

drop TABLE persons

CREATE table persons (person\_id int AUTO\_INCREMENT PRIMARY key, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null, email varchar(128) not null, **UNIQUE(email)**)

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'kenil', 'sangani', 'rajkot', 'kenil@gmail.com'), (NULL, 'ridham', 'vishnuswami', 'rajkot', 'ridham@gmail.com');

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', 'kenil@gmail.com'), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', 'ridham@gmail.com');

#1062 - Duplicate entry 'kenil@gmail.com' for key **'email'**

drop TABLE persons

Add unique with named constraint

CREATE table persons (person\_id int AUTO\_INCREMENT PRIMARY key, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null, email varchar(128) not null, **CONSTRAINT unq\_email UNIQUE(email)**)

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'kenil', 'sangani', 'rajkot', 'kenil@gmail.com'), (NULL, 'ridham', 'vishnuswami', 'rajkot', 'ridham@gmail.com');

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', 'kenil@gmail.com'), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', 'ridham@gmail.com');

#1062 - Duplicate entry 'kenil@gmail.com' for key **'unq\_email'**

## UNIQUE Constraint on ALTER TABLE

ALTER table persons drop CONSTRAINT **unq\_email**

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', 'kenil@gmail.com'), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', 'ridham@gmail.com');

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', 'kenil@gmail.com'), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', 'ridham@gmail.com');

Select \* from persons

alter TABLE persons add CONSTRAINT unq\_email UNIQUE(email)

#1062 - Duplicate entry 'kenil@gmail.com' for key 'unq\_email'

TRUNCATE TABLE persons

alter TABLE persons add CONSTRAINT unq\_email UNIQUE(email)

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', 'kenil@gmail.com'), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', 'ridham@gmail.com');

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', 'kenil@gmail.com'), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', 'ridham@gmail.com');

## DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL

ALTER table persons drop CONSTRAINT **unq\_email**

The major drawback of unique constraint is its accept null values

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', null), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', null);

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`, `email`) VALUES (NULL, 'sumit', 'talsaniya', 'rajkot', null), (NULL, 'bhalabahi', 'bavaliya', 'rajkot', null);

## Unique Constraint allows multiple columns to create unique key

create TABLE listofVilalges (villageid int AUTO\_INCREMENT PRIMARY key,villegeName varchar(20), cityname varchar(20), taluka varchar(20), district varchar(20), state varchar(20), country varchar(20), pincode int**, CONSTRAINT unq\_villagename UNIQUE(villegename)**)

INSERT INTO `listofvilalges` (`villageid`, `villegeName`, `cityname`, `taluka`, `district`, `state`, `country`, `pincode`) VALUES (NULL**, 'Navagam'**, 'Rajkot', 'Rajkot', 'Rajkot', 'Guajarat', 'India', '360009');

INSERT INTO `listofvilalges` (`villageid`, `villegeName`, `cityname`, `taluka`, `district`, `state`, `country`, `pincode`) VALUES (NULL, **'Navagam'**, 'Gondal', 'Gondal', 'Rajkot', 'Guajarat', 'India', '360009');

#1062 - Duplicate entry 'Navagam' for key **'unq\_villagename'**

ALTER table listofvilalges drop CONSTRAINT unq\_villagename

ALTER TABLE listofvilalges add CONSTRAINT unq\_villageName UNIQUE(villegeName, cityname, taluka)

INSERT INTO `listofvilalges` (`villageid`, `villegeName`, `cityname`, `taluka`, `district`, `state`, `country`, `pincode`) VALUES (NULL, **'Navagam'**, **'Gondal', 'Gondal'**, 'Rajkot', 'Guajarat', 'India', '360009');

INSERT INTO `listofvilalges` (`villageid`, `villegeName`, `cityname`, `taluka`, `district`, `state`, `country`, `pincode`) VALUES (NULL, **'Navagam'**, **'Gondal', 'Gondal'**, 'Rajkot', 'Guajarat', 'India', '360009');

#1062 - Duplicate entry 'Navagam-Gondal-Gondal' for key 'unq\_villageName'

INSERT INTO `listofvilalges` (`villageid`, `villegeName`, `cityname`, `taluka`, `district`, `state`, `country`, `pincode`) VALUES (NULL, 'Navagam', 'patan', 'patan', 'Rajkot', 'Guajarat', 'India', '360009');

# MySQL PRIMARY KEY Constraint

## MySQL PRIMARY KEY Constraint

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

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

## PRIMARY KEY on CREATE TABLE

drop TABLE persons

CREATE table persons (person\_id int, fname varchar(20), lname varchar(20), city varchar(20))

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`) VALUES ('1', 'brijesh', 'sinojiya', 'Morbi'), ('1', 'ansh', 'amrutiya', 'Ahamdabad');

Select \* from persons

drop TABLE persons

CREATE table persons (person\_id int **PRIMARY key**, fname varchar(20), lname varchar(20), city varchar(20))

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`) VALUES ('1', 'brijesh', 'sinojiya', 'Morbi'), ('1', 'ansh', 'amrutiya', 'Ahamdabad');

#1062 - Duplicate entry '1' for key 'PRIMARY'

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (ID)  
);

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)  
);

drop TABLE persons

create TABLE persons (person\_id int, fname varchar(20), lname varchar(20), city varchar(20), **CONSTRAINT pk\_pid PRIMARY key (person\_id)**)

## DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

ALTER TABLE Persons DROP PRIMARY KEY;

## PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

**Note:** If you use ALTER TABLE to add a primary key, the primary key column(s) must have been declared to not contain NULL values (when the table was first created).

ALTER TABLE persons add CONSTRAINT pk\_pid PRIMARY key(person\_id)

alter table persons drop PRIMARY key

ALTER TABLE persons add CONSTRAINT pk\_pid PRIMARY key(person\_id, fname)

INSERT INTO `persons` (`person\_id`, `fname`, `lname`, `city`) VALUES ('1', 'Bhavdeep', 'sinojiya', 'Morbi'), ('1', 'kenil', 'Sangani', 'Ahamdabad');

# MySQL FOREIGN KEY Constraint

## MySQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/mysql/mysql_primarykey.asp) in another table.

The table with the **foreign key** is called the **child table**, and the table with the **primary key** is called the referenced or **parent table**.

Look at the following two tables:

### Persons Table

|  |  |  |  |
| --- | --- | --- | --- |
| **PersonID (Primary Key)** | **LastName** | **FirstName** | **Age** |
| **1** | Hansen | Ola | 30 |
| **2** | Svendson | Tove | 23 |
| **3** | Pettersen | Kari | 20 |
| **4** | Ridham | Vishniswami | 19 |

### Orders Table

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID (Foreign key)** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |
| 5 | 112233 | 4 |
| 6 | 1234543 | 4 |
| 7 | 12322 | 1 |
| 8 | 12121 | 5 |

Data row number 8 can not inserted because perosn id 5 is not exist in persons table

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

## FOREIGN KEY on CREATE TABLE

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

create TABLE feesInformation (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdata date, amount int, paymentmode varchar(20))

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (1, '2024-02-03', 12500, 'cash')

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (41, '2024-02-03', 12500, 'cash');

drop table feesinformation

create TABLE feesInformation (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdata date, amount int, paymentmode varchar(20), FOREIGN key(roll) REFERENCES students(roll))

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (1, '2024-02-03', 12500, 'cash')

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (51, '2024-02-03', 12500, 'cash');

#1452 - Cannot add or update a child row: a foreign key constraint fails (`9211\_2324`.`feesinformation`, CONSTRAINT `feesinformation\_ibfk\_1` FOREIGN KEY (`roll`) REFERENCES `students` (`roll`))

Drop table feesinformation

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

create TABLE feesInformation (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdata date, amount int, paymentmode varchar(20), CONSTRAINT **fk\_roll** FOREIGN key(roll) REFERENCES students(roll))

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (1, '2024-02-03', 12500, 'cash')

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (31, '2024-02-03', 12500, 'cash');

#1452 - Cannot add or update a child row: a foreign key constraint fails (`9211\_2324`.`feesinformation`, CONSTRAINT `**fk\_roll**` FOREIGN KEY (`roll`) REFERENCES `students` (`roll`))

## FOREIGN KEY on ALTER TABLE

drop TABLE feesinformation

create TABLE feesInformation (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdata date, amount int, paymentmode varchar(20))

**Graphically add foreign key**

Go to Table structure and open relation view add constraint name, select column

Select primary table and column from table for reference and press save

**Foreign key with query**

ALTER TABLE feesinformation add CONSTRAINT fk\_roll FOREIGN key(roll) REFERENCES students(roll)

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder

## DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

ALTER TABLE Orders DROP FOREIGN KEY FK\_PersonOrder;

ALTER TABLE feesinformation drop CONSTRAINT fk\_roll

Alter table feesinformation drop foreign key

**You cannot delete data from parent table if any of the record referred by any of the foreign key column.**

# MySQL CHECK Constraint

## MySQL 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 column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

## CHECK on CREATE TABLE

create TABLE feesInformation (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdata date, amount int, **check(amount >=0)**, paymentmode varchar(20), CONSTRAINT fk\_roll FOREIGN key(roll) REFERENCES students(roll))

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (1,

'2024-02-03',-12500, 'cash');

drop TABLE feesinformation

create TABLE feesInformation (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdata date, amount int, paymentmode varchar(20), CONSTRAINT fk\_roll FOREIGN key(roll) REFERENCES students(roll), CONSTRAINT check\_amount CHECK (amount >= 0))

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (1,

'2024-02-03',-12500, 'cash');

#4025 - CONSTRAINT `check\_amount` failed for `9211\_2324`.`feesinformation`

## CHECK on ALTER TABLE

ALTER TABLE feesinformation add CONSTRAINT pay\_mode CHECK (paymentmode = '**Cash**' or paymentmode = '**cheque**' or paymentmode = '**online**')

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (1, '2024-02-03',12500, 'cash');

INSERT into feesinformation (roll, paymentdata, amount, paymentmode) values (1, '2024-02-03',12500, 'baki');

#4025 - CONSTRAINT `pay\_mode` failed for `9211\_2324`.`feesinformation`

## DROP a CHECK Constraint

alter TABLE feesinformation drop CONSTRAINT pay\_mode

# MySQL DEFAULT Constraint

## MySQL DEFAULT Constraint

The DEFAULT constraint is used to set a default value for a column.

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

## DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);

CREATE table persons (person\_id int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), city varchar(20) DEFAULT 'Rajkot')

INSERT into persons (fname, lname) VALUES ('kenil', 'sangani')

ALTER TABLE persons add COLUMN created\_at timestamp DEFAULT CURRENT\_TIMESTAMP

INSERT into persons (fname, lname) VALUES ('kenil', 'sangani')

ALTER TABLE persons ALTER lname set DEFAULT 'NA'

INSERT into persons (fname) VALUES ('kenil')

## DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

ALTER TABLE Persons ALTER City DROP DEFAULT;

alter TABLE persons ALTER lname DROP DEFAULT

INSERT into persons (fname) VALUES ('kenil')

Warning: #1364 Field 'lname' doesn't have a default value

# MySQL CREATE INDEX Statement

## MySQL CREATE INDEX Statement

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

Indexes are used to retrieve data from the database more quickly than otherwise. 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, only create indexes on columns that will be frequently searched against.

### CREATE INDEX Syntax

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

CREATE INDEX index\_name ON table\_name (column1, column2, ...);

### CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name ON table\_name (column1, column2, ...);

CREATE INDEX index\_fname on students(fname)

drop INDEX index\_fname on students;

CREATE UNIQUE INDEX index\_fname on students(fname)

## DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

ALTER TABLE table\_nameDROP INDEX index\_name;

# MySQL Data Types

The data type of a column defines what value the column can hold: **integer, character, money, date and time, binary,** and so on.

## MySQL Data Types (Version 8.0)

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

In MySQL there are three main data types: string, numeric, and date and time.

### String Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHAR(size) | A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1 |
| VARCHAR(size) | A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535 |
| BINARY(size) | Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1 |
| VARBINARY(size) | Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes. |
| TINYBLOB | For BLOBs (Binary Large OBjects). Max length: 255 bytes |
| TINYTEXT | Holds a string with a maximum length of 255 characters |
| TEXT(size) | Holds a string with a maximum length of 65,535 bytes |
| BLOB(size) | For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data |
| MEDIUMTEXT | Holds a string with a maximum length of 16,777,215 characters |
| MEDIUMBLOB | For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data |
| LONGTEXT | Holds a string with a maximum length of 4,294,967,295 characters |
| LONGBLOB | For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data |
| ENUM(val1, val2, val3, ...) | A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them |
| SET(val1, val2, val3, ...) | A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list |

### Numeric Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| BIT(size) | A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1. |
| TINYINT(size) | A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255) |
| BOOL | Zero is considered as false, nonzero values are considered as true. |
| BOOLEAN | Equal to BOOL |
| SMALLINT(size) | A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255) |
| MEDIUMINT(size) | A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255) |
| INT(size) | A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255) |
| INTEGER(size) | Equal to INT(size) |
| BIGINT(size) | A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255) |
| FLOAT(size, d) | A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions |
| FLOAT(p) | A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE() |
| DOUBLE(size, d) | A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter |
| DOUBLE PRECISION(size, d) |  |
| DECIMAL(size, d) | An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0. |
| DEC(size, d) | Equal to DECIMAL(size,d) |

**Note:** All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

### Date and Time Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| DATE | A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME(fsp) | A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time |
| TIMESTAMP(fsp) | A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT\_TIMESTAMP and ON UPDATE CURRENT\_TIMESTAMP in the column definition |
| TIME(fsp) | A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59' |
| YEAR | A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format. |