**Download Xampp**

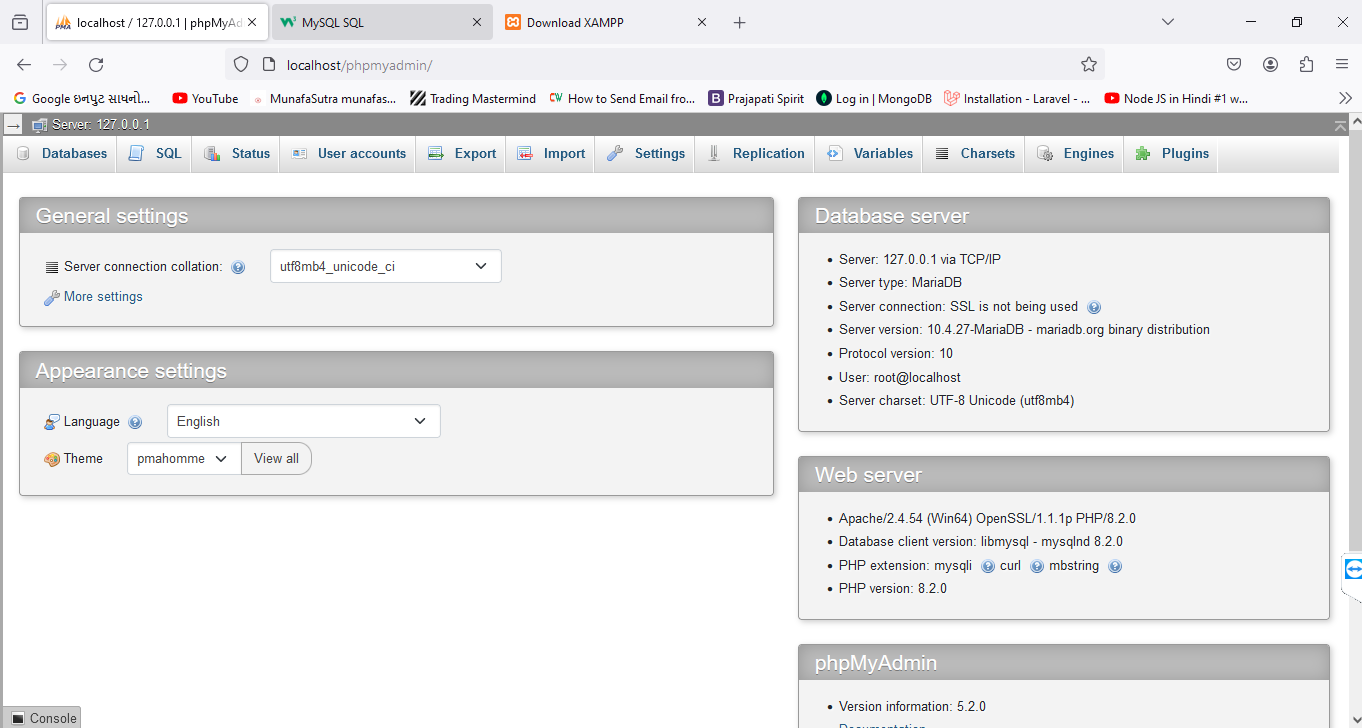
Open [h**ttps://www.apachefriends.org/download.html**](https://www.apachefriends.org/download.html) and download latest version for your operating system.

Install Downloaded software

Open Xampp Admin panel and start apache and mysql service

Close the Xampp Admin panel and open browser

In browser addressbar enter url : localhost/phpMyAdmin

****

**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.

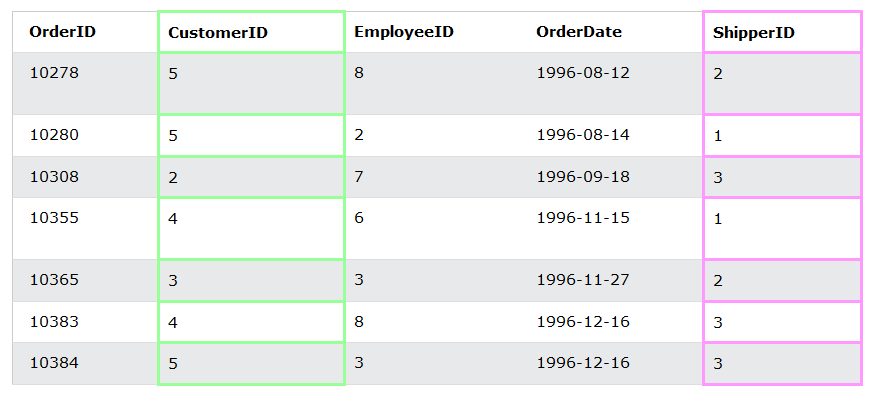
Look at the following three tables "Customers", "Orders", and "Shippers" from the Northwind database:

### Customers Table



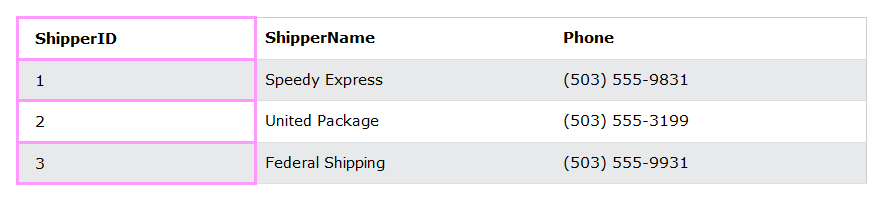
The relationship between the "Customers" table and the "Orders" table is the CustomerID column:

### Orders Table



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

### Shippers Table



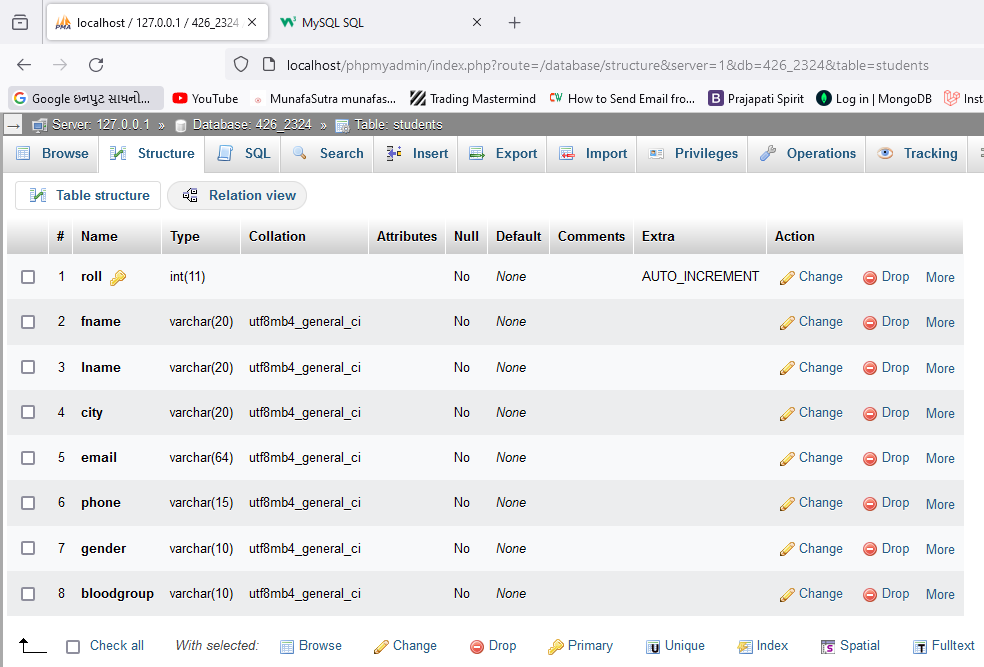
Before begin with MYsql Tutorial you need to create new Database and table with some of the data.

Open database section from localhost/phpMyAdmin -> then select create database section and enter your required database name then press create.

After successfully creation of new database, you will be redirected to newly created database page.

Create new table as following

Enter table name -> students -> create 8 columns



Create above table and add some of the data as below



# 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:

### Example

SELECT \* FROM Customers;

SELECT \* from students;

## 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.

select \* from students;

SELECT \* from students;

## 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

# 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.

SELECT roll, fname, lname from students

SELECT roll, fname, lname, city from students;

If you want to select all the fields available in the table, use the following syntax:

SELECT \* FROM *table\_name*;

SELECT \* FROM students;

## SELECT \* Example

The following SQL statement selects ALL the columns from the "Customers" table:

### Example

SELECT \* FROM Customers;

## 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 DISTINCT city from students

## SELECT Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "Country" column in the "Customers" table:

SELECT city from students;

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;

SELECT \* FROM students WHERE city = 'rajkot'

SELECT \* FROM students WHERE roll > 5

SELECT \* FROM students WHERE roll = 5;

SELECT \* FROM students WHERE roll <> 5;

SELECT \* 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.!

## 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 gender = 'male'

SELECT \* from students WHERE gender = male

SELECT \* from students WHERE roll = 8

SELECT \* from students WHERE roll = '8';

# 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 ...;

### OR Syntax

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

### NOT Syntax

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

SELECT roll, fname, lname, city from students WHERE roll = 1 and city = 'rajkot'

SELECT roll, fname, lname, city from students WHERE roll = 1 and city = 'surat';

SELECT roll, fname, lname, city from students WHERE city = 'rajkot' or city = 'surat';

SELECT roll, fname, lname, city from students WHERE not (city = 'rajkot' or city = 'surat');

SELECT roll, fname, lname, city from students WHERE not 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 = 'rajkot' or city = 'surat')

# 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;

SELECT \* from students ORDER by fname desc;

## ORDER BY Several Columns Example

SELECT \* from students ORDER by fname desc, lname 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 Example

The following SQL statement inserts a new record in the "Students" table:

INSERT into students (fname, lname, city, email, phone, gender, bloodgroup) VALUES ('patel', 'riya', 'ahamdabad', 'riya@gmail.com', '9988999900', 'female', 'o positive');

1 row inserted.   
Inserted row id: **11** (Query took 0.0032 seconds.)

**Did you notice that we did not insert any number into the Roll field?**  
The **roll** 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 into students (fname, lname, city, email, phone, gender, bloodgroup) VALUES ('patel', 'siya', 'ahamdabad', 'siya@gmail.com', '9988999900', 'female', 'o positive');

## Insert Data Only in Specified Columns

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

INSERT into students (fname, lname, city, gender, bloodgroup) VALUES ('patel', 'siya', 'ahamdabad', 'female', 'o positive');

INSERT into students VALUES ('patel', 'jiya', 'jiya@gmail.com', '9988998899','ahamdabad', 'female', 'o positive');

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

INSERT into students VALUES (null, 'patel', 'jiya', 'jiya@gmail.com', '9988998899','ahamdabad', 'female', 'o positive');

# 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!

SELECT \* from students WHERE email is null

SELECT \* from students WHERE email = '';

## 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;

INSERT into students (fname, lname, city, gender, bloodgroup) VALUES ('patel', 'siya', 'ahamdabad', 'female', 'o positive');

INSERT into students (fname, lname, city, gender, bloodgroup) VALUES ('pandya', 'priya', 'ahamdabad', 'female', 'o positive');

SELECT \* from students WHERE email is null

SELECT \* from students WHERE email is not null;

**Tip:** Always use IS NULL to look for NULL values.

## The IS NOT NULL Operator

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

# 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 = 'Gandhinagar' WHERE roll = 13

UPDATE students set city = 'Bhuj' WHERE roll > 13

## UPDATE Multiple Records

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

UPDATE students set email = 'demo@gmail.com', phone = '0000000000' where email is null

## Update Warning!

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

UPDATE students set bloodgroup = 'O Negative'

UPDATE students SET bloodgroup = 'B +ve' WHERE roll >= 12

# 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 LIMIT 5

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

What if we want to select records 4 - 6 (inclusive)?

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 students LIMIT 5 OFFSET 5;

SELECT \* FROM students LIMIT 5 OFFSET 10;

SELECT \* FROM students LIMIT 10, 5;

SELECT \* FROM students LIMIT 0, 5;

SELECT \* FROM students LIMIT 20, 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(roll) FROM students

SELECT min(roll) FROM students;

SELECT max(fees) FROM students

SELECT min(fees) 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 where fees > 10000;

SELECT COUNT(roll), sum(fees) from students where fees > 10000;

SELECT COUNT(roll), sum(fees) from students where city = 'rajkot';

SELECT COUNT(roll), sum(fees), avg(fees) from students where city = 'rajkot';

# 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

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 not 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.

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" |

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

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

# 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!

SELECT \* from students WHERE city like 'raj%'

# 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 = 'baroda' or city = 'jetpur'

## IN Operator Examples

SELECT \* from students WHERE city in ('Rajkot', 'baroda', 'jetpur');

SELECT \* from students WHERE city not in ('Rajkot', 'baroda', 'jetpur');

# 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 >= 5 and roll <= 10

SELECT \* from students WHERE roll between 1 and 5

SELECT \* from students WHERE roll between 5 and 10;

SELECT \* from students WHERE fname between 'aryan' and 'nishit';

SELECT \* from students WHERE fees between 10000 and 15000;

SELECT \* from students WHERE fees not between 10000 and 15000;

## BETWEEN with IN Example

SELECT \* from students WHERE fees not between 10000 and 15000 and roll in (1,2,3,4,5);

SELECT \* from students WHERE fees not between 10000 and 15000 and roll <= 5;

## BETWEEN Text Values Example

SELECT \* from students WHERE fname between 'aryan' and 'nishit';

## BETWEEN Dates Example

SELECT \* from students WHERE admissiondate BETWEEN '2020-01-01' and '2022-12-31'

SELECT \* from students WHERE admissiondate not BETWEEN '2020-01-01' and '2022-12-31';

# 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!

## SQL DELETE Example

DELETE from students WHERE city = 'Bhuj'

DELETE from students WHERE bloodgroup = 'B +ve'

## 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 students

Truncate table students

Delete all the data from table and reset all the auto increment from table

TRUNCATE TABLE 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;

SELECT roll, fname, lname FROM students;

SELECT roll as "Roll Number", fname as "First Name", lname as "Last Name" FROM students;

### Alias Table Syntax

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

**Note:** Single or double quotation marks are required if the alias name contains spaces:

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

SELECT concat\_ws(" - ",roll, fname, lname, city, email, gender, phone ) from students;

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

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

# 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.email, students.phone, marks.total, marks.result from **students inner join marks** on students.roll = marks.roll

SELECT students.roll, students.fname, students.lname, attendance.absents, attendance.presents from **students inner join attendance** on students.roll = attendance.roll

SELECT s.roll, s.fname, s.lname, a.absents, a.presents from students as s inner join attendance as a on s.roll = a.roll;

SELECT s.roll, s.fname, s.lname, a.absents, a.presents from students s inner join attendance a on s.roll = a.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

Top of Form

## 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.city, students.email, students.phone, students.gender, attendance.absents, attendance.presents from students INNER join attendance on students.roll = attendance.roll

SELECT s.roll, s.fname, s.lname, s.city, s.email, s.phone, s.gender, s.admissiondate, a.absents, a.presents from students s **INNER join** attendance a on s.roll = a.roll

Bottom of Form

## JOIN Three Tables

SELECT s.roll, s.fname, s.lname, s.city, s.email, s.phone, s.gender, s.admissiondate, 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;

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders). If not data match in right table empty fields filled with NULL value.

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, 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;

**Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders). If not data match in left table empty fields filled with NULL value.

SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, 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!

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

SELECT \* from students CROSS JOIN attendance;

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 attendance WHERE students.roll = attendance.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, concat\_ws("-", s1.fname, s1.lname) as "Student From Table 1", s2.roll, concat\_ws("-", s2.fname, s2.lname) as "Student From Table 2" from students s1, students s2 WHERE s1.roll <> s2.roll and s1.city = s2.city ORDER by s1.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 WHERE city = 'Rajkot'

### 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;

SELECT \* from students

UNION

SELECT \* from students1

SELECT \* from students

UNION ALL

SELECT \* from students1;

## SQL UNION With WHERE

SELECT \* from students WHERE city = 'Rajkot'

UNION ALL

SELECT \* 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 gender, COUNT(gender) as "Total Students" from students GROUP by gender

SELECT city, COUNT(city) as "Total Students From " from students GROUP by(city)

SELECT city, COUNT(city) as "Total Students From " from students GROUP by(city) ORDER by (COUNT(city));

SELECT city, COUNT(city) as "Total Students From " from students GROUP by(city) ORDER by (COUNT(city)) desc;

### GROUP BY Syntax

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

## GROUP BY With JOIN Example

SELECT students.roll, students.fname, students.lname, COUNT(fees.roll) as "Times", sum(fees.amount) as "Total Payment" from students INNER join fees on students.roll = fees.roll GROUP by (fees.roll);

SELECT students.roll, students.fname, students.lname, COUNT(fees.roll) as "Times", sum(fees.amount) as "Total Payment" from students INNER join fees on students.roll = fees.roll GROUP by (fees.roll) ORDER by sum(fees.amount) desc;

# 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 city, count(roll) from students GROUP BY(city) HAVING COUNT(roll) >= 5;

SELECT city, count(roll) from students GROUP BY(city) HAVING COUNT(roll) >= 2;

SELECT students.roll, students.fname, students.lname, COUNT(fees.roll) as "Total Times", sum(fees.amount) as "Total Amount" FROM students INNER JOIN fees on students.roll = fees.roll GROUP by (fees.roll) HAVING sum(fees.amount) >= 20000 ORDER by (sum(fees.amount)) desc;

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, COUNT(fees.roll) as "Total Times", sum(fees.amount) as "Total Amount" FROM students INNER JOIN fees on students.roll = fees.roll WHERE students.roll = 1 or students.roll = 2 GROUP by (fees.roll) HAVING sum(fees.amount) >= 20000 ORDER by (sum(fees.amount)) desc;

## 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 \* from students WHERE 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);

SELECT \* from students

SELECT \* from marks

SELECT \* from marks WHERE result <> 'pass';

SELECT roll from marks WHERE result <> 'pass';

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

SELECT \* from students WHERE **roll** = any (SELECT **roll** from attendance WHERE presents > 150);

**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;

### 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 <=).

The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10):

SELECT roll, fname, lname from students where roll = all (SELECT roll from attendance WHERE presents > 100 AND students.roll = attendance.roll);

SELECT roll,fname,lname from students where roll = all (SELECT roll from attendance WHERE presents > 150 AND students.roll = attendance.roll);

# 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

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, gender) SELECT fname, lname, city, gender from students WHERE city = 'rajkot'

# 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.

SELECT roll, fname, lname, city, gender, case when gender = 'male' THEN 'Blue Color' when gender = 'female' THEN 'Pink Color' end as "Favorite Color" from students

SELECT roll, fname, lname, city, gender, case when gender = 'male' THEN 'Blue Color' when gender = 'female' THEN 'Pink Color' else 'Brown Color' end as "Favorite Color" from students;

## CASE Syntax

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

SELECT roll, fname, lname, city, gender, case when city = 'rajkot' THEN 'Home Town' when city = 'surat' THEN 'Far From Home' when city = 'ahamdabad' THEN 'Far Home' when city = 'bhuj' THEN 'Kutchh Area' when city = 'jetpur' THEN 'Near by Home' else 'Unknown Distance' end as "Distance From Home" from students;

# MySQL NULL Functions

## MySQL IFNULL() and COALESCE() Functions

SELECT roll, absents, presents, (absents+presents) as "Total Working Days" from attendance;

## 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.

SELECT roll, absents, presents, (ifnull(absents, 0)+ifnull(presents,0)) as "Total Working Days" from attendance;

## MySQL COALESCE() Function

Or we can use the [COALESCE()](https://www.w3schools.com/mysql/func_mysql_coalesce.asp) function, like this:

SELECT roll, absents, presents, (COALESCE(absents, 0) + COALESCE(presents,0)) as "Total Working Days" from attendance;

# 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:

-- Select all:  
SELECT \* FROM Customers;

## 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:

### Example

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

SELECT \* FROM Customers -- WHERE City='Berlin';

-- SELECT \* FROM Customers;  
SELECT \* FROM Products;

/\*SELECT \* FROM Customers;  
SELECT \* FROM Products;  
SELECT \* FROM Orders;  
SELECT \* FROM Categories;\*/  
SELECT \* FROM Suppliers;

To ignore just a part of a statement, also use the /\* \*/ comment.

The following example uses a comment to ignore part of a line:

SELECT roll, **/\*absents, presents,\*/** (COALESCE(absents, 0) + COALESCE(presents,0)) as "Total Working Days" from attendance;

SELECT \* FROM Customers WHERE (CustomerName LIKE 'L%'  
OR CustomerName LIKE 'R%' /\*OR CustomerName LIKE 'S%'  
OR CustomerName LIKE 'T%'\*/ OR CustomerName LIKE 'W%')  
AND Country='USA' ORDER BY CustomerName;

# MySQL Operators

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

SELECT \* FROM students WHERE roll = some (SELECT roll FROM marks WHERE result = 'fail');

# 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

The following SQL statement creates a database called "testDB":

CREATE DATABASE testDB

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

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!

## DROP DATABASE Example

The following SQL statement drops the existing database "testDB":

show DATABASES;

drop DATABASE testDB

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).

create TABLE persons(personid int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), city varchar(20), email varchar(20), phone varchar(20), gender varchar(10))

#1046 - No database selected

Use 426\_2324

Use your database name

create TABLE persons(personid int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), city varchar(20), email varchar(20), phone varchar(20), gender varchar(10))

## 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.

### Syntax

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

CREATE table persons\_backup as SELECT \* from persons

# 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 - 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 - 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 email varchar(128)

# 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;

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

## MySQL DROP TABLE Example

drop TABLE persons

## MySQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.(reset auto increment to 1)

### Syntax

TRUNCATE TABLE table\_name;

# 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 (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);

## NOT NULL on ALTER TABLE

ALTER TABLE Persons  
MODIFY Age int NOT NULL;

CREATE TABLE persons (personid int, fname varchar(20), lname varchar(20), city varchar(20))

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot')

INSERT into persons (personid, fname, lname, city) values (2, 'Jeel', null, 'Rajkot')

INSERT into persons (personid, fname, lname, city) values (3, null, null, 'Rajkot')

select \* from persons

drop table persons

CREATE TABLE persons (personid int not null, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null)

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot')

select \* from persons

INSERT into persons (personid, fname, lname, city) values (2, 'Jeel', null, 'Rajkot')

#1048 - Column 'lname' cannot be null

alter TABLE persons add COLUMN age int

INSERT into persons (personid, fname, lname, city, age) values (1, 'Jeel', 'Ghodasara', 'Rajkot', null)

ALTER TABLE persons MODIFY COLUMN age int NOT null

INSERT into persons (personid, fname, lname, city, age) values (1, 'Jeel', 'Ghodasara', 'Rajkot', 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 allows null values in column**

## UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint 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,  
    UNIQUE (ID)  
);

drop table persons;

create table persons (personid int, fname varchar(20), lname varchar(20), city varchar(20), UNIQUE(personid))

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot')

Select \* from persons

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot')

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

INSERT into persons (personid, fname, lname, city) values (2, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (**null**, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (**null**, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (**null**, 'Jeel', 'Ghodasara', 'Rajkot');

drop TABLE persons

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

CREATE TABLE persons (personid int, fname varchar(20), lname varchar(20), city varchar(20), CONSTRAINT **unq\_pid** UNIQUE (personid))

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot');

#1062 - Duplicate entry '1' for key '**unq\_pid**'

## DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

ALTER TABLE persons drop CONSTRAINT unq\_pid

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot');

## UNIQUE Constraint on ALTER TABLE

TRUNCATE TABLE persons

ALTER TABLE persons add CONSTRAINT unq\_pid UNIQUE(personid)

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (1, 'Jeel', 'Ghodasara', 'Rajkot');

## UNIQUE Constraint on Multiple Columns TABLE

create table listofvillages (villageid int AUTO\_INCREMENT PRIMARY key, villagename varchar(20), taluka varchar(20), distrcit varchar(20), CONSTRAINT unq\_village\_taluka UNIQUE(villagename, taluka))

INSERT into listofvillages (villagename, taluka, distrcit) VALUES('Navagam', 'Rajkot', 'Rajkot')

INSERT into listofvillages (villagename, taluka, distrcit) VALUES('Navagam', 'Gondal', 'Rajkot');

INSERT into listofvillages (villagename, taluka, distrcit) VALUES('Navagam', 'Gondal', 'Rajkot');

**#1062 - Duplicate entry 'Navagam-Gondal' for key 'unq\_village\_taluka'**

INSERT into listofvillages (villagename, taluka, distrcit) VALUES('Madhapar', 'Rajkot', 'Rajkot');

INSERT into listofvillages (villagename, taluka, distrcit) VALUES('Madhapar', 'Bhuj', 'Kutchh');

SELECT \* from listofvillages

# 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

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

drop table persons

create TABLE persons (personid int PRIMARY KEY, fname varchar(20), lname varchar(20), city varchar(20))

INSERT into persons (personid, fname, lname, city) values (**1**, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (**1**, 'Jeel', 'Ghodasara', 'Rajkot');

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

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

drop table persons

create TABLE persons (personid int, fname varchar(20), lname varchar(20), city varchar(20), CONSTRAINT pri\_key\_pid PRIMARY KEY (personid))

INSERT into persons (personid, fname, lname, city) values (**1**, 'Jeel', 'Ghodasara', 'Rajkot');

INSERT into persons (personid, fname, lname, city) values (**1**, 'Jeel', 'Ghodasara', 'Rajkot');

## 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:

drop table persons

create TABLE persons (personid int, fname varchar(20), lname varchar(20), city varchar(20))

ALTER TABLE persons add PRIMARY key (personid)

## DROP a PRIMARY KEY Constraint

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

ALTER TABLE Persons  
DROP PRIMARY KEY;

ALTER TABLE persons drop PRIMARY key;

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 pri\_key\_pid\_fname PRIMARY key (personid, fname)

CREATE TABLE students (roll int AUTO\_INCREMENT PRIMARY KEY, fname varchar(20), lname varchar(20), city varchar(20), phone varchar(20), email varchar(20), gender varchar(10))

INSERT into students (fname, lname, city, phone, email, gender) values ('priyajitsinh', 'jadeja', 'rajkot', '9988990000', 'jadeja@gmail.com', 'male'),('smit', 'vanzara', 'rajkot', '9988991100', 'smit@gmail.com', 'male'),('aryan', 'thakrar', 'rajkot', '9988911000', 'aryan@gmail.com', 'male'),('hemal', 'varu', 'rajkot', '9228990000', 'hemal@gmail.com', 'female'),('dhruvisha', 'bhatt', 'rajkot', '9228990000', 'bhatt@gmail.com', 'female'),('devangi', 'dave', 'rajkot', '3388990000', 'devangi@gmail.com', 'female'),('hensi', 'lambiya', 'rajkot', '7788990000', 'hensi@gmail.com', 'female')

select \* from students

# 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 |

### Orders Table

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID (foreign key)** |
| 1 | 77895 | **3** |
| 2 | 44678 | **3** |
| 3 | 22456 | **2** |
| 4 | 24562 | **1** |

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 attendance (attendance\_id int AUTO\_INCREMENT PRIMARY key, roll int, absents int, present int, FOREIGN key(roll) REFERENCES students(roll))

INSERT INTO `attendance` (`attendance\_id`, `roll`, `absents`, `present`) VALUES (NULL, '5', '112', '112');

INSERT INTO `attendance` (`attendance\_id`, `roll`, `absents`, `present`) VALUES (NULL, '15', '112', '112');

#1452 - Cannot add or update a child row: a foreign key constraint fails (`426\_2324`.`attendance`, CONSTRAINT `**attendance\_ibfk\_1**` FOREIGN KEY (`roll`) REFERENCES `students` (`roll`))

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 exams (examid int AUTO\_INCREMENT PRIMARY KEY, roll int, examdate date, subject varchar(20), marks int, result varchar(20), CONSTRAINT **fk\_student\_exam** FOREIGN KEY (roll) REFERENCES students(roll))

INSERT INTO `exams` (`examid`, `roll`, `examdate`, `subject`, `marks`, `result`) VALUES (NULL, '5', '2024-04-01', 'MySql', '123', 'fail');

INSERT INTO `exams` (`examid`, `roll`, `examdate`, `subject`, `marks`, `result`) VALUES (NULL, '15', '2024-04-01', 'MySql', '123', 'fail');

#1452 - Cannot add or update a child row: a foreign key constraint fails (`426\_2324`.`exams`, CONSTRAINT `**fk\_student\_exam**` FOREIGN KEY (`roll`) REFERENCES `students` (`roll`))

## FOREIGN KEY on ALTER TABLE

create TABLE fees (feesid int AUTO\_INCREMENT PRIMARY KEY, roll int, paymentdate date, amount decimal(10,2), paymentmode varchar(20), referenceid varchar(20))

ALTER TABLE fees add CONSTRAINT fk\_student\_fees\_roll FOREIGN KEY(roll) REFERENCES students(roll)

## DROP a FOREIGN KEY Constraint

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

ALTER TABLE fees drop CONSTRAINT fk\_student\_fees\_roll

FROM students WHERE `students`.`roll` = 5"

You cannot delete data from primary table until single row present with same reference in child table

# 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 Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CHECK (Age>=18)  
);

INSERT into persons (id, lastname, firstname, age) values (1,'smit', 'vanzara', 17)

#4025 - CONSTRAINT `CONSTRAINT\_1` failed for `426\_2324`.`persons`

INSERT into persons (id, lastname, firstname, age) values (1,'smit', 'vanzara', 27)

## CHECK on ALTER TABLE

ALTER TABLE students add CONSTRAINT check\_gender CHECK (**gender = 'male' or gender = 'female'**)

INSERT INTO students (fname, lname, city, email, phone, gender) VALUES ('Demo', 'text', 'morbi', 'demo@gmail.com', '9900887766', **'fenal'**)

#4025 - CONSTRAINT `check\_gender` failed for `426\_2324`.`students`

## DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

ALTER TABLE students drop CONSTRAINT check\_gender

# 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

CREATE TABLE students\_address (addressid int AUTO\_INCREMENT PRIMARY KEY, roll int, line1 varchar(40), line2 varchar(40), landmark varchar(40), city varchar(20), **state varchar(20) DEFAULT 'Gujarat'**, **country varchar(20) DEFAULT 'india'**)

## DEFAULT on ALTER TABLE

ALTER TABLE students add COLUMN admission\_date timestamp DEFAULT CURRENT\_TIMESTAMP

INSERT INTO `students` (`roll`, `fname`, `lname`, `city`, `phone`, `email`, `gender`, `admission\_date`) VALUES (NULL, 'Another ', 'Example', 'of', '9988990088', 'demo@msil.com', 'male', current\_timestamp());

## DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

ALTER TABLE students\_address ALTER STATE drop DEFAULT

# 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 INDEX students\_fname on students(fname)

### 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 unique INDEX students\_fname1 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;

drop INDEX students\_fname1 on students;

ALTER TABLE students DROP index students\_fname;

# MySQL AUTO INCREMENT Field

## What is an AUTO INCREMENT Field?

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

## MySQL AUTO\_INCREMENT Keyword

MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO\_INCREMENT sequence start with another value, use the following SQL statement:

ALTER TABLE Persons AUTO\_INCREMENT=100;

ALTER TABLE students AUTO\_INCREMENT = 200

# MySQL Working with Dates

## MySQL Dates

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

## MySQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

DATE - format YYYY-MM-DD

DATETIME - format: YYYY-MM-DD HH:MI:SS

TIMESTAMP - format: YYYY-MM-DD HH:MI:SS

YEAR - format YYYY or YY

**Note:** The date data type are set for a column when you create a new table in your database!

SELECT \* from students WHERE dateofbirth >= '2002-12-15'

SELECT \* from students WHERE dateofbirth >= '2006-01-01';

SELECT \* from students WHERE dateofbirth <= '2006-01-01';

**Note:** Two dates can easily be compared if there is no time component involved!

SELECT \* from students WHERE admission\_date = '2024-04-05'

SELECT \* from students WHERE admission\_date like '2024-04-05%'

SELECT \* from students WHERE date(admission\_date) = '2024-04-05';

SELECT \* from students WHERE not date(admission\_date) = '2024-04-05';

**Tip:** To keep your queries simple and easy to maintain, do not use time-components in your dates, unless you have to!

# MySQL Views

## MySQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

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

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

A view is created with the CREATE VIEW statement.

### CREATE VIEW Syntax

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

create view getStudents as SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.dateofbirth, students.gender, attendance.absents, attendance.present from students inner join attendance on students.roll = attendance.roll;

SELECT \* from getstudents;

SELECT \* FROM `getstudents` WHERE roll = 2;

**Note:** A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

## MySQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

### CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

create or replace view getStudents as SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.dateofbirth, students.gender, attendance.absents, attendance.present from students inner join attendance on students.roll = attendance.roll;

create or replace view getStudents as SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.email, students.dateofbirth, students.gender, attendance.absents, attendance.present, fees.paymentdate, fees.amount, fees.paymentmode from students inner join attendance on students.roll = attendance.roll INNER join fees on students.roll = fees.roll;

SELECT \* from getstudents

## MySQL Dropping a View

A view is deleted with the DROP VIEW statement.

### DROP VIEW Syntax

DROP VIEW view\_name;

drop view getstudents;

# 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**.

### 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.

### 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.

### 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.

### 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.

### Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

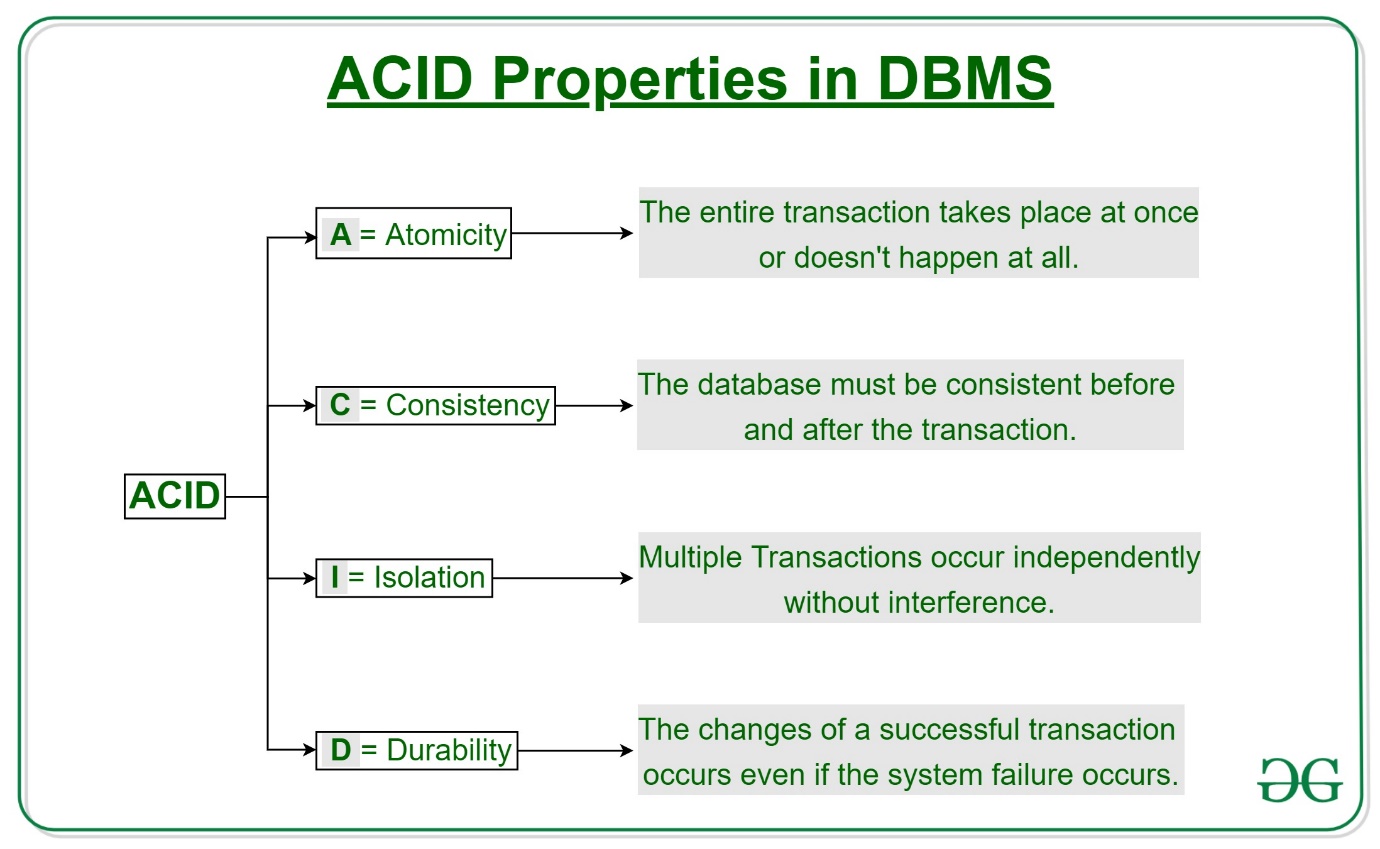
* SELECT

**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

# ACID Properties in DBMS

A **transaction** is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.



### **Atomicity:**

By this, we mean that either the entire transaction takes place at once or doesn’t happen at all. There is no midway ie. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

—**Abort**: If a transaction aborts, changes made to the database are not visible.   
—**Commit**: If a transaction commits, changes made are visible.

Atomicity is also known as the ‘All or nothing rule’.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.



If the transaction fails after completion of **T1** but before completion of **T2**.( say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an **inconsistent database state**. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

### Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,   
The total amount before and after the transaction must be maintained.

Total **before T** occurs = **500 + 200 = 700**. Total **after T occurs** = **400 + 300 = 700**.

Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

### Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.   
Let **X**= 500, **Y** = 500.

Consider two transactions **T** and **T”.**



Suppose **T** has been executed till **Read (Y)** and then **T’’** starts. As a result, interleaving of operations takes place due to which **T’’** reads the correct value of **X** but the incorrect value of **Y** and sum computed

by   
**T’’: (X+Y = 50, 000+500=50, 500)**   
is thus not consistent with the sum at end of the transaction:   
**T: (X+Y = 50, 000 + 450 = 50, 450)**

by   
**T’’: (X+Y = 50, 000+500=50, 500)**   
is thus not consistent with the sum at end of the transaction:   
**T: (X+Y = 50, 000 + 450 = 50, 450)**

### Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

**Some important points:**

| **Property** | **Responsibility for maintaining properties** |
| --- | --- |
| Atomicity | Transaction Manager |
| Consistency | Application programmer |
| Isolation | Concurrency Control Manager |
| Durability | Recovery Manager |

The **ACID** properties, in totality, provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

ACID properties are the four key characteristics that define the reliability and consistency of a transaction in a Database Management System (DBMS). The acronym ACID stands for Atomicity, Consistency, Isolation, and Durability. Here is a brief description of each of these properties:

1. **Atomicity:** Atomicity ensures that a transaction is treated as a single, indivisible unit of work. Either all the operations within the transaction are completed successfully, or none of them are. If any part of the transaction fails, the entire transaction is rolled back to its original state, ensuring data consistency and integrity.
2. **Consistency**: Consistency ensures that a transaction takes the database from one consistent state to another consistent state. The database is in a consistent state both before and after the transaction is executed. Constraints, such as unique keys and foreign keys, must be maintained to ensure data consistency.
3. **Isolation:** Isolation ensures that multiple transactions can execute concurrently without interfering with each other. Each transaction must be isolated from other transactions until it is completed. This isolation prevents dirty reads, non-repeatable reads, and phantom reads.
4. **Durability:** Durability ensures that once a transaction is committed, its changes are permanent and will survive any subsequent system failures. The transaction’s changes are saved to the database permanently, and even if the system crashes, the changes remain intact and can be recovered.

Overall, ACID properties provide a framework for ensuring data consistency, integrity, and reliability in DBMS. They ensure that transactions are executed in a reliable and consistent manner, even in the presence of system failures, network issues, or other problems. These properties make DBMS a reliable and efficient tool for managing data in modern organizations.

### Advantages of ACID Properties in DBMS:

1. **Data Consistency**: ACID properties ensure that the data remains consistent and accurate after any transaction execution.
2. **Data Integrity**: ACID properties maintain the integrity of the data by ensuring that any changes to the database are permanent and cannot be lost.
3. **Concurrency Control**: ACID properties help to manage multiple transactions occurring concurrently by preventing interference between them.
4. **Recovery**: ACID properties ensure that in case of any failure or crash, the system can recover the data up to the point of failure or crash.

### Disadvantages of ACID Properties in DBMS:

1. **Performance:** The ACID properties can cause a performance overhead in the system, as they require additional processing to ensure data consistency and integrity.
2. **Scalability:** The ACID properties may cause scalability issues in large distributed systems where multiple transactions occur concurrently.
3. **Complexity**: Implementing the ACID properties can increase the complexity of the system and require significant **expertise** and resources.

Overall, the advantages of ACID properties in DBMS outweigh the disadvantages. They provide a reliable and consistent approach to data management, ensuring data integrity, accuracy, and reliability. However, in some cases, the overhead of implementing ACID properties can cause performance and scalability issues. Therefore, it’s important to balance the benefits of ACID properties against the specific needs and requirements of the system.

# 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 |