***MySQL***

**INTRODUCTION TO SQL**

SQL is a standard language for accessing and manipulating databases.

What is SQL?

SQL stands for Structured Query Language

Structured Query Language (SQL), which is a computer language for storing, manipulating, and retrieving data stored in relational database management systems (RDBMS).

SQL was developed at IBM by **Donald. C** in the 1970s.

## How Does MySQL Work?

MySQL is open-source and user-friendly. It creates a database to store and manipulate the data. To perform various operations users, make requests by typing specific statements. The server responds to the information from the user and displays it on the user side.

What Can SQL Do?

* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

### **Application of MySQL:**

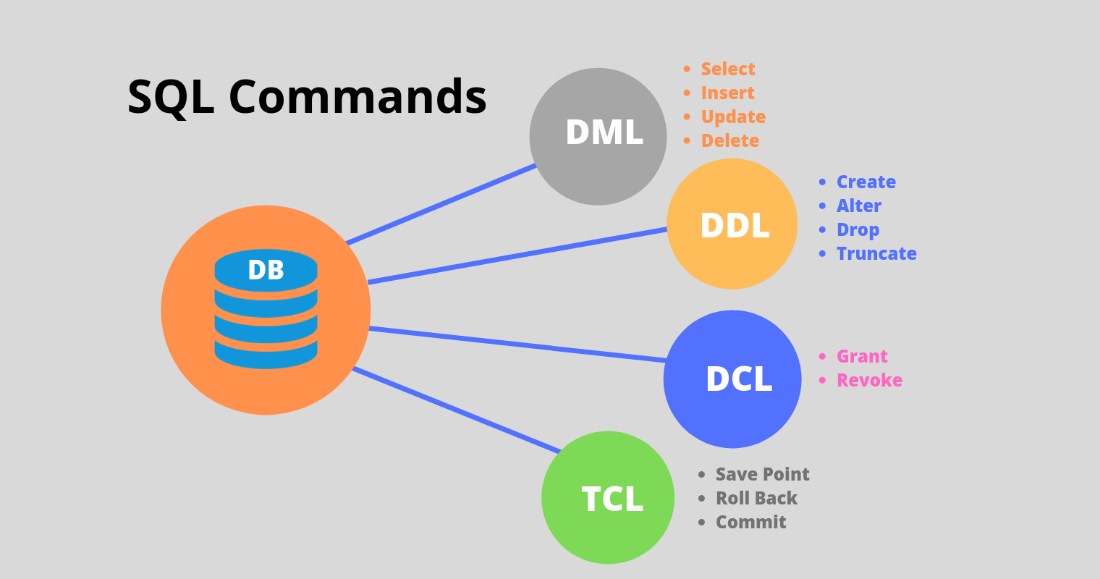
* MySQL is used in E-Commerce websites.
* MySQL is used in Data Warehousing.
* MySQL is used in the Login Application.

## What is a Database?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those type of systems. Nowadays, we use relational database management systems (RDBMS) to store and manage huge volumes of data.

Type of Databases Language:



**DDL (Data Definition Language)**

DDL, or Data Definition Language, is a subset of SQL (Structured Query Language) that is used to define and manage the structure of a relational database. In MySQL, DDL statements allow you to perform operations related to the database schema, such as creating or modifying tables, defining constraints, and altering the database structure. Here are some key DDL statements in MySQL:

### **1. CREATE:**

The **CREATE** statement is used to create database objects, such as tables, indexes, or views.

* **Creating a Table:**
* CREATE TABLE employees (  
   id INT PRIMARY KEY,  
   name VARCHAR(50),  
   salary DECIMAL(10,2)  
  );

### **2. ALTER:**

The **ALTER** statement is used to modify the structure of an existing database object, such as adding or dropping columns.

* **Adding a Column to a Table:**

ALTER TABLE employees  
 ADD COLUMN hire\_date DATE;

### **3. DROP:**

The **DROP** statement is used to remove database objects, such as tables or indexes.

* **Dropping a Table:**

DROP TABLE employees;

### **4. TRUNCATE:**

The **TRUNCATE** statement is used to remove all records from a table but retain the table structure for further use.

* **Truncating a Table:**

TRUNCATE TABLE employees;

**DML (Data Manipulation Language)**

DML, or Data Manipulation Language, is a subset of SQL (Structured Query Language) that deals with the manipulation and retrieval of data stored in a relational database. DML operations are responsible for interacting with the actual data within the database, allowing users to insert, update, retrieve, and delete records. Here are some key DML statements in SQL, including how they are used in MySQL:

### **1. SELECT:**

The **SELECT** statement is used to retrieve data from one or more tables. It is a fundamental part of querying and fetching information from a database.

* **Basic SELECT Query:**

SELECT column1, column2  
FROM table\_name  
WHERE condition;

### **2. INSERT:**

The **INSERT** statement is used to add new records (rows) to a table.

* **Inserting a Single Record:**

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

* **Inserting Multiple Records:**

INSERT INTO table\_name (column1, column2)  
VALUES  
 (value1a, value2a),  
 (value1b, value2b),  
 (value1c, value2c);

-- Example: Inserting Multiple Records into the 'employees' Table

INSERT INTO employees (id, name, salary)

VALUES

(1, 'John Doe', 50000),

(2, 'Jane Smith', 60000),

(3, 'Bob Johnson', 55000),

(4, 'Alice Brown', 70000);

### **3. UPDATE:**

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

* **Updating Records:**

UPDATE table\_name  
SET column1 = new\_value1, column2 = new\_value2  
WHERE condition;

### **4. DELETE:**

The **DELETE** statement is used to remove records from a table.

* **Deleting Records:**

DELETE FROM table\_name  
WHERE condition;

## **DCL (Data Control Language)**

DCL, or Data Control Language, is a subset of SQL (Structured Query Language) used for controlling access to the data within a relational database management system (RDBMS). DCL includes statements that grant or revoke permissions, allowing database administrators to manage user access rights and control the security of the database. The primary DCL statements are **GRANT** and **REVOKE**.

### **1. GRANT:**

The **GRANT** statement is used to give specific privileges or permissions to users or roles. These privileges allow users to perform certain actions on database objects, such as tables or views.

#### Syntax:

GRANT privilege\_name  
ON object\_name  
TO {user\_name | PUBLIC | role\_name}  
[WITH GRANT OPTION];

* **Example: Grant SELECT on a Table:**

GRANT SELECT ON employees TO user1;

### **2. REVOKE:**

The **REVOKE** statement is used to take back or remove previously granted privileges from users or roles.

#### Syntax:

REVOKE privilege\_name  
ON object\_name  
FROM {user\_name | PUBLIC | role\_name};

* **Example: Revoke SELECT privilege:**

REVOKE SELECT ON employees FROM user1;

## **TCL (Transaction Control Language)**

TCL, or Transaction Control Language, is a subset of SQL (Structured Query Language) that deals with the management and control of transactions in a relational database management system (RDBMS). Transactions are sequences of one or more SQL statements that are executed as a single unit, ensuring the consistency, integrity, and durability of the database. TCL statements are used to initiate, commit, or roll back transactions. The primary TCL statements are **COMMIT**, **ROLLBACK**, and **SAVEPOINT**.

### **1. COMMIT:**

The **COMMIT** statement is used to permanently save the changes made during the current transaction. Once a **COMMIT** statement is executed, the changes become permanent, and they cannot be undone.

#### Syntax:

COMMIT;

* **Example: Committing a Transaction:**

-- Start a transaction  
BEGIN;  
  
-- SQL statements within the transaction  
UPDATE accounts SET balance = balance - 100 WHERE user\_id = 1;  
INSERT INTO transaction\_log (user\_id, amount) VALUES (1, -100);  
  
-- Commit the transaction  
COMMIT;

### **2. ROLLBACK:**

The **ROLLBACK** statement is used to undo the changes made during the current transaction. It is often used when an error occurs or when there is a need to discard the changes for any reason.

#### Syntax:

ROLLBACK;

* **Example: Rolling Back a Transaction:**

-- Start a transaction  
BEGIN;  
  
-- SQL statements within the transaction  
UPDATE accounts SET balance = balance - 100 WHERE user\_id = 1;  
INSERT INTO transaction\_log (user\_id, amount) VALUES (1, -100);  
  
-- Roll back the transaction (undo changes)  
ROLLBACK;

### **3. SAVEPOINT:**

The **SAVEPOINT** statement is used to set a named savepoint within a transaction. This allows you to later roll back to that savepoint without rolling back the entire transaction.

#### Syntax:

SAVEPOINT savepoint\_name;

* **Example: Using SAVEPOINT and ROLLBACK TO:**

-- Start a transaction  
BEGIN;  
  
-- SQL statements within the transaction  
UPDATE accounts SET balance = balance - 100 WHERE user\_id = 1;  
SAVEPOINT update\_savepoint;  
INSERT INTO transaction\_log (user\_id, amount) VALUES (1, -100);  
  
-- Roll back to the savepoint (undo changes after the savepoint)  
ROLLBACK TO update\_savepoint;  
  
-- Commit the transaction  
COMMIT;

## RDBMS Terminology:

let us revise a few definitions related to the database.

* **Database** − A database is a collection of tables with related data.
* **Table** − A table is a matrix with data. A table in a database looks like a simple spreadsheet.
* **Column** − One column (data element) contains data of one and the same kind, for example the column postcode.
* **Row** − A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.
* **Redundancy** − Storing data twice, redundantly to make the system faster.
* **Primary Key** − A primary key is unique. A key value cannot occur twice in one table. With a key, you can only find one row.
* **Foreign Key** − A foreign key is the linking pin between two tables.
* **Compound Key** − A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
* **Index** − An index in a database resembles an index at the back of a book.

**Comments**

In MySQL, comments are used to provide explanations or annotations within SQL code. Comments have no impact on the execution of the SQL statements; they are ignored by the database engine. Comments are useful for documenting the code, making it more readable, and explaining the purpose of certain sections or lines. MySQL supports two types of comments:

### **1. Single-Line Comments:**

Single-line comments start with the **--** (double hyphen) syntax and continue to the end of the line. Anything after **--** is treated as a comment.

#### Example:

-- This is a single-line comment  
SELECT \* FROM employees; -- Another comment on the same line

### **2. Multi-Line Comments:**

Multi-line comments start with **/\*** and end with **\*/**. Everything between **/\*** and **\*/** is treated as a comment, and it can span multiple lines.

#### Example:

/\*  
This is a multi-line comment.  
It can span across multiple lines.  
\*/  
  
SELECT \* FROM customers;

**MySQL DATA Types:**

### **1. Numeric Types:**

* **INTEGER (or INT):**
* A whole number without a fractional component.
* Example:

CREATE TABLE employees (  
 employee\_id INT,  
 salary INT  
);

* **DECIMAL:**
* Fixed-point precision number. It is suitable for storing monetary values.
* Example:

CREATE TABLE products (  
 product\_id INT,  
 price DECIMAL(10,2)  
);

* **FLOAT:**
* A floating-point number with a decimal component.
* Example:

CREATE TABLE sensor\_data (  
 sensor\_id INT,  
 temperature FLOAT  
);

### **2. String Types:**

* **CHAR:**
* Fixed-length character string.
* Example:

CREATE TABLE customers (  
 customer\_id INT,  
 customer\_name CHAR(50)  
);

* **VARCHAR:**
* Variable-length character string.
* Example:

CREATE TABLE orders (  
 order\_id INT,  
 customer\_name VARCHAR(50)  
);

* **TEXT:**
* Variable-length text string for large data.
* Example:

CREATE TABLE articles (  
 article\_id INT,  
 article\_content TEXT  
);

### **3. Date and Time Types:**

* **DATE:**
* Date value in 'YYYY-MM-DD' format.
* Example:

CREATE TABLE events (  
 event\_id INT,  
 event\_date DATE  
);

* **DATETIME:**
* Date and time value in 'YYYY-MM-DD HH:MM:SS' format.
* Example:

CREATE TABLE log\_entries (  
 log\_id INT,  
 log\_timestamp DATETIME  
);

* **TIMESTAMP:**
* A timestamp representing the number of seconds since the epoch (1970-01-01 00:00:00).
* Example:

CREATE TABLE posts (  
 post\_id INT,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

### **4. Boolean Type:**

* **BOOLEAN (or BOOL):**
* A true/false or 1/0 value.
* Example:

CREATE TABLE tasks (  
 task\_id INT,  
 task\_name VARCHAR(50),  
 is\_completed BOOLEAN  
);

### **5. Binary Types:**

* **BINARY:**
* Fixed-length binary string.
* Example:

CREATE TABLE images (  
 image\_id INT,  
 image\_data BINARY  
);

* **VARBINARY:**
* Variable-length binary string.
* Example:

CREATE TABLE files (  
 file\_id INT,  
 file\_data VARBINARY(255)  
);

**MySQL Syntax**

# Create Database

A database is used to store the collection of records in an organized form. It allows us to hold the data into tables, rows, columns, and indexes to find the relevant information frequently. We can access and manage the records through the database very easily.

Syntax:

CREATE DATABASE *databasename*;

## DROP DATABASE

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

### Syntax

DROP DATABASE *databasename*;

## CREATE TABLE Statement

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

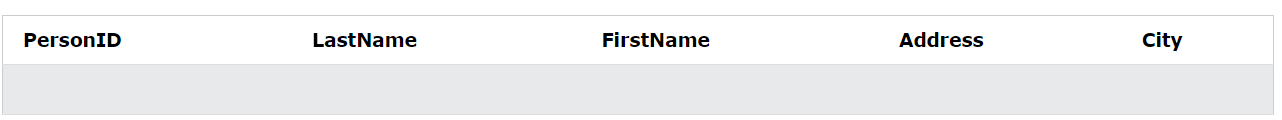
### Syntax

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

## Example

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

Output:



## DROP TABLE Statement

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

### Syntax

DROP TABLE *table\_name*;

### Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

DROP TABLE Shippers;

## TRUNCATE TABLE

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

### Syntax

TRUNCATE TABLE *table\_name*;

## 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.

### Syntax

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

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

SELECT \* FROM *table\_name*;

### Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

SELECT CustomerName, City, Country FROM Customers;

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

### Example

SELECT \* FROM Customers;

## WHERE Clause

The WHERE clause is used to filter records.

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

### Syntax

SELECT *column1*, *column2, ...*  
*FROM* *table\_name*  
*WHERE* *condition*;



### Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

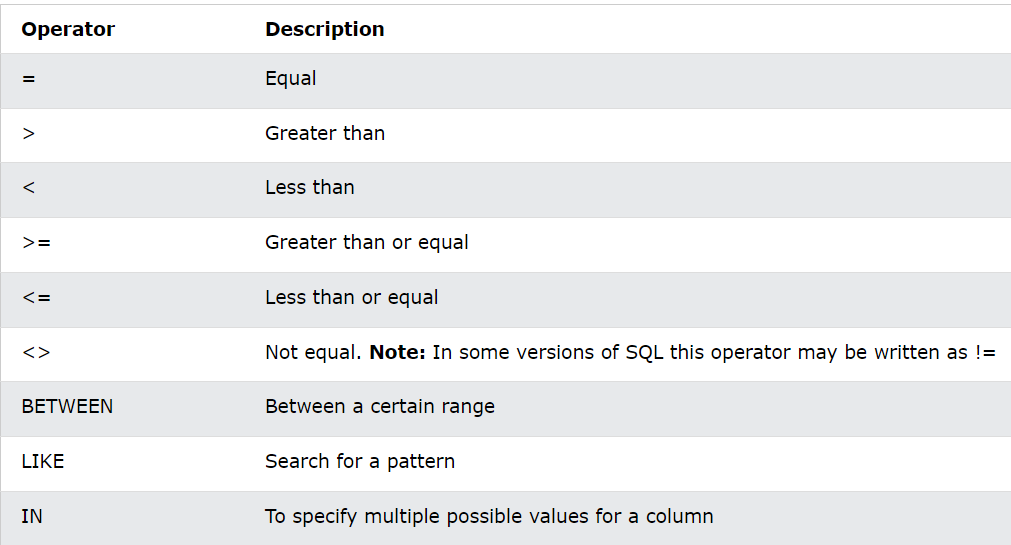
SELECT \* FROM Customers  
WHERE Country = 'Mexico';

Or

### Example

SELECT \* FROM Customers  
WHERE CustomerID = 1;

The following operators can be used in the WHERE clause:



1) Equal

SELECT \* FROM Products

WHERE Price = 18;

2) Greater than

SELECT \* FROM Products

WHERE Price > 30;

3) Less than

SELECT \* FROM Products

WHERE Price < 30;

4) Greater than or equal

SELECT \* FROM Products

WHERE Price >= 30;

5) Less than or equal

SELECT \* FROM Products

WHERE Price <= 30;

6) Not equal

SELECT \* FROM Products

WHERE Price <> 18;

7) Between

SELECT \* FROM Products

WHERE Price BETWEEN 50 AND 60;

8) LIKE

SELECT \* FROM Customers

WHERE City LIKE 's%';

9) IN

SELECT \* FROM Customers

WHERE City IN ('Paris', 'London');

## 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*;



## AND Example

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

### Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

SELECT \* FROM Customers  
WHERE Country = 'Germany' AND City = 'Berlin';

## OR Example

The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "Stuttgart":

### Example

SELECT \* FROM Customers  
WHERE City = 'Berlin' OR City = 'Stuttgart';

The following SQL statement selects all fields from "Customers" where country is "Germany" OR "Spain":

### Example

SELECT \* FROM Customers  
WHERE Country = 'Germany' OR Country = 'Spain';

## NOT Example

The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

### Example

SELECT \* FROM Customers  
WHERE NOT Country = 'Germany';

## ORDER BY DESC Example

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

### Example

SELECT \* FROM Customers  
ORDER BY Country DESC;

## 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

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

### **MYSQL Constraint**

In MySQL, constraints are rules or conditions applied to columns or tables to enforce data integrity. They define limits on the values that columns can take, ensuring that the data in the database remains accurate and consistent. Here are some common types of constraints in MySQL along with examples:

### 1. PRIMARY KEY Constraint:

The PRIMARY KEY constraint is used to uniquely identify each record in a table. It must contain unique values, and it cannot contain NULL values.

#### Example:

CREATE TABLE students (  
 student\_id INT PRIMARY KEY,  
 student\_name VARCHAR(50),  
 age INT  
);

### 2. FOREIGN KEY Constraint:

The FOREIGN KEY constraint is used to link two tables together by referencing the primary key of one table as a foreign key in another. It ensures referential integrity.

#### Example:

CREATE TABLE courses (  
 course\_id INT PRIMARY KEY,  
 course\_name VARCHAR(50)  
);  
  
CREATE TABLE student\_enrollments (  
 enrollment\_id INT PRIMARY KEY,  
 student\_id INT,  
 course\_id INT,  
 FOREIGN KEY (student\_id) REFERENCES students(student\_id),  
 FOREIGN KEY (course\_id) REFERENCES courses(course\_id)  
);

### 3. UNIQUE Constraint:

The UNIQUE constraint ensures that all values in a column are unique. It can be applied to one or more columns.

#### Example:

CREATE TABLE employees (  
 employee\_id INT PRIMARY KEY,  
 employee\_email VARCHAR(50) UNIQUE,  
 employee\_name VARCHAR(50)  
);

### 4. CHECK Constraint:

The CHECK constraint enforces a condition on the values allowed in a column.

#### Example:

CREATE TABLE orders (  
 order\_id INT PRIMARY KEY,  
 order\_date DATE,  
 order\_amount DECIMAL(10,2),  
 CHECK (order\_amount >= 0) -- Ensure order\_amount is non-negative  
);

### 5. NOT NULL Constraint:

The NOT NULL constraint ensures that a column cannot contain NULL values.

#### Example:

CREATE TABLE customers (  
 customer\_id INT PRIMARY KEY,  
 customer\_name VARCHAR(50) NOT NULL,  
 customer\_email VARCHAR(50) UNIQUE,  
 phone\_number VARCHAR(15) NOT NULL  
);

### 6. DEFAULT Constraint:

The DEFAULT constraint provides a default value for a column if no value is specified during an INSERT operation.

#### Example:

CREATE TABLE products (  
 product\_id INT PRIMARY KEY,  
 product\_name VARCHAR(50) NOT NULL,  
 price DECIMAL(10,2) DEFAULT 0.00  
);

### 7. INDEX Constraint:

While not strictly a constraint, an INDEX is used to create an index on one or more columns, which can improve query performance.

#### Example:

CREATE TABLE books (  
 book\_id INT PRIMARY KEY,  
 title VARCHAR(100),  
 author VARCHAR(50),  
 publication\_year INT,  
 INDEX idx\_title (title),  
 INDEX idx\_author\_year (author, publication\_year)  
);

### **CRUD Operations**

* **Create (INSERT):** Adding new records to a table.

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

* **Read (SELECT):** Retrieving data from one or more tables.

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

* **Update (UPDATE):** Modifying existing records in a table.

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

* **Delete (DELETE):** Removing records from a table.

DELETE FROM table\_name WHERE condition;

**MySQL Functions**

### **1. String Functions:**

#### **CONCAT()**: Concatenates two or more strings.

sqlCopy code

SELECT CONCAT('Hello', ' ', 'World') AS concatenated\_string;  
-- Output: 'Hello World'

#### **SUBSTRING()**: Returns a substring from a string.

sqlCopy code

SELECT SUBSTRING('MySQL Functions', 1, 5) AS substring\_result;  
-- Output: 'MySQL'

#### **UPPER()** and **LOWER()**: Converts a string to uppercase or lowercase.

sqlCopy code

SELECT UPPER('hello') AS upper\_case, LOWER('WORLD') AS lower\_case;  
-- Output: 'HELLO', 'world'

#### **LENGTH()**: Returns the length of a string.

sqlCopy code

SELECT LENGTH('MySQL') AS string\_length;  
-- Output: 5

### **2. Numeric Functions:**

#### **SUM()**, **AVG()**, **MIN()**, **MAX()**: Aggregate functions for numeric data.

sqlCopy code

SELECT SUM(salary) AS total\_salary, AVG(salary) AS average\_salary,  
 MIN(salary) AS min\_salary, MAX(salary) AS max\_salary  
FROM employees;

#### **ABS()**: Returns the absolute value of a number.

sqlCopy code

SELECT ABS(-10) AS absolute\_value;  
-- Output: 10

#### **ROUND()**: Rounds a number to a specified number of decimal places.

sqlCopy code

SELECT ROUND(15.789, 1) AS rounded\_number;  
-- Output: 15.8

### **3. Date and Time Functions:**

#### **NOW()**: Returns the current date and time.

sqlCopy code

SELECT NOW() AS current\_datetime;

#### **DATE()**: Extracts the date part from a datetime expression.

sqlCopy code

SELECT DATE('2022-01-15 18:30:45') AS extracted\_date;  
-- Output: '2022-01-15'

#### **TIMESTAMPDIFF()**: Calculates the difference between two timestamps.

sqlCopy code

SELECT TIMESTAMPDIFF(MINUTE, '2022-01-15 12:00:00', '2022-01-15 12:45:00') AS time\_difference\_in\_minutes;  
-- Output: 45

### **4. Mathematical Functions:**

#### **SQRT()**: Returns the square root of a number.

sqlCopy code

SELECT SQRT(25) AS square\_root;  
-- Output: 5

#### **POWER()**: Raises a number to a specified power.

sqlCopy code

SELECT POWER(2, 3) AS raised\_to\_power;  
-- Output: 8

#### **RAND()**: Returns a random floating-point value between 0 and 1.

sqlCopy code

SELECT RAND() AS random\_value;

### **5. Conditional Functions:**

#### **IF()**: Returns a value depending on whether a condition is true or false.

sqlCopy code

SELECT IF(salary > 50000, 'High Salary', 'Low Salary') AS salary\_category  
FROM employees;

#### **CASE WHEN**: Provides conditional logic in a more elaborate form.

sqlCopy code

SELECT   
 CASE   
 WHEN salary > 80000 THEN 'Very High Salary'  
 WHEN salary > 50000 THEN 'High Salary'  
 ELSE 'Low Salary'  
 END AS salary\_category  
FROM employees;

**Aggregate functions**

Aggregate functions in MySQL are used to perform calculations on a set of values and return a single result. Here are some commonly used aggregate functions along with examples:

### **1. COUNT():**

Counts the number of rows in a result set.

**Example:**

sqlCopy code

SELECT COUNT(employee\_id) AS total\_employees FROM employees;  
-- Output: Number of rows in the 'employees' table

### **2. SUM():**

Calculates the sum of a numeric column.

**Example:**

sqlCopy code

SELECT SUM(salary) AS total\_salary FROM employees;  
-- Output: Sum of salaries in the 'employees' table

### **3. AVG():**

Calculates the average (mean) of a numeric column.

**Example:**

sqlCopy code

SELECT AVG(salary) AS average\_salary FROM employees;  
-- Output: Average salary in the 'employees' table

### **4. MIN():**

Returns the minimum value in a numeric column.

**Example:**

sqlCopy code

SELECT MIN(salary) AS min\_salary FROM employees;  
-- Output: Minimum salary in the 'employees' table

### **5. MAX():**

Returns the maximum value in a numeric column.

**Example:**

sqlCopy code

SELECT MAX(salary) AS max\_salary FROM employees;  
-- Output: Maximum salary in the 'employees' table

**SQL JOIN**

 SQL joins are used to query data from two or more tables, based on a relationship between certain

columns in these tables.

 The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a

relationship between certain columns in these tables.

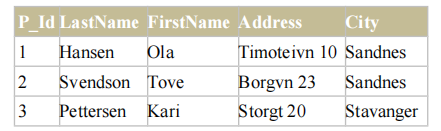
 Tables in a database are often related to each other with keys.

 A primary key is a column (or a combination of columns) with a unique value for each row. Each

primary key value must be unique within the table. The purpose is to bind data together, across

tables, without repeating all of the data in every table.

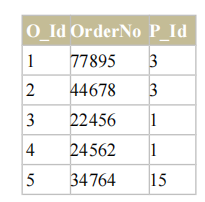
Look at the "Persons" table:



Note that the "P\_Id" column is the primary key in the "Persons" table. This means that **no** two rows can

have the same P\_Id. The P\_Id distinguishes two persons even if they have the same name.

Next, we have the "Orders" table:



Note that the "O\_Id" column is the primary key in the "Orders" table and that the "P\_Id" column refers to

the persons in the "Persons" table without using their names.

Notice that the relationship between the two tables above is the "P\_Id" column.

**Different SQL JOINs**

Before we continue with examples, we will list the types of JOIN you can use, and the differences

between them.

**JOIN**: Return rows when there is at least one match in both tables

**LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table

**RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table

**FULL JOIN**: Return rows when there is a match in one of the tables

**SQL INNER JOIN Keyword**

 The INNER JOIN keyword returns rows when there is at least one match in both tables.

 SQL INNER JOIN Syntax:

SELECT column\_name(s)

FROM table\_name1

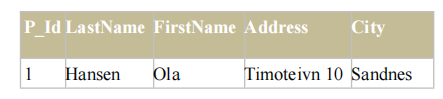
INNER JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

 **PS:** INNER JOIN is the same as JOIN.

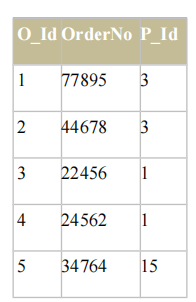
**SQL INNER JOIN Example**

The "Persons" table:





The "Orders" table:



Now we want to list all the persons with any orders.

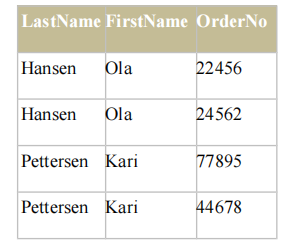
We use the following SELECT statement:

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons INNER JOIN Orders ON Persons.P\_Id=Orders.P\_Id

ORDER BY Persons.LastName

The result-set will look like this:



The INNER JOIN keyword return rows when there is at least one match in both tables. If there are rows

in "Persons" that do not have matches in "Orders", those rows will NOT be listed.

**SQL LEFT JOIN Keyword**

 The LEFT JOIN keyword returns all rows from the left table (table\_name1), even if there are no

matches in the right table (table\_name2).

 SQL LEFT JOIN Syntax:

SELECT column\_name(s)

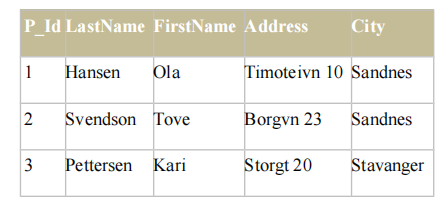
FROM table\_name1 LEFT JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

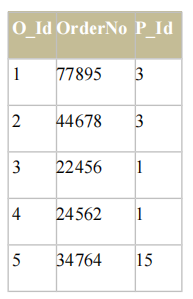
 **PS:** In some databases LEFT JOIN is called LEFT OUTER JOIN.

**SQL LEFT JOIN Example**

The "Persons" table:



The "Orders" table:



Now we want to list all the persons and their orders - if any, from the tables above.

We use the following SELECT statement:

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons LEFT JOIN Orders

ON Persons.P\_Id=Orders.P\_Id

ORDER BY Persons.LastName

The result-set will look like this:



**Notes:** The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no

matches in the right table (Orders).

**SQL RIGHT JOIN Keyword**

 The RIGHT JOIN keyword Return all rows from the right table (table\_name2), even if there are

no matches in the left table (table\_name1).

 SQL RIGHT JOIN Syntax:

SELECT column\_name(s)

FROM table\_name1

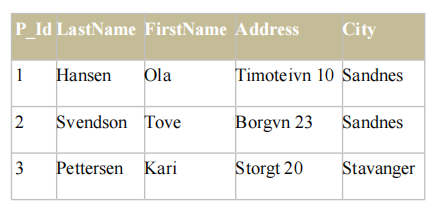
RIGHT JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

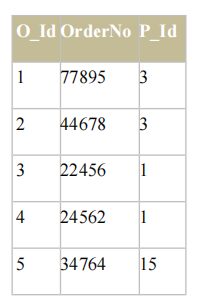
 **PS:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

**SQL RIGHT JOIN Example**

The "Persons" table:



The "Orders" table:



Now we want to list all the orders with containing persons - if any, from the tables above.

We use the following SELECT statement:

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

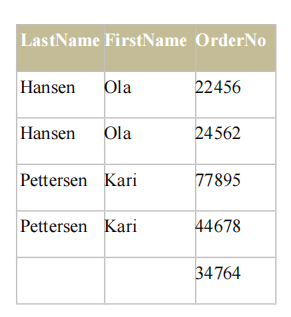
FROM Persons

RIGHT JOIN Orders

ON Persons.P\_Id=Orders.P\_Id

ORDER BY Persons.LastName

The result-set will look like this:



**Notes:** The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no

matches in the left table (Persons).

**SQL FULL JOIN Keyword**

 The FULL JOIN keyword return rows when there is a match in one of the tables.

 SQL FULL JOIN Syntax:

SELECT column\_name(s)

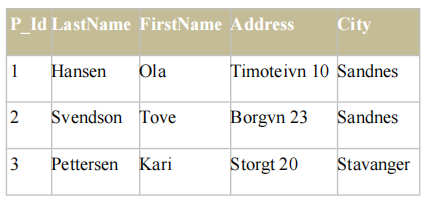
FROM table\_name1

FULL JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

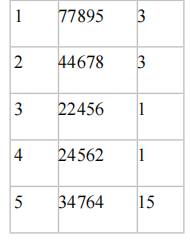
**SQL FULL JOIN Example**

The "Persons" table:



The "Orders" table:





Now we want to list all the persons and their orders, and all the orders with their persons.

We use the following SELECT statement:

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

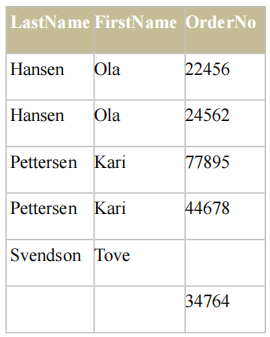
FROM Persons

FULL JOIN Orders

ON Persons.P\_Id=Orders.P\_Id

ORDER BY Persons.LastName

The result-set will look like this:



**Notes:** The FULL JOIN keyword returns all the rows from the left table (Persons), and all the rows from

the right table (Orders). If there are rows in "Persons" that do not have matches in "Orders", or if there are

rows in "Orders" that do not have matches in "Persons", those rows will be listed as well.

**The SQL UNION Operator**

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

 Notice that each SELECT statement within the UNION must have the same number of columns.

The columns must also have similar data types. Also, the columns in each SELECT statement

must be in the same order.

 SQL UNION Syntax:

SELECT column\_name(s) FROM table\_name1

UNION

SELECT column\_name(s) FROM table\_name2

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

use UNION ALL.

 SQL UNION ALL Syntax:

SELECT column\_name(s) FROM table\_name1

UNION ALL

SELECT column\_name(s) FROM table\_name2

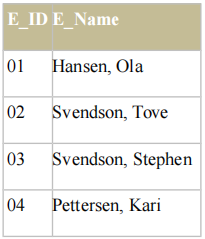
 **PS:** The column names in the result-set of a UNION are always equal to the column names in the

first SELECT statement in the UNION.

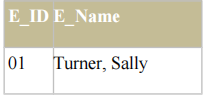
**SQL UNION Example**

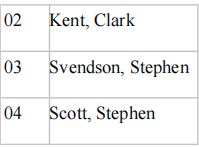
Look at the following tables:

**"Employees\_Norway"**:



**"Employees\_USA"**:





Now we want to list **all the different** employees in Norway and USA.

We use the following SELECT statement:

SELECT E\_Name FROM Employees\_Norway

UNION

SELECT E\_Name FROM Employees\_USA

The result-set will look like this:



**Note:** This command cannot be used to list all employees in Norway and USA. In the example above we

have two employees with equal names, and only one of them will be listed. The UNION command selects

only distinct values.

**SQL UNION ALL Example**

Now we want to list **all** employees in Norway and USA:

SELECT E\_Name FROM Employees\_Norway

UNION ALL

SELECT E\_Name FROM Employees\_USA

**Result**

