Software requirements

Download Xampp from <https://www.apachefriends.org/download.html>

Download latest version of php 8.2.12 + (Feb 2024)

How to open mysql server

Go to xampp control panel and start apache server and mysql server

Then go to browser and enter URL <http://localhost/phpmyadmin/>

From home page of mysql server select databases section and create new database.

**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.

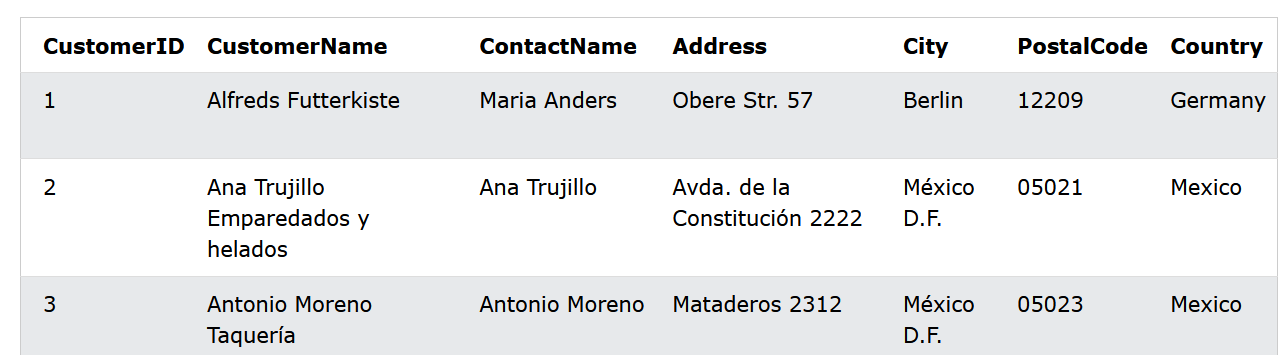
## What is a Database Table?

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:

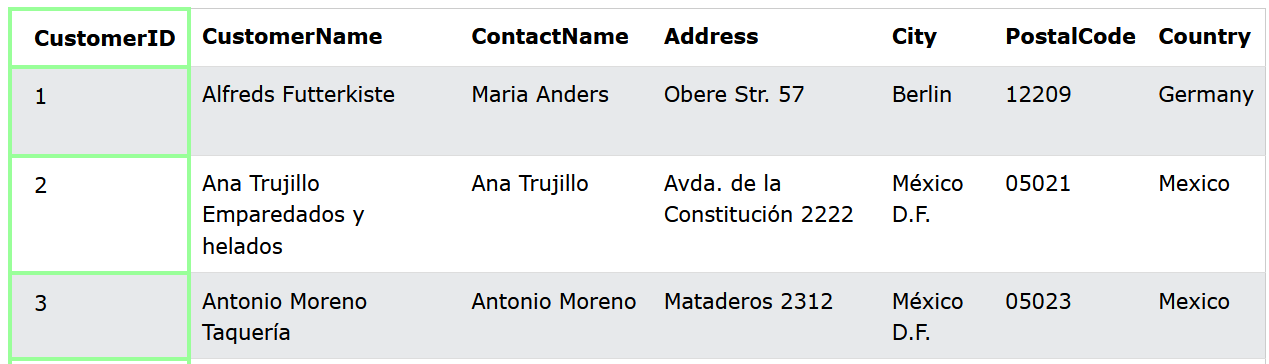


The columns in the "Customers" table above are: CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country.

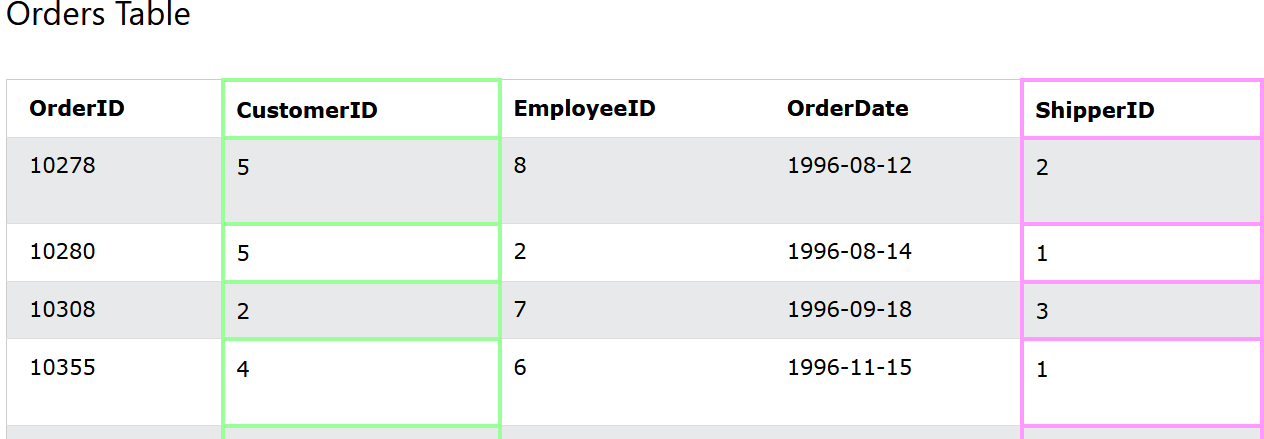
## What is a Relational Database?

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

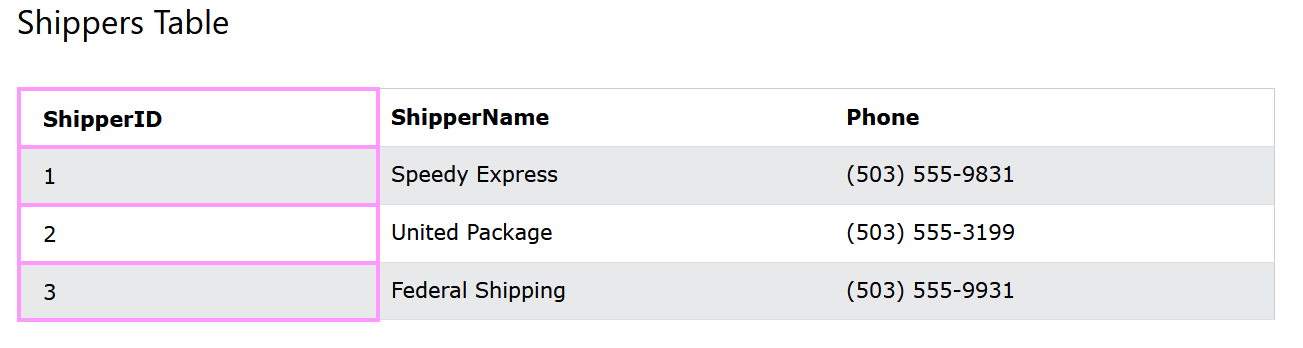
Look at the following three tables "Customers", "Orders", and "Shippers" from the Northwind database:



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



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



## 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

## 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.

## 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

## 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;

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

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 \* Example

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

Select \* from students

## The MySQL SELECT DISTINCT Statement

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

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

### SELECT DISTINCT Syntax

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

SELECT DISTINCT city FROM students

SELECT DISTINCT fname 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

## 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.!

## 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 city = 'rajkot'

Select \* from students where roll > 5

Select \* from students where not roll > 5;

Select \* from students where roll != 5;

## 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 \* FROM `students` WHERE roll = 1

SELECT \* FROM `students` WHERE roll = 1 and fname = 'udit';

SELECT \* FROM `students` WHERE roll = 1 or roll = 2;

SELECT \* FROM `students` WHERE roll = 1 or city = 'rajkot';

SELECT \* FROM `students` WHERE roll = 1 and city = 'rajkot' or city = 'surat';

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

## 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 ORDER by fname

SELECT \* from students ORDER by fname desc;

SELECT \* FROM students ORDER by fname, city

SELECT \* FROM students ORDER by fname, city desc

## 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, gender, dateofbirth, phone) values ("Rachit", "Chauhan", "Baroda", "rachit@gmail.com", "male", "2002-02-02", "9900009900")

INSERT into students values ("Rachit", "Chauhan", "Baroda", "rachit@gmail.com", "male", "2002-02-02", "9900009900")

Error : Column count doesn't match value count at row 1

## Insert Data Only in Specified Columns

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

INSERT into students (fname, lname, city, email, gender) values ("Rachit", "Chauhan", "Baroda", "rachit@gmail.com", "male");

## 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!

## 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.

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

SELECT \* FROM `students` WHERE dateofbirth = 'NULL'; // no data

SELECT \* FROM `students` WHERE dateofbirth is NULL;

SELECT \* FROM `students` WHERE dateofbirth is not NULL;

### 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;

## The IS NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

**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).

## 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' //always use where while update data

update students set fname = 'Yagnik', lname = 'Yadav', city = 'Bhavanagar' WHERE roll = 3

update students set fname = 'Yagnik', lname = 'Yadav', city = 'Bhavanagar' WHERE roll > 3 and roll < 6;

UPDATE students set city = 'Rajkot' WHERE roll >= 5

## UPDATE Multiple Records

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

## Update Warning!

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

UPDATE students set fname = 'tushar', lname = 'kadam' WHERE roll = 4

## 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

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;

## ADD a WHERE CLAUSE

SELECT \* FROM `students` where city = ‘rajkot’ Limit 5

## 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 min(roll) FROM students;

SELECT max(roll) FROM students

SELECT max(fees) FROM students

SELECT min(fees) FROM students;

## 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;

SELECT COUNT(fees) FROM students

SELECT COUNT(fees) FROM students WHERE fees > 10000;

SELECT sum(fees) FROM students WHERE fees > 10000;

SELECT count(fees), sum(fees) FROM students WHERE fees > 10000;

SELECT count(fees), sum(fees), avg(fees) FROM students

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;

## 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 roll = 5

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 students

INSERT into students (fname, lname, city, email, gender, dateofbirth, phone, fees) values ("Rachit", "Chauhan", "Baroda", "rachit@gmail.com", "male", "2002-02-02", "9900009900", 15000);

* Add multiple rows and check for roll number

It is possible to delete all the data from table and reset complete structure of the table with truncate clause

TRUNCATE TABLE students

INSERT into students (fname, lname, city, email, gender, dateofbirth, phone, fees) values ("Rachit", "Chauhan", "Baroda", "rachit@gmail.com", "male", "2002-02-02", "9900009900", 15000);

* Add multiple rows and check for roll number // started from 1

Insert some data in table

INSERT INTO `students` (`roll`, `fname`, `lname`, `city`, `email`, `gender`, `dateofbirth`, `phone`, `fees`, `admissiondate`) VALUES (NULL, 'udit', 'ghetiya', 'Rajula', 'udit@gmail.com', 'male', '2004-02-11', '998899889900', '18000', current\_timestamp()),(NULL, 'Gaurang', 'Pandya', 'Baroda', 'gaurang@gmail.com', 'male', '2004-02-11', '998899889900', '15000', current\_timestamp()),(NULL, 'yograjsinh', 'Rana', 'Junagadh', 'yorajsinh@gmail.com', 'male', '2004-02-11', '998899889900', '22000', current\_timestamp()),(NULL, 'yadav', 'yagnik', 'Bhavanagar', 'yagnik@gmail.com', 'male', '2004-02-11', '998899889900', '19000', current\_timestamp()),(NULL, 'Rachit', 'Chauhan', 'Jamnagar', 'rachit@gmail.com', 'male', '2004-02-11', '998899889900', '12000', current\_timestamp()),(NULL, 'adarsh', 'chavda', 'amreli', 'adarsh@gmail.com', 'male', '2004-02-11', '998899889900', '12000', current\_timestamp()),(NULL, 'rohan', 'dasadiya', 'Bhuj', 'rohan@gmail.com', 'male', '2004-02-11', '998899889900', '13000', current\_timestamp()),(NULL, 'Tushar', 'Kadam', 'Morbi', 'Tushar@gmail.com', 'male', '2004-02-11', '998899889900', '19000', current\_timestamp()),(NULL, 'Yash', 'Vaghela', 'Vadodara', 'yash@gmail.com', 'male', '2004-02-11', '998899889900', '22000', current\_timestamp()),(NULL, 'Jayrajsinh', 'Parmar', 'Limdi', 'jayrajsinh@gmail.com', 'male', '2004-02-11', '998899889900', '24000', current\_timestamp());

## 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 '%t';

SELECT \* FROM students WHERE fname like '%i%';

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

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

SELECT \* FROM students WHERE fname like 'r\_\_%';

SELECT \* FROM students WHERE fname like 'r%t';

SELECT \* FROM students WHERE fname not like 'r%t';

## 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 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, ...);

SELECT \* FROM students WHERE city = 'surat' or city = 'rajkot' OR city = 'baroda' or city = 'morbi'; SELECT \* FROM students WHERE city in ('rajkot', 'baroda', 'amreli', 'bhuj', 'junagadh')

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

SELECT \* FROM students WHERE roll in (SELECT roll from students WHERE not city = 'rajkot');

Outer query (sub query),

**First Execute sub query and get result from database, place this result data between in () and execute outer query,**

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

SELECT \* from students WHERE roll not in (SELECT roll FROM marks WHERE result = 'pass');

## 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 fees BETWEEN 5000 and 15000

SELECT \* FROM students WHERE fees not BETWEEN 5000 and 15000;

SELECT \* FROM students WHERE fname BETWEEN 'adarsh' and 'tushar'

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

## 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 fname as FirstName FROM students

SELECT fname as "First Name" FROM students;

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

SELECT concat\_ws(" - ", roll, fname, lname, city) as "Student Data" FROM students

Without join and alias

SELECT students.roll, students.fname, students.lname, students.city, students.email, students.gender, students.dateofbirth, students.phone, students.fees, marks.total, marks.result FROM students, marks WHERE students.fname = 'bhavdeep' and students.roll = marks.roll;

With alias

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

SELECT CURRENT\_TIMESTAMP

SELECT CURRENT\_TIMESTAMP as "Today is :";

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.gender, students.dateofbirth, students.phone, students.course, students.course, students.admissiondate, marks.total, marks.result from students inner JOIN marks on students.roll = marks.roll

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

## MySQL INNER JOIN Keyword

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



### INNER JOIN Syntax

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

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

SELECT s.\*, 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;

SELECT s.\*, m.\*, 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;

/// never do this

SELECT s.\*, m.\*, a.\* 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

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).

select s.roll, s.fname, s.lname, s.city, s.email, s.gender, s.dateofbirth, s.phone, s.fees, s.course, s.admissiondate, m.total, m.result from students s left join marks m on s.roll = m.roll;

grab all the data from left table (students ) if there is no data in right table (marks) then sql show NULL for empty values

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

## 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.gender, s.dateofbirth, s.phone, s.fees, s.course, s.admissiondate, 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).

## 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

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;

SELECT t1.fname, t1.lname from students t1, students t2 WHERE t1.roll != t2.roll and t1.city = t2.city;

SELECT t1.roll, t1.fname, t1.lname, t1.city from students t1, students t2 WHERE t1.roll != t2.roll and t1.city = t2.city;

## 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;

SELECT \* from students

UNION

SELECT \* FROM students1

### UNION ALL Syntax

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

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

SELECT \* from students

UNION ALL

SELECT \* FROM students1;

SELECT \* from students

UNION ALL

SELECT \* FROM students1 ORDER by city;

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 city, count(city) FROM students GROUP by (city)

## GROUP BY With JOIN Example

SELECT students.roll, COUNT(marks.marksid), sum(marks.total) from students INNER join marks on students.roll = marks.roll GROUP by (roll);

## The MySQL HAVING Clause

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

### HAVING Syntax

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

SELECT city, count(city) from students GROUP by (city) having count(city) = 1;

SELECT city, count(city) from students GROUP by (city) having count(city) > 1;

SELECT students.roll, COUNT(marks.marksid), sum(marks.total) from students INNER join marks on students.roll = marks.roll GROUP by (roll) HAVING sum(marks.total) >= 400;

## 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');

SELECT \* from students WHERE EXISTS (SELECT roll FROM marks WHERE result = 'ATKT');

SELECT students.roll, students.fname, students.lname from students where EXISTS (SELECT roll FROM marks WHERE students.roll = marks.roll and marks.result = 'pass')

SELECT students.roll, students.fname, students.lname from students where not EXISTS (SELECT roll FROM marks WHERE students.roll = marks.roll and marks.result = 'pass');

# MySQL ANY and ALL Operators

## The MySQL ANY and ALL Operators

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

## The ANY Operator

The ANY operator:

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

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

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

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

## 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.

SELECT all fname FROM students WHERE true

# 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 (fname, lname, city, email, gender, dateofbirth, phone, fees, course) SELECT fname, lname, city, email, gender, dateofbirth, phone, fees, course from students

INSERT into students1 (fname, lname, city, email, gender, dateofbirth, phone, fees, course) SELECT fname, lname, city, email, gender, dateofbirth, phone, fees, course from students where city = 'rajkot';

## MySQL INSERT INTO SELECT Examples

The following SQL statement copies "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):

# 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, gender, city, email FROM students

SELECT roll, fname, lname, gender, city,

case

WHEN city = 'Rajkot' THEN "Home Town"

WHEN city = 'Surat' THEN "Too much Far from Home Town"

WHEN city = 'morbi' THEN "Near by Home Town"

WHEN city = 'limdi' THEN "Far from Home Town"

WHEN city = 'Bhavanagar' THEN "Far from Home Town"

WHEN city = 'Amreli' THEN "Near by Home Town"

WHEN city = 'Jamnagar' THEN "Near by Home Town"

WHEN city = 'Bhuj' THEN "Far from Home Town"

WHEN city = 'Rajula' THEN "Far from Home Town"

WHEN city = 'vadodara' or city = 'baroda' THEN "Far from Home Town"

else "Unkonown istance"

end as "Distance From Rajkot", email FROM students;

# MySQL NULL Functions

## MySQL IFNULL() and COALESCE() Functions

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

## MySQL COALESCE() Function

SELECT roll, sum(coalesce(absents, 0) + coalesce(presents, 0)) as "Total Working Days" from attendance GROUP by (roll);

## 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).

-- find null values and replace with 0

SELECT roll, sum(coalesce(absents, 0) + coalesce(presents, 0)) as "Total Working Days" from attendance GROUP by (roll);

## 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;

/\* find null values and replace with 0 (this is multiline comment in MySql)\*/

SELECT roll, sum(coalesce(absents, 0) + coalesce(presents, 0)) as "Total Working Days" from attendance GROUP by (roll);

# 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

### 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.

### 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

### 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

### 5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

* SELECT

**a. 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;

**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;

create DATABASE demo729;

show DATABASES

# 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 demo729

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;

## 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.).

CREATE TABLE persons (personid int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), email varchar(128), phone varchar(15), gender varchar(6), `dateof birth` date)

## 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

CREATE TABLE persons\_backup1 as SELECT personid, fname, lname 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 city varchar(30)

ALTER TABLE persons add COLUMN state varchar(30), add COLUMN country varchar(20)

ALTER TABLE persons add COLUMN zipcode int after city

## 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 zipcode

## 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 gender varchar(10)

# 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!

drop TABLE persons\_backup1

## MySQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete (reset the table structure to new – auto Increment) the data inside a table, but **not** the table itself.

### Syntax

TRUNCATE TABLE table\_name;

TRUNCATE TABLE students1

# 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 **persons1** (personid int AUTO\_INCREMENT PRIMARY key, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null)

INSERT INTO persons1 (fname, lname, city) values ('Bhavdeep', 'Sorathiya', 'Rajkot')

INSERT INTO persons1 (fname, lname, city) values ('Bhavdeep', 'Sorathiya', NULL);

1048 - Column 'city' cannot be null

## NOT NULL on ALTER TABLE

ALTER table persons1 add COLUMN age int

ALTER TABLE persons1 MODIFY COLUMN age int not null

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

INSERT INTO persons1 (fname, lname, city, age) values ('Bhavdeep', 'Sorathiya', 'baroda', 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

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

create table persons (personid int not null, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null, UNIQUE(personid))

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

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

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

INSERT into persons (personid, fname, lname, city) values (**2**, 'Udit', 'patel', 'Rajkot');

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

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, CONSTRAINT **unqid** UNIQUE (personid))

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

#1062 - Duplicate entry '1' for key '**unqid**'

Following query drop named constrained unqid from persons table.

alter TABLE persons drop CONSTRAINT unqid

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

## UNIQUE Constraint on ALTER TABLE

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

Truncate table persons

alter TABLE persons add UNIQUE(personid)

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

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

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)

ALTER TABLE persons add CONSTRAINT unqid UNIQUE (personid)

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

INSERT into persons (personid, fname, lname, city) values (**1**, 'Udit', 'Ghetiya', 'Rajkot')

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

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

* Major drawback of unique constraint is it allows NULL values in column.

## DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

ALTER table persons drop CONSTRAINT unqid

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

create table listoftown (townid int AUTO\_INCREMENT PRIMARY key, townname varchar(20) UNIQUE, cityname varchar(20), district varchar(20), state varchar(20))

INSERT INTO listoftown (townname, cityname, district, state) values ('Navagam', 'Rajkot', 'Rajkot', 'Gujarat')

INSERT INTO listoftown (townname, cityname, district, state) values ('Navagam', 'Bhuj', 'Kutchh', 'Gujarat');

#1062 - Duplicate entry 'Navagam' for key 'townname'

drop table listoftown

create table listoftown (townid int AUTO\_INCREMENT PRIMARY key, townname varchar(20), cityname varchar(20), district varchar(20), state varchar(20), CONSTRAINT unqtown UNIQUE(townname, cityname, district))

INSERT INTO listoftown (townname, cityname, district, state) values ('Navagam', 'Rajkot', 'Rajkot', 'Gujarat')

INSERT INTO listoftown (townname, cityname, district, state) values ('Navagam', 'Bhuj', 'Kutchh', 'Gujarat');

# 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

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

**Note:** In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

CREATE TABLE person (id int, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null, PRIMARY key(id))

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

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

drop TABLE person

CREATE TABLE person (id int, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null, CONSTRAINT prikey\_id PRIMARY key (id))

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

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

ALTER TABLE person drop PRIMARY KEY

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

Drop table person

## 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).

CREATE TABLE person (id int, fname varchar(20) not null, lname varchar(20) not null, city varchar(20) not null)

ALTER TABLE person add PRIMARY key (id)

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

INSERT into person (id, fname, lname, city) VALUES(1, 'bhavdeep', 'sorathiya', 'rajkot')

ALTER TABLE person drop PRIMARY key

ALTER TABLE person add PRIMARY key (id, fname)

INSERT into person (id, fname, lname, city) VALUES(**1, 'bhavdeep'**, 'sorathiya', 'rajkot');

INSERT into person (id, fname, lname, city) VALUES(**1, 'bhavdip'**, 'sorathiya', 'rajkot')

INSERT into person (id, fname, lname, city) VALUES(**2, 'bhavdip',** 'sorathiya', 'rajkot');

INSERT into person (id, fname, lname, city) VALUES(**2, 'bhavdip'**, 'sorathiya', 'rajkot');

#1062 - Duplicate entry '2-bhavdip' for key 'PRIMARY'

## DROP a PRIMARY KEY Constraint

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

ALTER TABLE Person DROP PRIMARY KEY;

# 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** | **LastName** | **FirstName** | **Age** |
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |
| 3 | Pettersen | Kari | 20 |

### Orders Table

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID** |
| 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 Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

create TABLE fees (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdate date, amount int, paymentmode varchar(10), paymenttimestamp timestamp DEFAULT CURRENT\_TIMESTAMP)

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`, `paymentmode`, `paymenttimestamp`) VALUES (NULL, '123', '2024-02-17', '5000', 'cash', current\_timestamp());

Above data is invalid because we don’t have student with roll 123

drop TABLE fees

create TABLE fees (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdate date, amount int, paymentmode varchar(10), paymenttimestamp timestamp DEFAULT CURRENT\_TIMESTAMP, FOREIGN key (roll) REFERENCES students(roll))

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`, `paymentmode`, `paymenttimestamp`) VALUES (NULL, '123', '2024-02-17', '5000', 'cash', current\_timestamp());

#1452 - Cannot add or update a child row: a foreign key constraint fails (`729\_2324`.`fees`, CONSTRAINT `fees\_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 Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

drop TABLE fees

create TABLE fees (feesid int AUTO\_INCREMENT PRIMARY key, roll int, paymentdate date, amount int, paymentmode varchar(10), paymenttimestamp timestamp DEFAULT CURRENT\_TIMESTAMP, CONSTRAINT fk\_roll FOREIGN key (roll) REFERENCES students(roll))

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`, `paymentmode`, `paymenttimestamp`) VALUES (NULL, '123', '2024-02-17', '5000', 'cash', current\_timestamp());

#1452 - Cannot add or update a child row: a foreign key constraint fails (`729\_2324`.`fees`, CONSTRAINT `fk\_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\_roll

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`, `paymentmode`, `paymenttimestamp`) VALUES (NULL, '123', '2024-02-17', '5000', 'cash', current\_timestamp());

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`, `paymentmode`, `paymenttimestamp`) VALUES (NULL, '1234', '2024-02-17', '5000', 'cash', current\_timestamp());

## FOREIGN KEY on ALTER TABLE

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

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

#1452 - Cannot add or update a child row: a foreign key constraint fails (`729\_2324`.`#sql-2668\_353`, CONSTRAINT `fk\_roll` FOREIGN KEY (`roll`) REFERENCES `students` (`roll`))

TRUNCATE TABLE fees

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

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`, `paymentmode`, `paymenttimestamp`) VALUES (NULL, '1', '2024-02-17', '4000', 'cash', current\_timestamp()), (NULL, '2', '2024-02-17', '3500', 'case', current\_timestamp());

DELETE from students WHERE roll = 1

#1451 - Cannot delete or update a parent row: a foreign key constraint fails (`729\_2324`.`fees`, CONSTRAINT `fk\_roll` FOREIGN KEY (`roll`) REFERENCES `students` (`roll`))

## 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

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

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

drop TABLE person

CREATE TABLE Persons ( ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255) not null, Age int, CHECK (Age>=18) );

INSERT into persons (id, lastname, firstname, age) VALUES (1, 'bhavdeep', 'sorathiya', 21)

INSERT into persons (id, lastname, firstname, age) VALUES (1, 'bhavdeep', 'sorathiya', 17);

#4025 - CONSTRAINT `CONSTRAINT\_1` failed for `729\_2324`.`persons`

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

drop TABLE persons

CREATE TABLE persons (id int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), city varchar(20), age int, CONSTRAINT **checkAge** CHECK (age >= 18 and age <=50))

INSERT into persons (fname, lname, city, age) VALUES ('bhavdeep', 'sorathiya', 'Rajkot', 19)

INSERT into persons (fname, lname, city, age) VALUES ('bhavdeep', 'sorathiya', 'Rajkot', 51);

#4025 - CONSTRAINT `**checkAge`** failed for `729\_2324`.`persons`

ALTER TABLE persons drop CONSTRAINT checkage

INSERT into persons (fname, lname, city, age) VALUES ('bhavdeep', 'sorathiya', 'Rajkot', 51);

INSERT into persons (fname, lname, city, age) VALUES ('bhavdeep', 'sorathiya', 'Rajkot', 15);

ALTER TABLE persons add CONSTRAINT checkAge check (age >= 18 and age <= 50)

TRUNCATE TABLE persons

ALTER TABLE persons add CONSTRAINT checkAge check (age >= 18 and age <= 50)

INSERT into persons (fname, lname, city, age) VALUES ('bhavdeep', 'sorathiya', 'Rajkot', 15);

#4025 - CONSTRAINT `checkAge` failed for `729\_2324`.`persons`

# 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'  
);

drop TABLE persons

CREATE TABLE persons (id int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), city varchar(20) DEFAULT 'Rajkot', age int)

INSERT into persons (fname, lname, age) VALUES ('Udit', 'Ghetiya', 22)

Select \* from persons

The DEFAULT constraint can also be used to insert system values, by using functions like [CURRENT\_DATE()](https://www.w3schools.com/mysql/func_mysql_current_date.asp):

ALTER TABLE persons add COLUMN recordDate timestamp DEFAULT CURRENT\_TIMESTAMP

INSERT into persons (fname, lname, age) VALUES ('Udit', 'Ghetiya', 22)

Select \* from persons

ALTER TABLE persons add COLUMN state varchar(20) after city

ALTER TABLE persons MODIFY COLUMN state varchar(20) DEFAULT 'Gujarat'

INSERT into persons (fname, lname, age) VALUES ('Udit', 'Ghetiya', 22)

Select \* from persons

ALTER TABLE persons ALTER state DROP DEFAULT

INSERT into persons (fname, lname, age) VALUES ('Udit', 'Ghetiya', 22)

# 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 fnameData 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, ...);

ALTER TABLE students drop index fnamedata

CREATE UNIQUE index fnameData on 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.

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE persons (id int AUTO\_INCREMENT PRIMARY key, fname varchar(20), lname varchar(20), city varchar(20))

INSERT into persons (fname, lname, city) VALUES ('yadav', 'yagnik', 'rajkot')

Select \* from persons

ALTER TABLE persons AUTO\_INCREMENT = 1000

INSERT into persons (fname, lname, city) VALUES ('yadav', 'yagnik', 'rajkot')

SELECT \* FROM `persons`

# 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!

## Working with Dates

SELECT \* from students WHERE dateofbirth > '2004-12-31'

SELECT \* from students WHERE dateofbirth < '2004-12-31'

SELECT \* from students WHERE dateofbirth = '2004-02-11';

SELECT \* from students WHERE dateofbirth = '2004-2-11';

SELECT \* from students WHERE dateofbirth = '04-2-11';

SELECT \* from students WHERE dateofbirth BETWEEN '2000-01-01' and '2003-12-31'

SELECT \* from students WHERE dateofbirth BETWEEN '2000-01-01' and '2003-12-31' ORDER by (dateofbirth);

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

SELECT \* from students WHERE admissiondate = '2024-02-09'

SELECT \* from students WHERE admissiondate like '2024-02-09%';

SELECT \* from students WHERE admissiondate not like '2024-02-09%';

SELECT \* from students WHERE date(admissiondate) = '2024-02-09';

**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 \* from students

SELECT \* from getstudents

CREATE or REPLACE VIEW getstudents as SELECT students.roll, students.fname, students.lname, students.city, students.email, students.gender, students.dateofbirth, students.phone, students.fees, students.course, 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 \* FROM getstudents;

SELECT \* FROM getstudents WHERE roll = 5;

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

CREATE or REPLACE VIEW **`get students`** as SELECT students.roll, students.fname, students.lname, students.city, students.email, students.gender, students.dateofbirth, students.phone, students.fees, students.course, 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;

## 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;

## MySQL Dropping a View

A view is deleted with the DROP VIEW statement.

### DROP VIEW Syntax

DROP VIEW view\_name;

DROP view `get Students`

drop VIEW getstudents

# 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.

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. |

# MySQL Functions

MySQL has many built-in functions.

This reference contains string, numeric, date, and some advanced functions in MySQL.

## MySQL String Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ASCII](https://www.w3schools.com/mysql/func_mysql_ascii.asp) | Returns the ASCII value for the specific character |
| [CHAR\_LENGTH](https://www.w3schools.com/mysql/func_mysql_char_length.asp) | Returns the length of a string (in characters) |
| [CHARACTER\_LENGTH](https://www.w3schools.com/mysql/func_mysql_character_length.asp) | Returns the length of a string (in characters) |
| [CONCAT](https://www.w3schools.com/mysql/func_mysql_concat.asp) | Adds two or more expressions together |
| [CONCAT\_WS](https://www.w3schools.com/mysql/func_mysql_concat_ws.asp) | Adds two or more expressions together with a separator |
| [FIELD](https://www.w3schools.com/mysql/func_mysql_field.asp) | Returns the index position of a value in a list of values |
| [FIND\_IN\_SET](https://www.w3schools.com/mysql/func_mysql_find_in_set.asp) | Returns the position of a string within a list of strings |
| [FORMAT](https://www.w3schools.com/mysql/func_mysql_format.asp) | Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places |
| [INSERT](https://www.w3schools.com/mysql/func_mysql_insert.asp) | Inserts a string within a string at the specified position and for a certain number of characters |
| [INSTR](https://www.w3schools.com/mysql/func_mysql_instr.asp) | Returns the position of the first occurrence of a string in another string |
| [LCASE](https://www.w3schools.com/mysql/func_mysql_lcase.asp) | Converts a string to lower-case |
| [LEFT](https://www.w3schools.com/mysql/func_mysql_left.asp) | Extracts a number of characters from a string (starting from left) |
| [LENGTH](https://www.w3schools.com/mysql/func_mysql_length.asp) | Returns the length of a string (in bytes) |
| [LOCATE](https://www.w3schools.com/mysql/func_mysql_locate.asp) | Returns the position of the first occurrence of a substring in a string |
| [LOWER](https://www.w3schools.com/mysql/func_mysql_lower.asp) | Converts a string to lower-case |
| [LPAD](https://www.w3schools.com/mysql/func_mysql_lpad.asp) | Left-pads a string with another string, to a certain length |
| [LTRIM](https://www.w3schools.com/mysql/func_mysql_ltrim.asp) | Removes leading spaces from a string |
| [MID](https://www.w3schools.com/mysql/func_mysql_mid.asp) | Extracts a substring from a string (starting at any position) |
| [POSITION](https://www.w3schools.com/mysql/func_mysql_position.asp) | Returns the position of the first occurrence of a substring in a string |
| [REPEAT](https://www.w3schools.com/mysql/func_mysql_repeat.asp) | Repeats a string as many times as specified |
| [REPLACE](https://www.w3schools.com/mysql/func_mysql_replace.asp) | Replaces all occurrences of a substring within a string, with a new substring |
| [REVERSE](https://www.w3schools.com/mysql/func_mysql_reverse.asp) | Reverses a string and returns the result |
| [RIGHT](https://www.w3schools.com/mysql/func_mysql_right.asp) | Extracts a number of characters from a string (starting from right) |
| [RPAD](https://www.w3schools.com/mysql/func_mysql_rpad.asp) | Right-pads a string with another string, to a certain length |
| [RTRIM](https://www.w3schools.com/mysql/func_mysql_rtrim.asp) | Removes trailing spaces from a string |
| [SPACE](https://www.w3schools.com/mysql/func_mysql_space.asp) | Returns a string of the specified number of space characters |
| [STRCMP](https://www.w3schools.com/mysql/func_mysql_strcmp.asp) | Compares two strings |
| [SUBSTR](https://www.w3schools.com/mysql/func_mysql_substr.asp) | Extracts a substring from a string (starting at any position) |
| [SUBSTRING](https://www.w3schools.com/mysql/func_mysql_substring.asp) | Extracts a substring from a string (starting at any position) |
| [SUBSTRING\_INDEX](https://www.w3schools.com/mysql/func_mysql_substring_index.asp) | Returns a substring of a string before a specified number of delimiter occurs |
| [TRIM](https://www.w3schools.com/mysql/func_mysql_trim.asp) | Removes leading and trailing spaces from a string |
| [UCASE](https://www.w3schools.com/mysql/func_mysql_ucase.asp) | Converts a string to upper-case |
| [UPPER](https://www.w3schools.com/mysql/func_mysql_upper.asp) | Converts a string to upper-case |

Examples

select ascii('a');

select char\_length('Rajkot');

SELECT roll, fname, char\_length(fname) from students

SELECT roll, fname, character\_length(fname) from students;

SELECT roll, concat(fname, lname) as "Full Name" from students;

SELECT roll, concat(fname, " ",lname) as "Full Name" from students;

SELECT roll, concat\_ws(" ",fname,lname,city) as "Full Name" from students;

SELECT roll, concat\_ws(" - ",fname,lname,city) as "Full Name" from students;

SELECT field("s", "s", "q", "l")

SELECT field("k", "r", "a", "j", "k", "o", "t");

SELECT find\_in\_set("k", "r,a,j,k,o,t");

SELECT find\_in\_set("a", "r,a,j,k,o,t");

SELECT format(12345.6789, 2);

SELECT INSERT("W3Schools.com", 1, 9, "Example");

SELECT INSTR("W3Schools.com", "3") AS MatchPosition;

SELECT lcase(fname), lcase(lname), lcase(city) from students

SELECT lcase(concat\_ws(" ", fname, lname, city)) from students;

SELECT roll, fname, left(fname, 4) from students

SELECT roll, fname, right(fname, 4) from students;

SELECT roll, fname, length(fname) from students;

SELECT LOCATE("3", "W3Schools.com") AS MatchPosition;

SELECT roll, fname, lower(fname), lcase(fname) FROM students

SELECT roll, fname, lpad(fname, 20) FROM students;

SELECT roll, fname, rpad(fname, 20) FROM students;

SELECT ltrim(" Rajkot ")

SELECT rtrim(" Rajkot ");

SELECT mid("Rajkot", 1, 3)

SELECT POSITION("3" IN "W3Schools.com") AS MatchPosition;

SELECT fname, REPEAT(fname, 5) from students

SELECT REPLACE("SQL Tutorial", "SQL", "HTML");

SELECT roll, fname, reverse(fname) from students

SELECT space(10)

SELECT strcmp("Rajkot", "Rajkot")

SELECT strcmp("Rajkot", "tajkot");

SELECT strcmp("tajkot", "rajkot");

SELECT SUBSTR("SQL Tutorial", 5, 3) AS ExtractString;

SELECT SUBSTRING("SQL Tutorial", 5, 3) AS ExtractString;

SELECT SUBSTRING\_INDEX("www.w3schools.com", ".", 1);

SELECT trim(" www.w3schools.com ");

SELECT roll, fname, ucase(fname), upper(fname) FROM students

## MySQL Advanced Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [BIN](https://www.w3schools.com/mysql/func_mysql_bin.asp) | Returns a binary representation of a number |
| [BINARY](https://www.w3schools.com/mysql/func_mysql_binary.asp) | Converts a value to a binary string |
| [CASE](https://www.w3schools.com/mysql/func_mysql_case.asp) | Goes through conditions and return a value when the first condition is met |
| [CAST](https://www.w3schools.com/mysql/func_mysql_cast.asp) | Converts a value (of any type) into a specified datatype |
| [COALESCE](https://www.w3schools.com/mysql/func_mysql_coalesce.asp) | Returns the first non-null value in a list |
| [CONNECTION\_ID](https://www.w3schools.com/mysql/func_mysql_connection_id.asp) | Returns the unique connection ID for the current connection |
| [CONV](https://www.w3schools.com/mysql/func_mysql_conv.asp) | Converts a number from one numeric base system to another |
| [CONVERT](https://www.w3schools.com/mysql/func_mysql_convert.asp) | Converts a value into the specified datatype or character set |
| [CURRENT\_USER](https://www.w3schools.com/mysql/func_mysql_current_user.asp) | Returns the user name and host name for the MySQL account that the server used to authenticate the current client |
| [DATABASE](https://www.w3schools.com/mysql/func_mysql_database.asp) | Returns the name of the current database |
| [IF](https://www.w3schools.com/mysql/func_mysql_if.asp) | Returns a value if a condition is TRUE, or another value if a condition is FALSE |
| [IFNULL](https://www.w3schools.com/mysql/func_mysql_ifnull.asp) | Return a specified value if the expression is NULL, otherwise return the expression |
| [ISNULL](https://www.w3schools.com/mysql/func_mysql_isnull.asp) | Returns 1 or 0 depending on whether an expression is NULL |
| [LAST\_INSERT\_ID](https://www.w3schools.com/mysql/func_mysql_last_insert_id.asp) | Returns the AUTO\_INCREMENT id of the last row that has been inserted or updated in a table |
| [NULLIF](https://www.w3schools.com/mysql/func_mysql_nullif.asp) | Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned |
| [SESSION\_USER](https://www.w3schools.com/mysql/func_mysql_session_user.asp) | Returns the current MySQL user name and host name |
| [SYSTEM\_USER](https://www.w3schools.com/mysql/func_mysql_system_user.asp) | Returns the current MySQL user name and host name |
| [USER](https://www.w3schools.com/mysql/func_mysql_user.asp) | Returns the current MySQL user name and host name |
| [VERSION](https://www.w3schools.com/mysql/func_mysql_version.asp) | Returns the current version of the MySQL database |

SELECT bin(12)

SELECT BINARY "W3Schools.com";

SELECT roll, fname, lname, gender, case when gender = 'male' then 'Bike' when gender = 'female' then 'Scooter' else 'City Bus' end as "Vehicle" from students

SELECT cast(CURRENT\_TIMESTAMP as DATE)

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

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

SELECT CONNECTION\_ID();

SELECT CONV(15, 10, 2); --convert 15 decimal to binary

SELECT CONV(15, 10, 16);

SELECT CONV(1110011010101000101, 2, 10);

SELECT CURRENT\_USER()

SELECT SESSION\_USER()

SELECT SYSTEM\_USER();

SELECT USER();

SELECT DATABASE()

SELECT roll, if(roll > 5, "2024-02-25", "2024-02-26") as "Exam Date", fname, lname from students;

SELECT roll, absents, isnull(absents), presents, isnull(presents) from attendance;

SELECT \* from attendance WHERE isnull(absents);

SELECT \* from attendance WHERE isnull(absents) or isnull(presents);

SELECT last\_insert\_id()

SELECT NULLIF(25, "Hello");

SELECT NULLIF(25,25);

SELECT version()

## MySQL Numeric Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ABS](https://www.w3schools.com/mysql/func_mysql_abs.asp) | Returns the absolute value of a number |
| [ACOS](https://www.w3schools.com/mysql/func_mysql_acos.asp) | Returns the arc cosine of a number |
| [ASIN](https://www.w3schools.com/mysql/func_mysql_asin.asp) | Returns the arc sine of a number |
| [ATAN](https://www.w3schools.com/mysql/func_mysql_atan.asp) | Returns the arc tangent of one or two numbers |
| [ATAN2](https://www.w3schools.com/mysql/func_mysql_atan2.asp) | Returns the arc tangent of two numbers |
| [AVG](https://www.w3schools.com/mysql/func_mysql_avg.asp) | Returns the average value of an expression |
| [CEIL](https://www.w3schools.com/mysql/func_mysql_ceil.asp) | Returns the smallest integer value that is >= to a number |
| [CEILING](https://www.w3schools.com/mysql/func_mysql_ceiling.asp) | Returns the smallest integer value that is >= to a number |
| [COS](https://www.w3schools.com/mysql/func_mysql_cos.asp) | Returns the cosine of a number |
| [COT](https://www.w3schools.com/mysql/func_mysql_cot.asp) | Returns the cotangent of a number |
| [COUNT](https://www.w3schools.com/mysql/func_mysql_count.asp) | Returns the number of records returned by a select query |
| [DEGREES](https://www.w3schools.com/mysql/func_mysql_degrees.asp) | Converts a value in radians to degrees |
| [DIV](https://www.w3schools.com/mysql/func_mysql_div.asp) | Used for integer division |
| [EXP](https://www.w3schools.com/mysql/func_mysql_exp.asp) | Returns e raised to the power of a specified number |
| [FLOOR](https://www.w3schools.com/mysql/func_mysql_floor.asp) | Returns the largest integer value that is <= to a number |
| [GREATEST](https://www.w3schools.com/mysql/func_mysql_greatest.asp) | Returns the greatest value of the list of arguments |
| [LEAST](https://www.w3schools.com/mysql/func_mysql_least.asp) | Returns the smallest value of the list of arguments |
| [LN](https://www.w3schools.com/mysql/func_mysql_ln.asp) | Returns the natural logarithm of a number |
| [LOG](https://www.w3schools.com/mysql/func_mysql_log.asp) | Returns the natural logarithm of a number, or the logarithm of a number to a specified base |
| [LOG10](https://www.w3schools.com/mysql/func_mysql_log10.asp) | Returns the natural logarithm of a number to base 10 |
| [LOG2](https://www.w3schools.com/mysql/func_mysql_log2.asp) | Returns the natural logarithm of a number to base 2 |
| [MAX](https://www.w3schools.com/mysql/func_mysql_max.asp) | Returns the maximum value in a set of values |
| [MIN](https://www.w3schools.com/mysql/func_mysql_min.asp) | Returns the minimum value in a set of values |
| [MOD](https://www.w3schools.com/mysql/func_mysql_mod.asp) | Returns the remainder of a number divided by another number |
| [PI](https://www.w3schools.com/mysql/func_mysql_pi.asp) | Returns the value of PI |
| [POW](https://www.w3schools.com/mysql/func_mysql_pow.asp) | Returns the value of a number raised to the power of another number |
| [POWER](https://www.w3schools.com/mysql/func_mysql_power.asp) | Returns the value of a number raised to the power of another number |
| [RADIANS](https://www.w3schools.com/mysql/func_mysql_radians.asp) | Converts a degree value into radians |
| [RAND](https://www.w3schools.com/mysql/func_mysql_rand.asp) | Returns a random number |
| [ROUND](https://www.w3schools.com/mysql/func_mysql_round.asp) | Rounds a number to a specified number of decimal places |
| [SIGN](https://www.w3schools.com/mysql/func_mysql_sign.asp) | Returns the sign of a number |
| [SIN](https://www.w3schools.com/mysql/func_mysql_sin.asp) | Returns the sine of a number |
| [SQRT](https://www.w3schools.com/mysql/func_mysql_sqrt.asp) | Returns the square root of a number |
| [SUM](https://www.w3schools.com/mysql/func_mysql_sum.asp) | Calculates the sum of a set of values |
| [TAN](https://www.w3schools.com/mysql/func_mysql_tan.asp) | Returns the tangent of a number |
| [TRUNCATE](https://www.w3schools.com/mysql/func_mysql_truncate.asp) | Truncates a number to the specified number of decimal places |

SELECT abs(-123)

SELECT acos(0.3)

SELECT asin(0.3);

SELECT atan(0.3);

SELECT atan2(0.3);

SELECT avg(presents) from attendance;

SELECT ceil(12.34)

SELECT ceiling(12.34);

SELECT cos(0.12)

SELECT cot(0.12);

SELECT COUNT(roll) from students WHERE city = 'rajkot'

SELECT degrees(1.2)

SELECT 100 div 5

SELECT 100 mod 6;

SELECT exp(1)

SELECT floor(12.34)

SELECT greatest(11, 22, 33, 44, 55, 66, 77, 88, 99)

SELECT least(11, 22, 33, 44, 55, 66, 77, 88, 99);

SELECT ln(0.12)

SELECT log(0.12);

SELECT log2(0.12);

SELECT log10(0.12);

SELECT max(roll) FROM students

SELECT min(roll) FROM students;

SELECT pi()

SELECT pow(2, 10)

SELECT power(2, 10);

SELECT radians(180)

SELECT rand();

SELECT round(12345.6789, 6);

SELECT sign(10)

SELECT sin(0.12);

SELECT sqrt(144)

SELECT sum(absents) from attendance;

SELECT tan(0.12)

SELECT TRUNCATE(123.456, 1)

## MySQL Date Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ADDDATE](https://www.w3schools.com/mysql/func_mysql_adddate.asp) | Adds a time/date interval to a date and then returns the date |
| [ADDTIME](https://www.w3schools.com/mysql/func_mysql_addtime.asp) | Adds a time interval to a time/datetime and then returns the time/datetime |
| [CURDATE](https://www.w3schools.com/mysql/func_mysql_curdate.asp) | Returns the current date |
| [CURRENT\_DATE](https://www.w3schools.com/mysql/func_mysql_current_date.asp) | Returns the current date |
| [CURRENT\_TIME](https://www.w3schools.com/mysql/func_mysql_current_time.asp) | Returns the current time |
| [CURRENT\_TIMESTAMP](https://www.w3schools.com/mysql/func_mysql_current_timestamp.asp) | Returns the current date and time |
| [CURTIME](https://www.w3schools.com/mysql/func_mysql_curtime.asp) | Returns the current time |
| [DATE](https://www.w3schools.com/mysql/func_mysql_date.asp) | Extracts the date part from a datetime expression |
| [DATEDIFF](https://www.w3schools.com/mysql/func_mysql_datediff.asp) | Returns the number of days between two date values |
| [DATE\_ADD](https://www.w3schools.com/mysql/func_mysql_date_add.asp) | Adds a time/date interval to a date and then returns the date |
| [DATE\_FORMAT](https://www.w3schools.com/mysql/func_mysql_date_format.asp) | Formats a date |
| [DATE\_SUB](https://www.w3schools.com/mysql/func_mysql_date_sub.asp) | Subtracts a time/date interval from a date and then returns the date |
| [DAY](https://www.w3schools.com/mysql/func_mysql_day.asp) | Returns the day of the month for a given date |
| [DAYNAME](https://www.w3schools.com/mysql/func_mysql_dayname.asp) | Returns the weekday name for a given date |
| [DAYOFMONTH](https://www.w3schools.com/mysql/func_mysql_dayofmonth.asp) | Returns the day of the month for a given date |
| [DAYOFWEEK](https://www.w3schools.com/mysql/func_mysql_dayofweek.asp) | Returns the weekday index for a given date |
| [DAYOFYEAR](https://www.w3schools.com/mysql/func_mysql_dayofyear.asp) | Returns the day of the year for a given date |
| [EXTRACT](https://www.w3schools.com/mysql/func_mysql_extract.asp) | Extracts a part from a given date |
| [FROM\_DAYS](https://www.w3schools.com/mysql/func_mysql_from_days.asp) | Returns a date from a numeric datevalue |
| [HOUR](https://www.w3schools.com/mysql/func_mysql_hour.asp) | Returns the hour part for a given date |
| [LAST\_DAY](https://www.w3schools.com/mysql/func_mysql_last_day.asp) | Extracts the last day of the month for a given date |
| [LOCALTIME](https://www.w3schools.com/mysql/func_mysql_localtime.asp) | Returns the current date and time |
| [LOCALTIMESTAMP](https://www.w3schools.com/mysql/func_mysql_localtimestamp.asp) | Returns the current date and time |
| [MAKEDATE](https://www.w3schools.com/mysql/func_mysql_makedate.asp) | Creates and returns a date based on a year and a number of days value |
| [MAKETIME](https://www.w3schools.com/mysql/func_mysql_maketime.asp) | Creates and returns a time based on an hour, minute, and second value |
| [MICROSECOND](https://www.w3schools.com/mysql/func_mysql_microsecond.asp) | Returns the microsecond part of a time/datetime |
| [MINUTE](https://www.w3schools.com/mysql/func_mysql_minute.asp) | Returns the minute part of a time/datetime |
| [MONTH](https://www.w3schools.com/mysql/func_mysql_month.asp) | Returns the month part for a given date |
| [MONTHNAME](https://www.w3schools.com/mysql/func_mysql_monthname.asp) | Returns the name of the month for a given date |
| [NOW](https://www.w3schools.com/mysql/func_mysql_now.asp) | Returns the current date and time |
| [PERIOD\_ADD](https://www.w3schools.com/mysql/func_mysql_period_add.asp) | Adds a specified number of months to a period |
| [PERIOD\_DIFF](https://www.w3schools.com/mysql/func_mysql_period_diff.asp) | Returns the difference between two periods |
| [QUARTER](https://www.w3schools.com/mysql/func_mysql_quarter.asp) | Returns the quarter of the year for a given date value |
| [SECOND](https://www.w3schools.com/mysql/func_mysql_second.asp) | Returns the seconds part of a time/datetime |
| [SEC\_TO\_TIME](https://www.w3schools.com/mysql/func_mysql_sec_to_time.asp) | Returns a time value based on the specified seconds |
| [STR\_TO\_DATE](https://www.w3schools.com/mysql/func_mysql_str_to_date.asp) | Returns a date based on a string and a format |
| [SUBDATE](https://www.w3schools.com/mysql/func_mysql_subdate.asp) | Subtracts a time/date interval from a date and then returns the date |
| [SUBTIME](https://www.w3schools.com/mysql/func_mysql_subtime.asp) | Subtracts a time interval from a datetime and then returns the time/datetime |
| [SYSDATE](https://www.w3schools.com/mysql/func_mysql_sysdate.asp) | Returns the current date and time |
| [TIME](https://www.w3schools.com/mysql/func_mysql_time.asp) | Extracts the time part from a given time/datetime |
| [TIME\_FORMAT](https://www.w3schools.com/mysql/func_mysql_time_format.asp) | Formats a time by a specified format |
| [TIME\_TO\_SEC](https://www.w3schools.com/mysql/func_mysql_time_to_sec.asp) | Converts a time value into seconds |
| [TIMEDIFF](https://www.w3schools.com/mysql/func_mysql_timediff.asp) | Returns the difference between two time/datetime expressions |
| [TIMESTAMP](https://www.w3schools.com/mysql/func_mysql_timestamp.asp) | Returns a datetime value based on a date or datetime value |
| [TO\_DAYS](https://www.w3schools.com/mysql/func_mysql_to_days.asp) | Returns the number of days between a date and date "0000-00-00" |
| [WEEK](https://www.w3schools.com/mysql/func_mysql_week.asp) | Returns the week number for a given date |
| [WEEKDAY](https://www.w3schools.com/mysql/func_mysql_weekday.asp) | Returns the weekday number for a given date |
| [WEEKOFYEAR](https://www.w3schools.com/mysql/func_mysql_weekofyear.asp) | Returns the week number for a given date |
| [YEAR](https://www.w3schools.com/mysql/func_mysql_year.asp) | Returns the year part for a given date |
| [YEARWEEK](https://www.w3schools.com/mysql/func_mysql_yearweek.asp) | Returns the year and week number for a given date |

SELECT adddate(CURRENT\_DATE, INTERVAL 30 day)

SELECT adddate(CURRENT\_DATE, INTERVAL 30 month);

SELECT CURRENT\_TIMESTAMP, addtime(CURRENT\_TIMESTAMP, 2);

SELECT CURRENT\_DATE

SELECT CURRENT\_TIME

SELECT CURRENT\_TIMESTAMP

SELECT CURTIME()

SELECT year(CURRENT\_TIMESTAMP)

SELECT month(CURRENT\_TIMESTAMP);

SELECT day(CURRENT\_TIMESTAMP);

SELECT date(CURRENT\_TIMESTAMP);

SELECT hour(CURRENT\_TIMESTAMP);

SELECT minute(CURRENT\_TIMESTAMP);

SELECT second(CURRENT\_TIMESTAMP);

SELECT datediff(CURRENT\_DATE, "2001-04-26")

SELECT date\_add(CURRENT\_DATE, INTERVAL 20 day)

SELECT date\_format(CURRENT\_DATE, "%d / %m / %Y %h : %i : %s");

SELECT date\_sub(CURRENT\_TIMESTAMP, INTERVAL 50 month)

SELECT dayname(CURRENT\_TIMESTAMP)

SELECT dayofmonth(CURRENT\_TIMESTAMP);

SELECT dayofweek(CURRENT\_TIMESTAMP);

SELECT dayofyear(CURRENT\_TIMESTAMP);

SELECT extract(month from CURRENT\_TIMESTAMP)

SELECT from\_days(739303);

SELECT last\_day(CURRENT\_TIMESTAMP)

SELECT LOCALTIME

SELECT LOCALTIMESTAMP

SELECT makedate(2022, 5)

SELECT maketime(15, 15, 15)

SELECT microsecond(“2024-02-22 15:15:14.0384”)

SELECT monthname(CURRENT\_TIMESTAMP)

SELECT now()

SELECT period\_add(202401, 30);

SELECT period\_diff(202401, 202003);

SELECT quarter(CURRENT\_TIMESTAMP)

SELECT SEC\_TO\_TIME(3600);

SELECT STR\_TO\_DATE("August 10 2017", "%M %d %Y");

SELECT subdate(CURRENT\_TIMESTAMP, INTERVAL 10 year)

SELECT sysdate()

SELECT time(CURRENT\_TIMESTAMP)

SELECT TIME\_FORMAT(CURRENT\_TIMESTAMP, "%H %i %s");

SELECT TIME\_TO\_SEC(CURRENT\_TIMESTAMP);

SELECT TIMEDIFF("13:10:11", "13:10:10");

SELECT TIMESTAMP("2017-07-23",  "13:10:11");

SELECT to\_days(CURRENT\_TIMESTAMP)

SELECT week(CURRENT\_TIMESTAMP);

SELECT weekday(CURRENT\_TIMESTAMP);

SELECT weekofyear(CURRENT\_DATE);

SELECT yearweek(CURRENT\_DATE);

# Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

* Making relations very large.
* It isn't easy to maintain and update data as it would involve searching many records in relation.
* Wastage and poor utilization of disk space and resources.
* The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

## What is Normalization?

* Normalization is the process of organizing the data in the database.
* Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
* Normalization divides the larger table into smaller and links them using relationships.
* The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

**Data modification anomalies can be categorized into three types:**

* **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
* **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
* **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

## Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

**Following are the various types of Normal forms:**



|  |  |
| --- | --- |
| **Normal Form** | **Description** |
| [1NF](https://www.javatpoint.com/dbms-first-normal-form) | A relation is in 1NF if it contains an atomic value. |
| [2NF](https://www.javatpoint.com/dbms-second-normal-form) | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| [3NF](https://www.javatpoint.com/dbms-third-normal-form) | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| BCNF | A stronger definition of 3NF is known as Boyce Codd's normal form. |
| [4NF](https://www.javatpoint.com/dbms-forth-normal-form) | A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency. |
| [5NF](https://www.javatpoint.com/dbms-fifth-normal-form) | A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless. |

## Advantages of Normalization

* Normalization helps to minimize data redundancy.
* Greater overall database organization.
* Data consistency within the database.
* Much more flexible database design.
* Enforces the concept of relational integrity.

## Disadvantages of Normalization

* You cannot start building the database before knowing what the user needs.
* The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
* It is very time-consuming and difficult to normalize relations of a higher degree.
* Careless decomposition may lead to a bad database design, leading to serious problems.

# First Normal Form (1NF)

* A relation will be 1NF if it contains an atomic value.
* It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
* First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP\_PHONE.

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

The decomposition of the EMPLOYEE table into 1NF has been shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

# Second Normal Form (2NF)

* In the 2NF, relational must be in 1NF.
* In the second normal form, all non-key attributes are fully functional dependent on the primary key

**Example:** Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

**TEACHER table**

|  |  |  |
| --- | --- | --- |
| **TEACHER\_ID** | **SUBJECT** | **TEACHER\_AGE** |
| 25 | Chemistry | 30 |
| 25 | Biology | 30 |
| 47 | English | 35 |
| 83 | Math | 38 |
| 83 | Computer | 38 |

In the given table, non-prime attribute TEACHER\_AGE is dependent on TEACHER\_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

**TEACHER table:**

|  |  |
| --- | --- |
| **TEACHER\_ID (PK)** | **Firstname** |
| 25 | bhavdeep |
| 47 | Yagnik |
| 83 | udit |

**TEACHER\_DETAIL table:**

|  |  |  |
| --- | --- | --- |
| **DETAILS\_ID (PK)** | **TEACHER\_ID (FK)** | **TEACHER\_AGE** |
| 1 | 25 | 30 |
| 2 | 47 | 35 |
| 3 | 83 | 38 |

**TEACHER\_SUBJECT table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **SUBJECT** |
| 25 | Chemistry |
| 25 | Biology |
| 47 | English |
| 83 | Math |
| 83 | Computer |

# Third Normal Form (3NF)

* A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
* 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
* If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

X is a super key.

Y is a prime attribute, i.e., each element of Y is part of some candidate key.

**Example:**

**EMPLOYEE\_DETAIL table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 222 | Harry | 201010 | UP | Noida |
| 333 | Stephan | 02228 | US | Boston |
| 444 | Lan | 60007 | US | Chicago |
| 555 | Katharine | 06389 | UK | Norwich |
| 666 | John | 462007 | MP | Bhopal |

**Super key in the table above:**

{EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on

**Candidate key:** {EMP\_ID}

**Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.

Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.

That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.

**EMPLOYEE table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** |
| 222 | Harry | 201010 |
| 333 | Stephan | 02228 |
| 444 | Lan | 60007 |
| 555 | Katharine | 06389 |
| 666 | John | 462007 |
| 456 | Udit | 02228 |

**EMPLOYEE\_ZIP table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 201010 | UP | Noida |
| 02228 | US | Boston |
| 60007 | US | Chicago |
| 06389 | UK | Norwich |
| 462007 | MP | Bhopal |

# Boyce Codd normal form (BCNF)

* BCNF is the advance version of 3NF. It is stricter than 3NF.
* A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
* For BCNF, the table should be in 3NF, and for every FD, LHS is super key.
* **Example:** Let's assume there is a company where employees work in more than one department.
* **EMPLOYEE table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** | **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| 264 | India | Designing | D394 | 283 |
| 264 | India | Testing | D394 | 300 |
| 364 | UK | Stores | D283 | 232 |
| 364 | UK | Developing | D283 | 549 |

**In the above table Functional dependencies are as follows:**

1. EMP\_ID  →  EMP\_COUNTRY
2. EMP\_DEPT  →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate key: {EMP-ID, EMP-DEPT}**

The table is not in BCNF because neither EMP\_DEPT nor EMP\_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

**EMP\_COUNTRY table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** |
| 264 | India |
| 264 | India |

**EMP\_DEPT table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| Designing | D394 | 283 |
| Testing | D394 | 300 |
| Stores | D283 | 232 |
| Developing | D283 | 549 |

**EMP\_DEPT\_MAPPING table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_DEPT** |
| D394 | 283 |
| D394 | 300 |
| D283 | 232 |
| D283 | 549 |

1. EMP\_ID   →    EMP\_COUNTRY
2. EMP\_DEPT   →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate keys:**

**For the first table:** EMP\_ID   
**For the second table:** EMP\_DEPT  
**For the third table:** {EMP\_ID, EMP\_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

# Fourth normal form (4NF)

* A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
* For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

### Example

**STUDENT**

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **COURSE** | **HOBBY** |
| 21 | Computer | Dancing |
| 21 | Math | Singing |
| 34 | Chemistry | Dancing |
| 74 | Biology | Cricket |
| 59 | Physics | Hockey |

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU\_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

**STUDENT\_COURSE**

|  |  |
| --- | --- |
| **STU\_ID** | **COURSE** |
| 21 | Computer |
| 21 | Math |
| 34 | Chemistry |
| 74 | Biology |
| 59 | Physics |

**STUDENT\_HOBBY**

|  |  |
| --- | --- |
| **STU\_ID** | **HOBBY** |
| 21 | Dancing |
| 21 | Singing |
| 34 | Dancing |
| 74 | Cricket |
| 59 | Hockey |

# Fifth normal form (5NF)

* A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
* 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
* 5NF is also known as Project-join normal form (PJ/NF).

### Example

|  |  |  |
| --- | --- | --- |
| **SUBJECT** | **LECTURER** | **SEMESTER** |
| Computer | Anshika | Semester 1 |
| Computer | John | Semester 1 |
| Math | John | Semester 1 |
| Math | Akash | Semester 2 |
| Chemistry | Praveen | Semester 1 |

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

**P1**

|  |  |
| --- | --- |
| **SEMESTER** | **SUBJECT** |
| Semester 1 | Computer |
| Semester 1 | Math |
| Semester 1 | Chemistry |
| Semester 2 | Math |

**P2**

|  |  |
| --- | --- |
| **SUBJECT** | **LECTURER** |
| Computer | Anshika |
| Computer | John |
| Math | John |
| Math | Akash |
| Chemistry | Praveen |

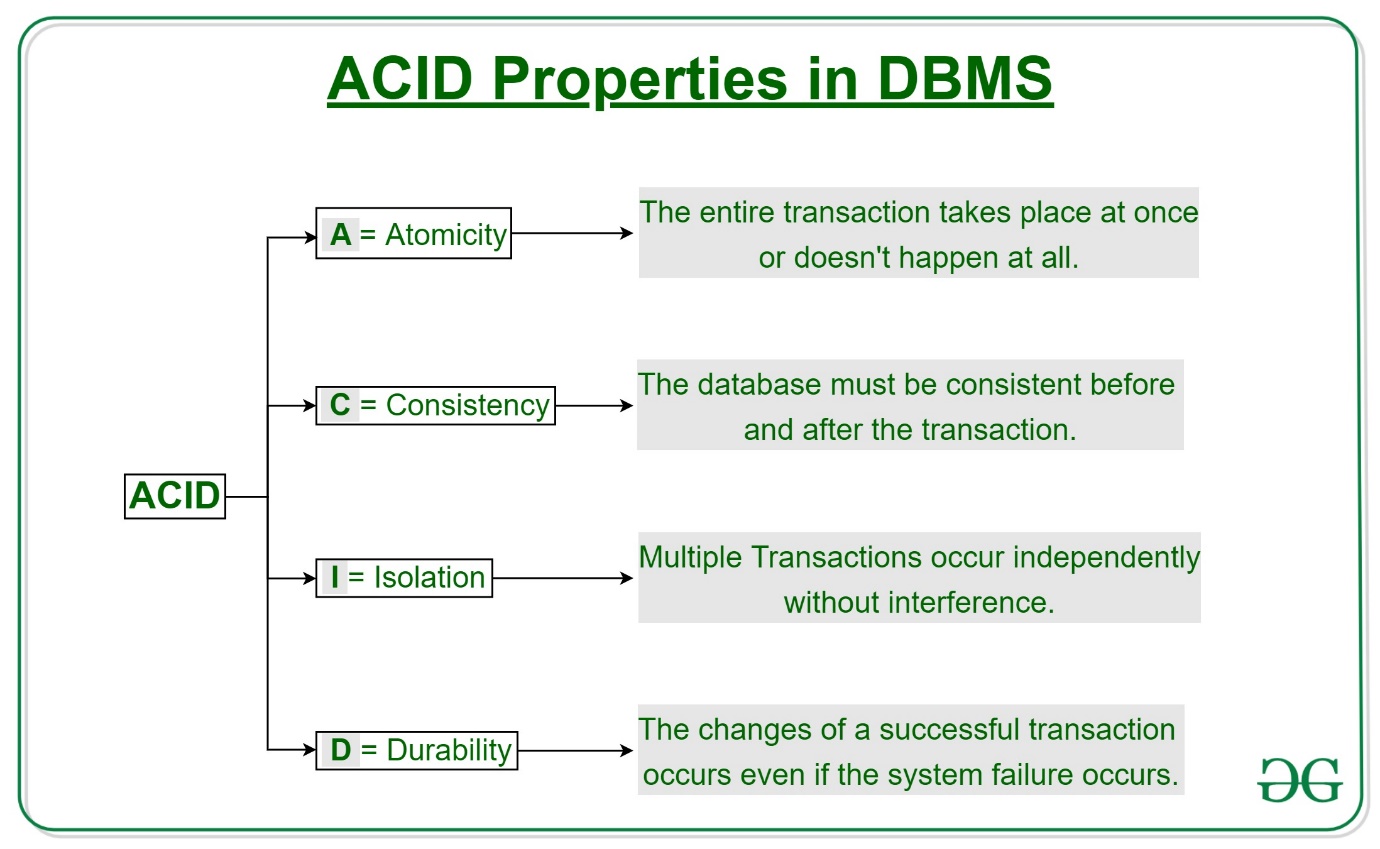
**P3**

|  |  |
| --- | --- |
| **SEMSTER** | **LECTURER** |
| Semester 1 | Anshika |
| Semester 1 | John |
| Semester 1 | John |
| Semester 2 | Akash |
| Semester 1 | Praveen |

# ACID Properties in DBMS

A [**transaction**](https://www.geeksforgeeks.org/sql-transactions/) 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 i.e. 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)**.

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

### 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:

**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.

**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.

**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.

**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
4. 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.