Foundation With SQL

# **Client:**A client is a device or system software that requests and consumes services or resources from another device or system software known as a server.

# **Server:**A server is a device or system software that provides services or resources to clients.

*Data and information are related but have distinct meanings.*

# **Data:**It is a raw and unprocessed fact and figures.

# **Information:**It is processed and meaningful output derived from the raw data.

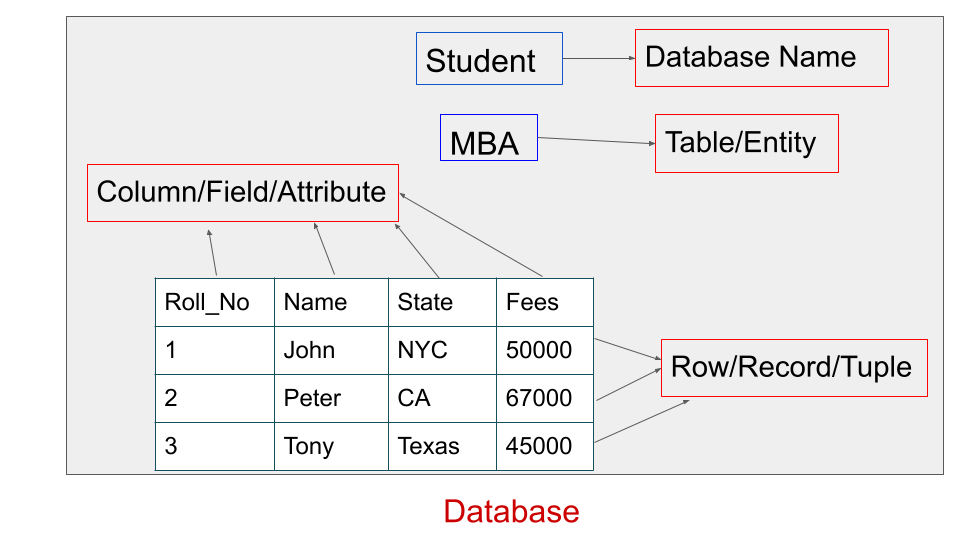
Example :

Data:

* Temperature: 25°C
* Humidity: 60%
* Wind Speed: 10 km/h
* Precipitation: 0.2 inches

Information: "Today's weather forecast: It will be a sunny and warm day with a temperature of 25°C, humidity at 60%, gentle wind blowing at 10 km/h, and a slight chance of 0.2 inches of precipitation."

# **Database:**



**Table/Entity:**It is a collection of data organized in row and column.

**Column/Field/Attribute:**Column represents the attributes or properties of data being stored. Each column has a name and datatype that defines what kind of data it can be stored. Eg. text, number, date etc

**Row/Records :**Row represents individual instances or records of data within the table.

**Cell:**Intersection point of row and column holding actual data values for a specific attribute of a particular record.

**Database Management System(DBMS):**DBMS is a system software that provides an interface and enables users to create, manage and interact with databases.eg dbase, Foxpro.

**Relational Database Management System(RDBMS):**It is used to deal with large amounts of data and relation between tables using primary key, foreign key and indexes.eg Microsoft SQL server ,MySQL Server, Oracle, MariaDB.



***Why should we learn SQL:***

# **SQL(Structure Query Language):**

**Structure:**Structure refers to the organization and arrangement of data within a database. It defines the layout and design of table,column, relationship, constraints and other components that make up database schema.

**Query:**It refers to a specific request or instruction to retrieve, manipulate or modify data stored in the database.

**Language:**Language refers to the way people communicate with each other in the same way users communicate with computers.

**SQL:**It is a programming language, which is used to interact with databases and perform operations on data within them.

SQL was first developed in the 1970s as a part of IBM system R project led by Dr.Edgar F codd and a language called SEQUEL(Structured English Query Language).

# ***CREATE DATABASE:***

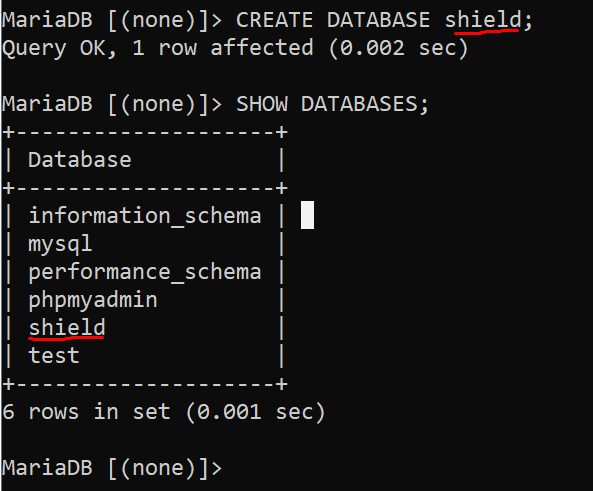
**Syntax:**

**CREATE DATABASE** database\_name;

Ex:**CREATE DATABASE** shield;

Rules:

* Only ‘\_’ we can use in database name
* Spaces, special characters, or SQL reserved keywords are not allowed.
* Name should be unique within the database management system.



**SHOW DATABASES;**

The SQL command "SHOW DATABASES" is used to **display a list of all the databases available on the MySQL server you are connected to**. It provides information about the existing databases that you have permission to access.

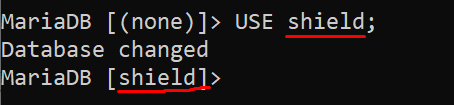
*Once we have created the database, we have to USE the database so that we can perform action on it.*

**USE statement** is used to specify which database should be the current or active database for the current session

**Syntax:**

USE database\_name;

Ex:USE shield;



# ***CREATE TABLE :***

**Syntax:**

CREATE TABLE table\_name (

column1 datatype1 [constraint1],

column2 datatype2 [constraint2],

...

columnN datatypeN [constraintN]

);

Rules:

* Only ‘\_’ we can use in table name
* Spaces, special characters, or SQL reserved keywords are not allowed.
* Column names should be unique within the table.

Ex:

*Creating a table without constraint:*

CREATE TABLE avengers(

ar\_id INT,

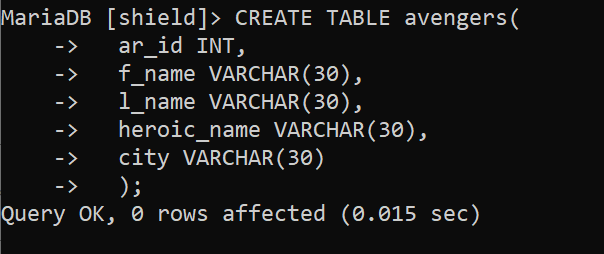
f\_name VARCHAR(30),

l\_name VARCHAR(30),

heroic\_name VARCHAR(30),

city VARCHAR(30)

);



*Creating a table with constraint:*

CREATE TABLE avengers(

ar\_id INT PRIMARY KEY AUTO\_INCREMENT,

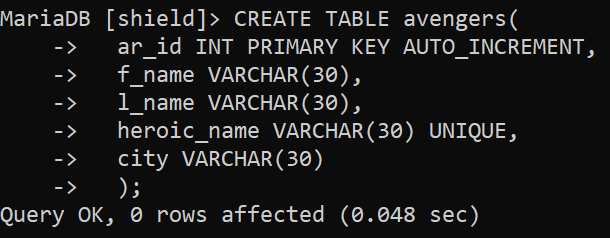
f\_name VARCHAR(30),

l\_name VARCHAR(30),

heroic\_name VARCHAR(30) UNIQUE,

city VARCHAR(30)

);



# ***DESCRIBE or DESC:***

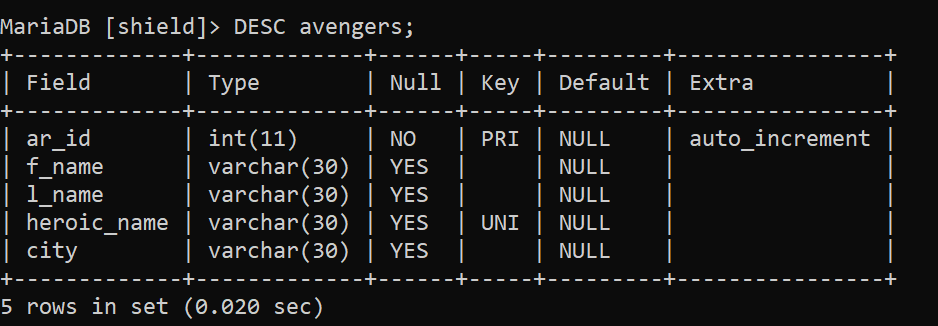
DESC or DESCRIBE is a keyword used to retrieve information about the columns or structure of a table. It doesn't show the data of the table.

**Syntax:**

DESCRIBE table\_name;

or

DESC table\_name;



# ***Datatypes in SQL:***

In SQL, different data types are used to represent various types of data that can be stored in a database table. The data types provide information about the type and size of the data that can be stored in each column of a table. The most common SQL data types include:

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

INT: Integer (whole numbers).

SMALLINT: Smaller integer.

BIGINT: Larger integer.

FLOAT: Floating Point number (approximate value).

DECIMAL or NUMERIC: Exact numeric value with a specified precision and scale.

## **2. Character Data Types:**

CHAR(n): Fixedlength character string of length 'n'.

VARCHAR(n): Variable Length character string with a maximum length of 'n'.

TEXT: Variable Length character string for large amounts of text data.

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

DATE: Stores date values (year, month, day).

TIME: Stores time values (hour, minute, second).

DATETIME or TIMESTAMP: Stores both date and time values.

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

BOOLEAN or BOOL: Represents true or false values.

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

BINARY(n): Fixed Length binary string of length 'n'.

VARBINARY(n): Variable Length binary string with a maximum length of 'n'.

BLOB: Binary Large Object for storing large binary data like images, videos, etc.

Each database system may have some variations in naming and implementation of data types, but the general categories remain consistent across most SQL databases. It's essential to choose the appropriate data type based on the nature and size of the data you want to store to ensure efficient storage and retrieval and to maintain data integrity.

# ***INSERT INTO:***

The **INSERT INTO** statement in SQL is used to insert new records or rows into a table. It allows you to add data to the table by specifying the column names and the corresponding values that you want to insert.

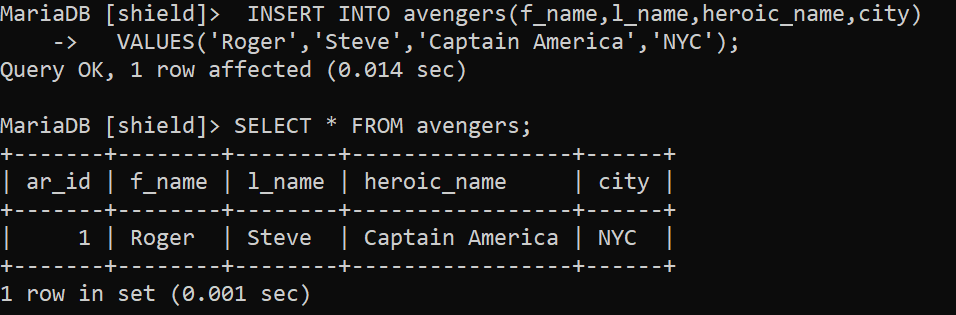
**Syntax:**

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

Rules:

* Order of both columns and values should be the same.
* Values must be declared in single quotes (‘values‘). For numeric datatypes we can declare values without single quotes.



## ***1. Inserting Records in multiple row:***

**Syntax:**

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES

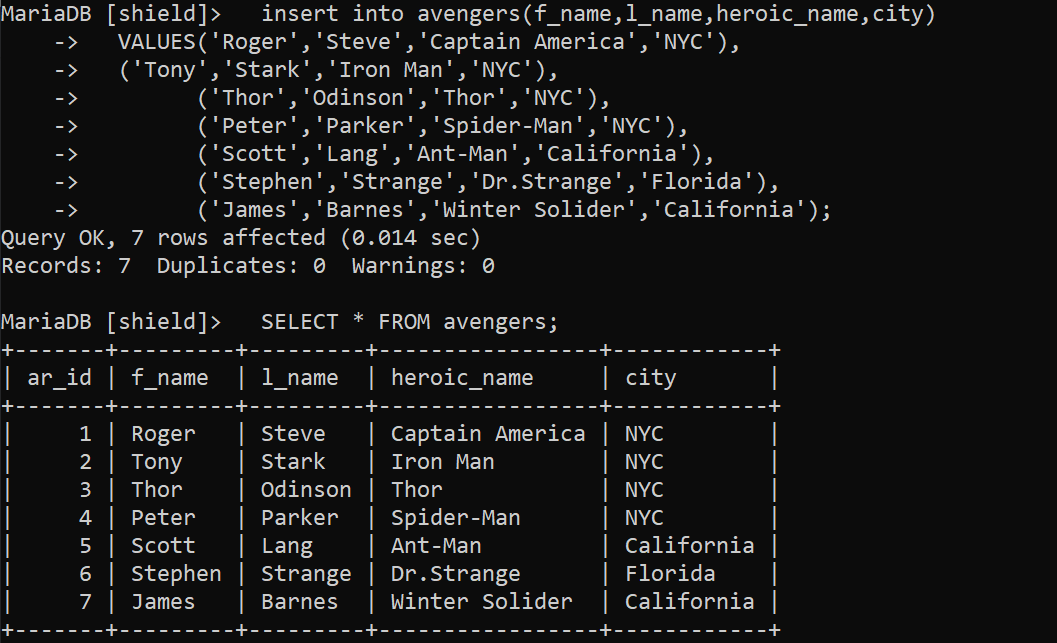
(value1\_row1, value2\_row1, value3\_row1, ...),

(value1\_row2, value2\_row2, value3\_row2, ...),

(value1\_row3, value2\_row3, value3\_row3, ...),

…

(value n\_row n, value n\_row n, value n\_row n, ...);



## ***2. Inserting Records Changing order of column :***

**Syntax:**

INSERT INTO table\_name (column3, column1, column2, ...)

VALUES

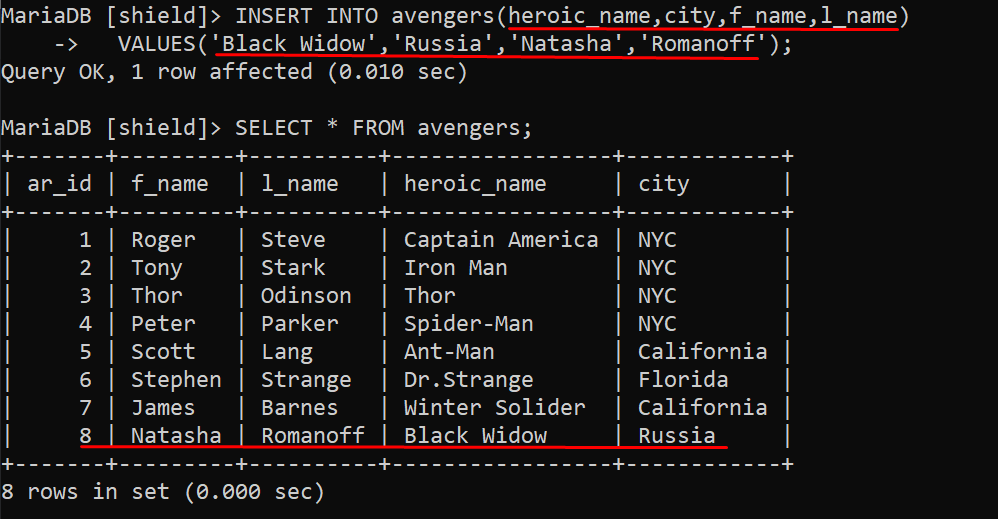
(value3\_row1, value1\_row1, value2\_row1, ...),

(value3\_row2, value1\_row2, value2\_row2, ...),

(value3\_row3, value1\_row3, value2\_row3, ...),

…

(value n\_row n, value n\_row n, value n\_row n, ...);



## ***3. Inserting Records In Specific Column :***

**Syntax:**

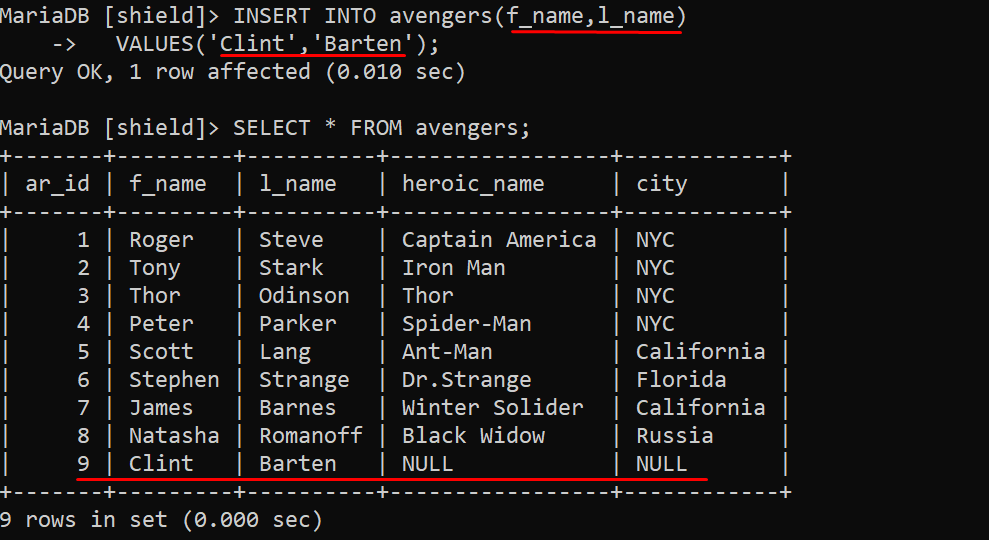
INSERT INTO table\_name (column1, column2, column3, ...)

VALUES

(value1\_row1, value2\_row1, value3\_row1, ...);

Rules:

* Order of both columns and values should be the same.
* Constraint should be null of the column of which we are not inserting values.
* By default all the values will be null for all other columns of which we are not inserting values.



## ***4. Inserting Records Without Specifying Column :***

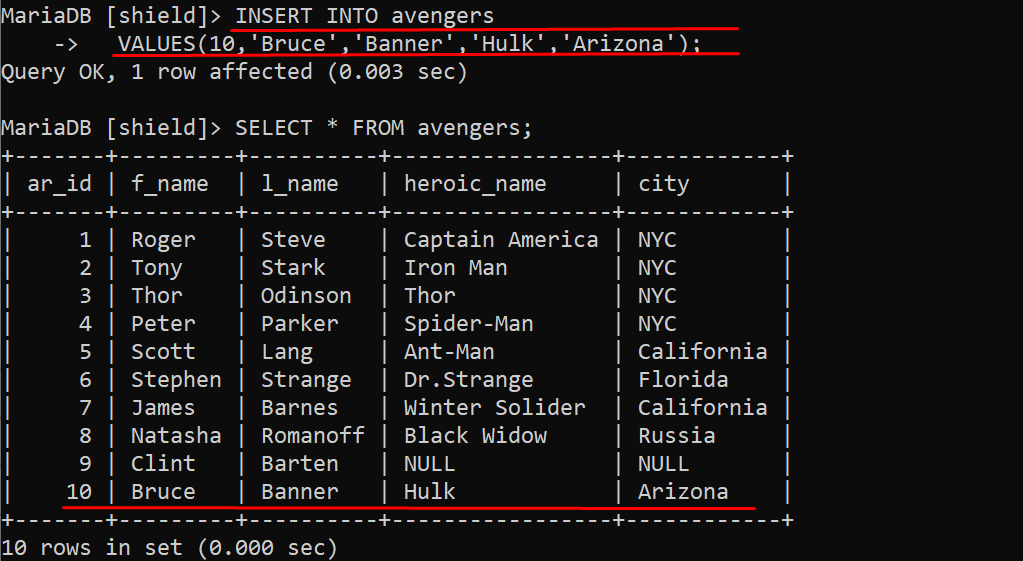
**Syntax:**

INSERT INTO table\_name

VALUES (value1\_row1, value2\_row1, value3\_row1, ...);

Rules:

* Order of both columns and values should be the same.
* All column values must be entered.



## ***5.Insert a single quote:***

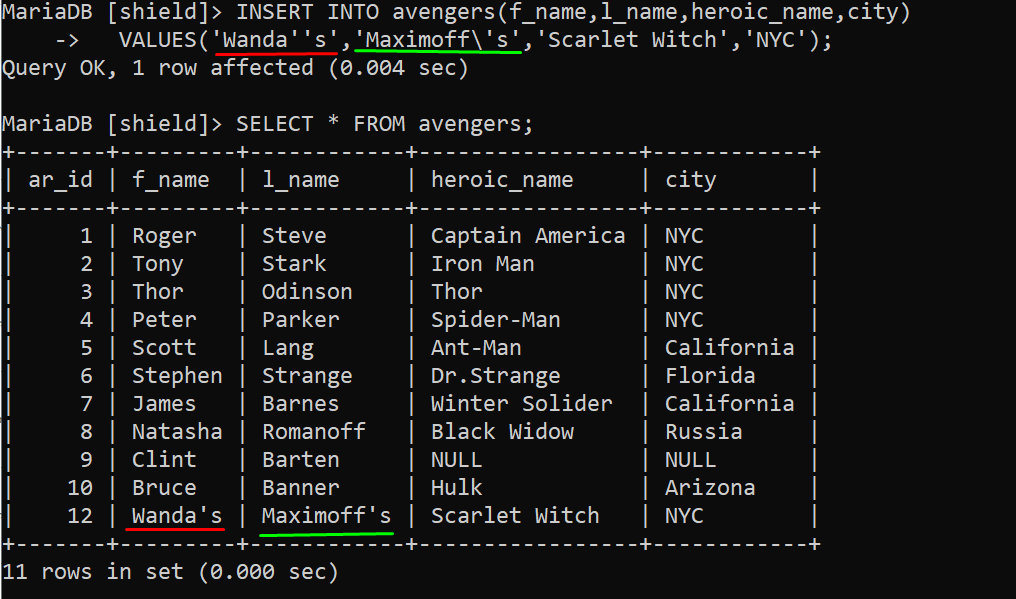
We can insert single quote into two ways:

* **1.Using two single quotes (''):**
* INSERT INTO table\_name (column1, column2)
* VALUES ('Wanda’’s', 'Maximoff’’s');

**2.Using the backslash character:**

INSERT INTO table\_name (column1,column2)

VALUES ('Wanda\’s', 'Maximoff\’s');



# ***SELECT :***

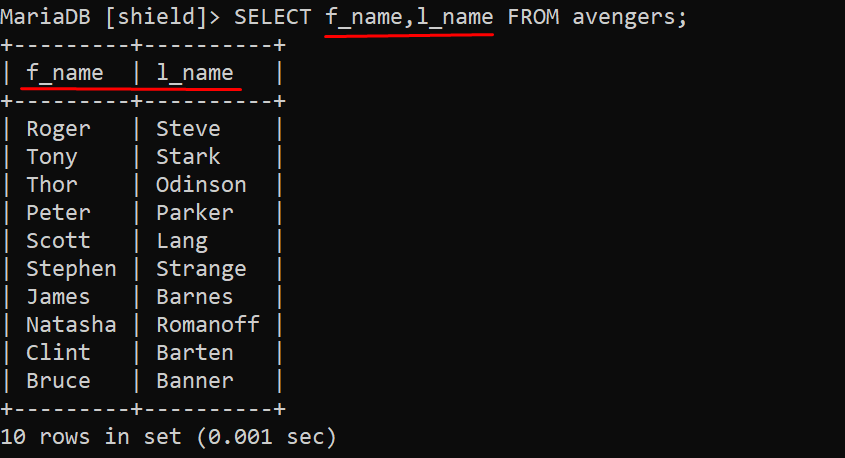
The **SELECT** statement in SQL is used to retrieve data from one or more tables in a database. It is one of the fundamental and most frequently used SQL commands.

**Syntax:**

SELECT column1, column2, column3, ...

FROM table\_name;

You can also use other clauses with the **SELECT** statement to further refine the results, such as:

* WHERE clause: To filter rows based on specified conditions.
* ORDER BY clause: To sort the results in ascending or descending order based on one or more columns.
* GROUP BY and HAVING clauses: To group data and apply aggregate functions.
* LIMIT (MySQL) or TOP (SQL Server) clause: To limit the number of rows returned.

**Display all the data on the table.**



## **LIMIT:**

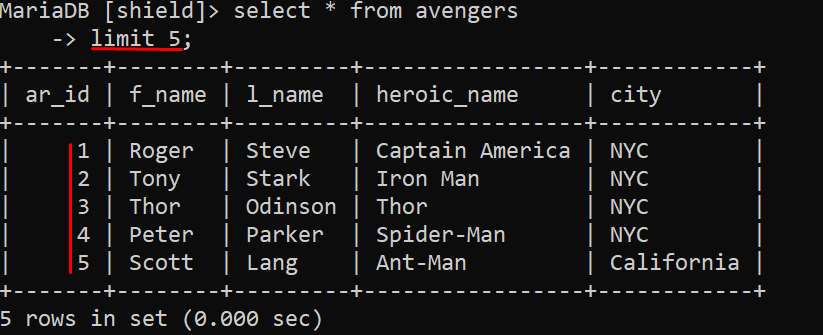
LIMIT clause is used to restrict the number of rows returned by a query. It is commonly used with the SELECT statement to limit the result set to a specific number of rows.

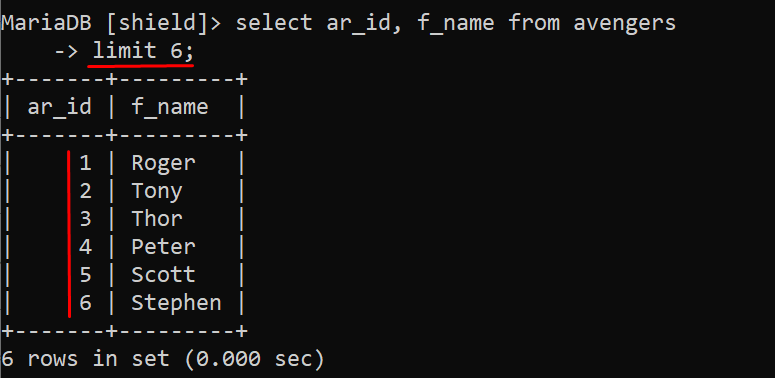
**Syntax:**

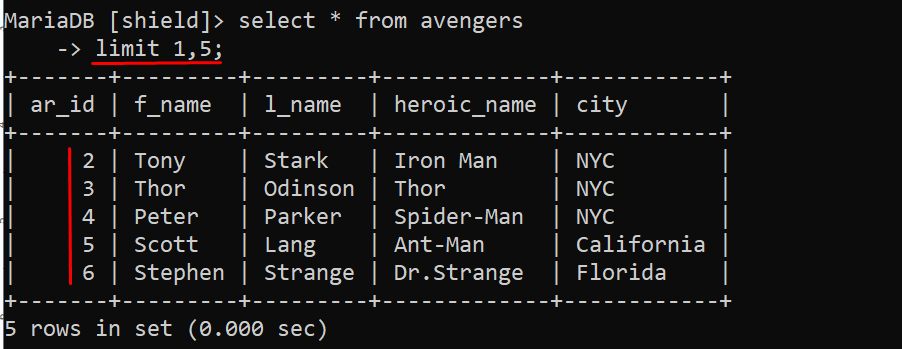
SELECT column1, column2, ...

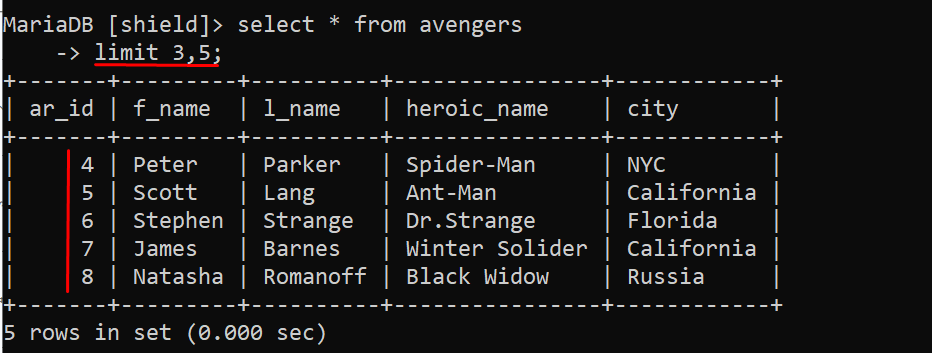
FROM table\_name

LIMIT number\_of\_rows;









# 

# 

# 

# 

# ***Operator:***

In SQL (Structured Query Language), **operators are used to perform operations on data in the database.**

**Assignment Operator (=):** Assigns a value to a variable.

## ***Arithmetic Operators:***

* **Addition (+):** Adds two values.
* **Subtraction (-):** Subtracts the second value from the first.
* **Multiplication ():** Multiplies two values.
* **Division (/):** Divides the first value by the second.
* **Modulus (%):** Returns the remainder of the division.

## ***Logical Operators:***

* **AND:** Returns true if both conditions are true.
* **OR:** Returns true if at least one condition is true.
* **NOT:** Reverses the result of a logical expression.

## ***Comparison Operators:***

* **Equal (==):** Tests if two values are equal.
* **Not equal (!= or <>):** Tests if two values are not equal.
* **Greater than (>):** Tests if the first value is greater than the second.
* **Less than (<):** Tests if the first value is less than the second.
* **Greater than or equal to (>=):** Tests if the first value is greater than or equal to the second.
* **Less than or equal to (<=):** Tests if the first value is less than or equal to the second.

**Operator:**

* **IS NULL:** Tests if a value is NULL.
* **IS NOT NULL:** Tests if a value is not NULL.
* **IN:** Tests if a value matches any value in a list.
* **BETWEEN:** Tests if a value is within a range.
* **LIKE:** Used in pattern matching with wildcard characters.

# 

# 

# 

# ***Where Clause:***

In SQL, the WHERE clause is used to filter rows from a table based on a specified condition.

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

* **SELECT:** Specifies the columns you want to retrieve in the result set.
* **FROM:** Specifies the table from which to retrieve data.
* **WHERE:** Specifies the condition that each row must satisfy to be included in the result set.
* **condition:** Specifies the condition that must be met. It can be a comparison, logical expression, or combination of conditions.

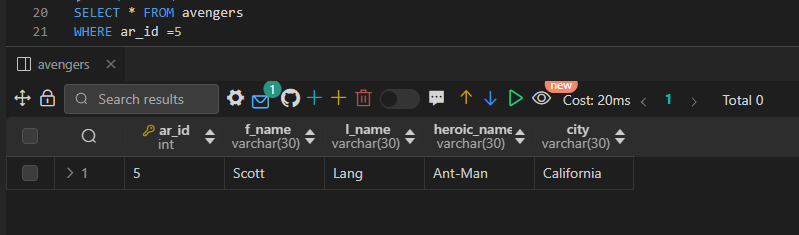
## ***1. Equal Operator:***

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name = value;**

****

## 

## 

## ***2. Not Equal Operator:***

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

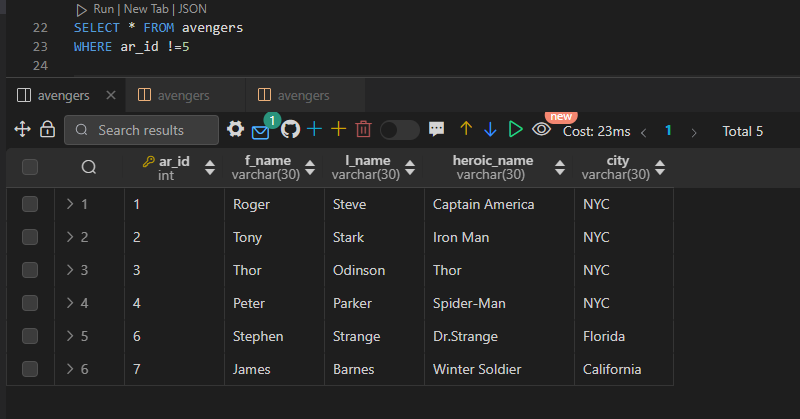
**WHERE column\_name != value;**

**====================================**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name <> value;**

****

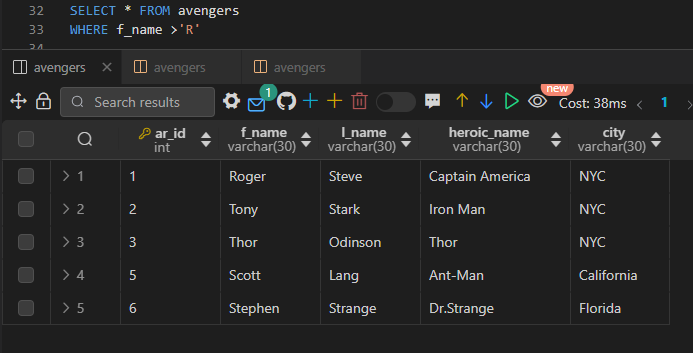
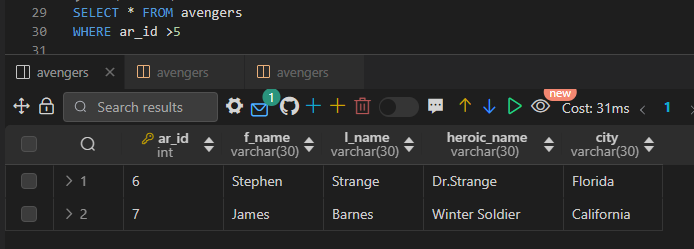
## ***3. Greater Than Operator:***

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name > value;**

****

* **In case of number It does not include the current number eg. ar\_id>5.**
* **5 will not be included in the result.**
* **In case of string It include the current string eg. f\_name >’R’.**
* **Names starting with ‘R’ will be included in the result.**

## 

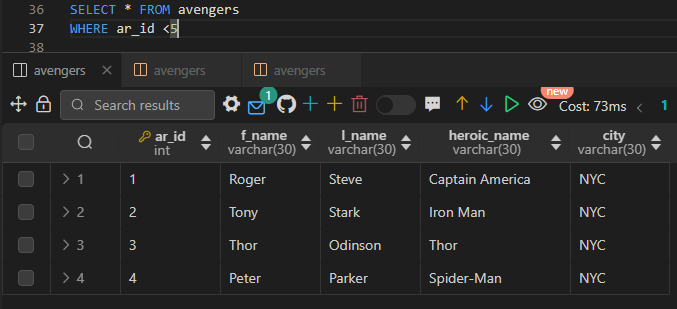
## ***4. Less Than Operator:***

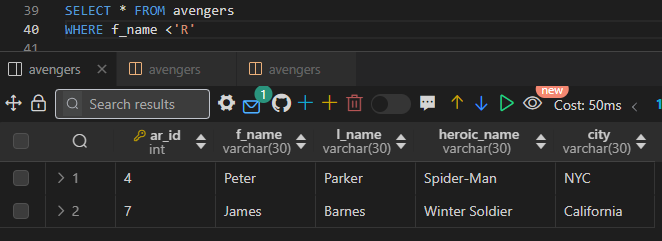
**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name < value;**

****

****

* **In case of number It does not include the current number eg. ar\_id<5.**
* **5 will not be included in the result.**
* **In case of less than string does not include the current string eg. f\_name <’R’.**
* **Names starting with ‘R’ will not be included in the result.**

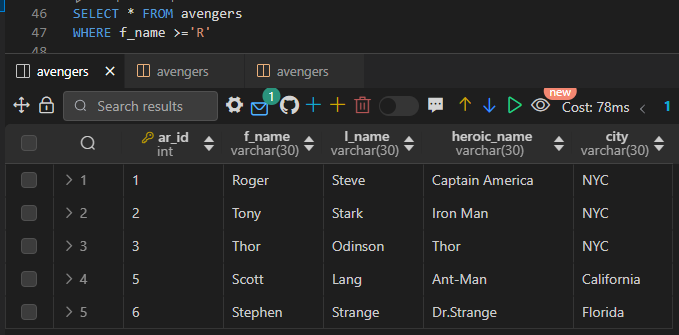
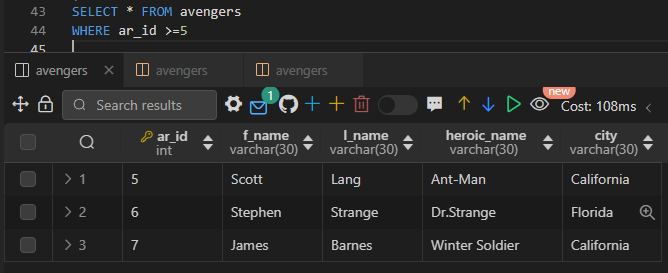
## ***5. Greater than or equal to (>=) Operator:***

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name >= value;**

****

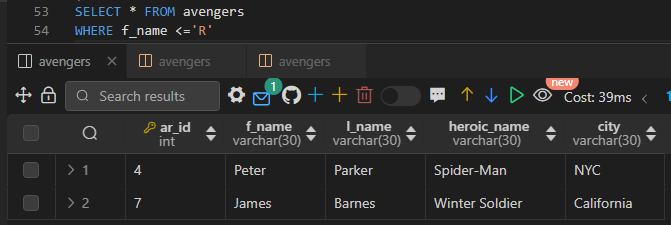
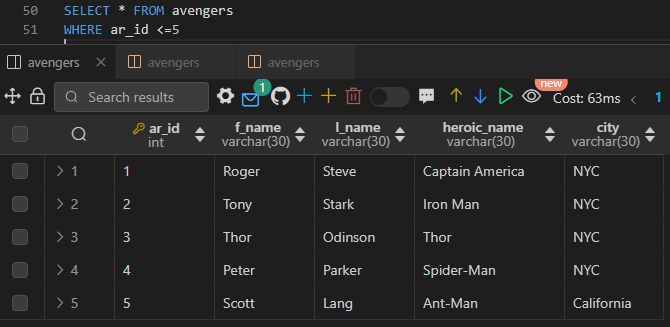
* **In case of number and string both the element will be included in the result.**

## ***6. Less than or equal to (<=) Operator:***

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name <= value;**

* **In case of number It include the current number eg. ar\_id<=5.**
* **5 will be included in the result.**
* **In case of less than string does not include the current string eg. f\_name <’R’.**
* **Names starting with ‘R’ will not be included in the result.**

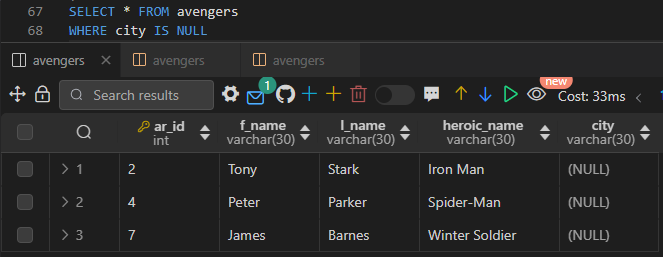
## **7. IS NULL:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name IS NULL**;



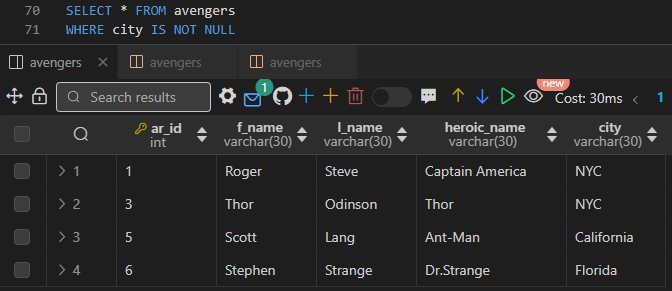
## **8. IS NOT NULL:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name IS NOT NULL**;



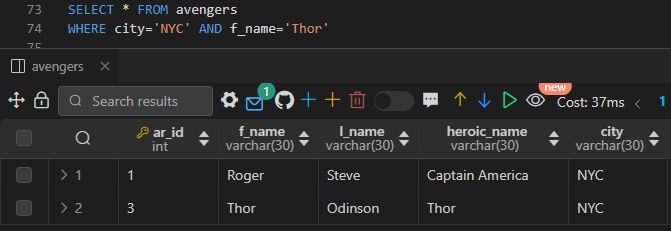
## **9. AND Operator:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition1 AND condition2 AND ...;**

****

## 

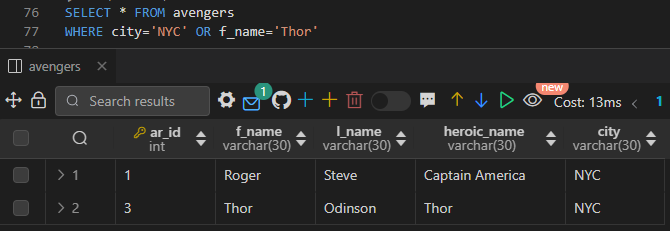
## **10. OR Operator:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition1 OR condition2 OR ...;**

****

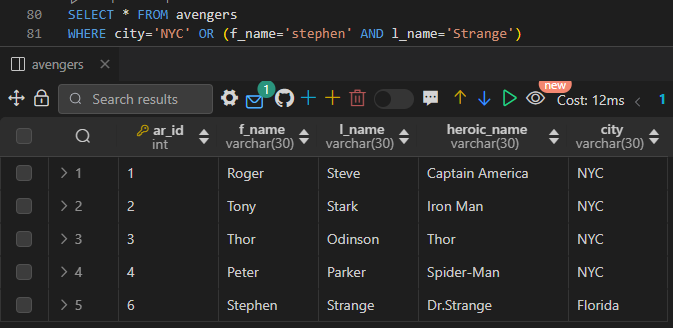
## **11. Combination AND and OR Operator:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE (condition1 AND condition2) OR (condition3 AND condition4) OR condition5 ...;**

****

## **12. IN Operator:**

* IN operator allows you to specify multiple values in a WHERE clause.
* It is used to filter results based on a specific list of values.

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name IN (value1, value2, ...);**

# 

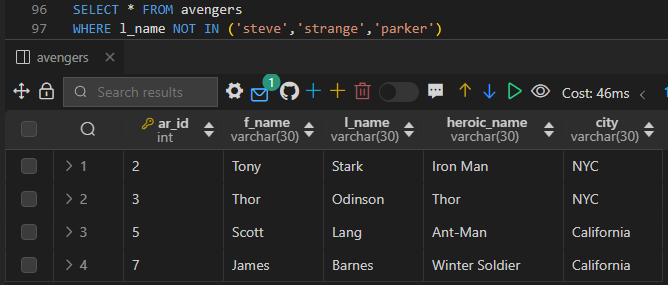
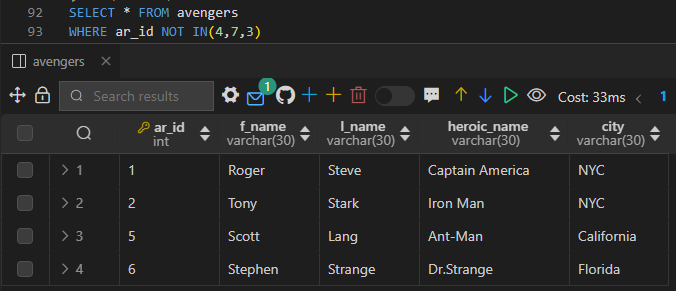
## **13. NOT IN Operator:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name NOT IN (value1, value2, ...);**

****

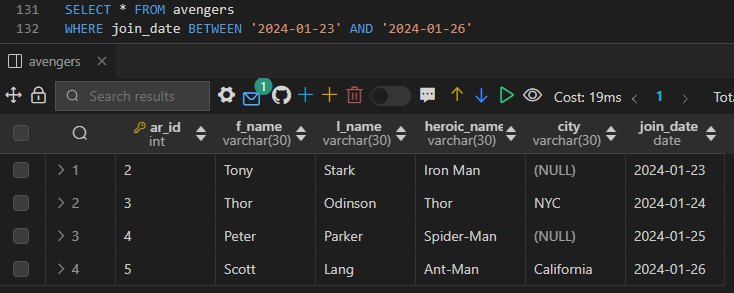
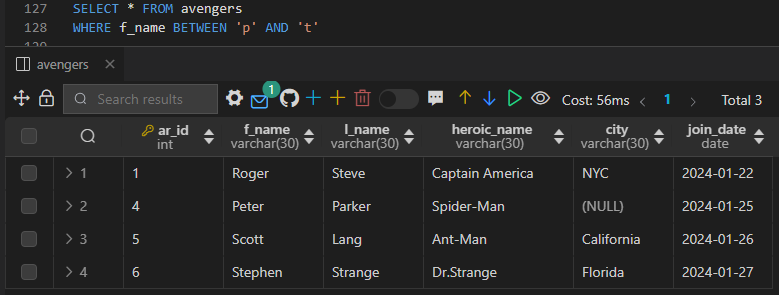
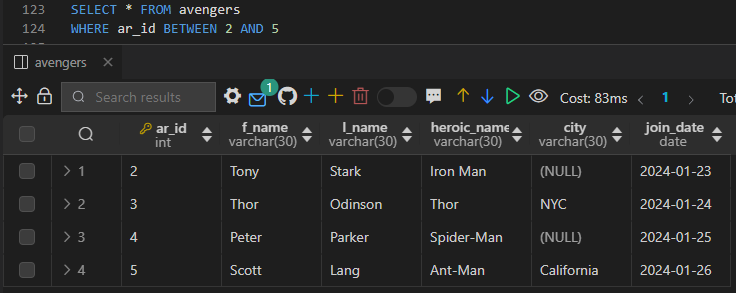
## 

## **14. BETWEEN Operator:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name BETWEEN value1 AND value2;**

* **For Number and Date both the Starting and End range will be included in the result.**
* **For String, Only Starting Range will be included , End range will be excluded from the result.**

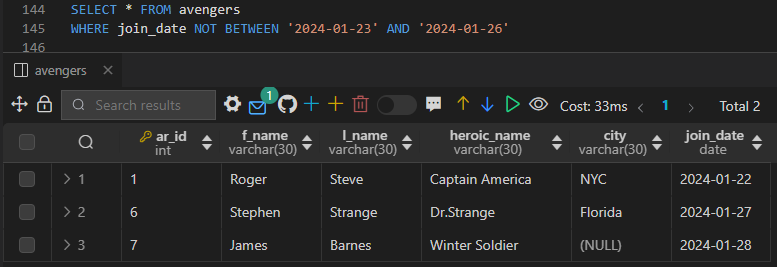
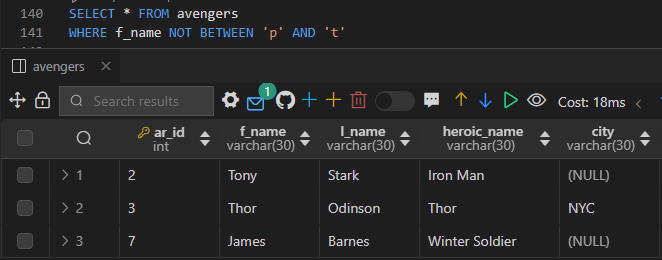
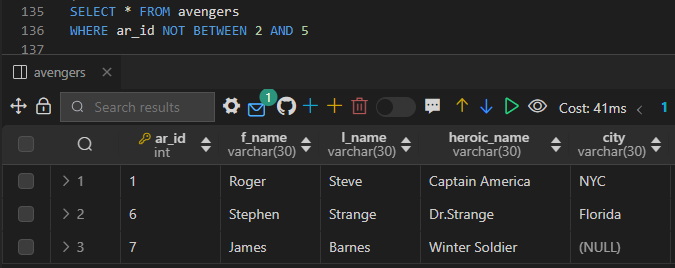
## **15. NOT BETWEEN Operator:**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name BETWEEN value1 AND value2;**

****

* **For Number and Date both the Starting and End range will not be included in the result.**
* **For String, Only Starting Range will not be included , End range will be included in the result.**

## **16. Like Operator:**

* The LIKE operator in SQL is used to search for a specified pattern in a column.
* It's commonly used with **wildcard characters** to match patterns rather than exact values.

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column\_name LIKE pattern;**

**Wildcard characters:**

* **%** represents zero, one, or multiple characters.
* **\_** represents a single character.

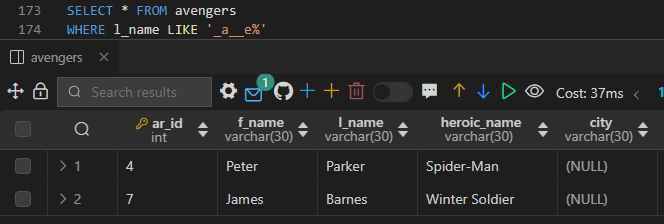
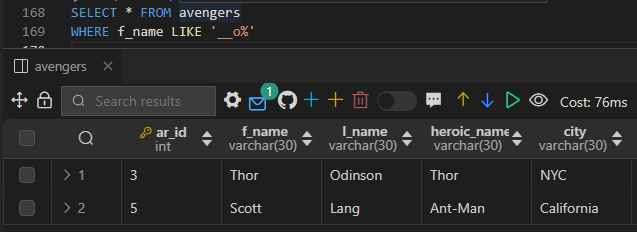
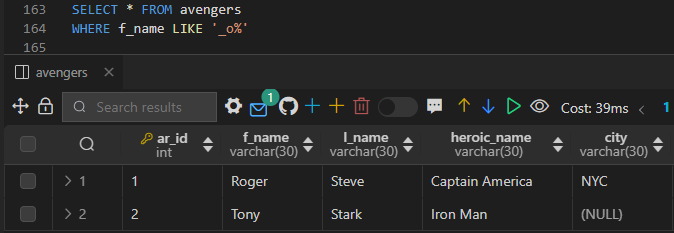
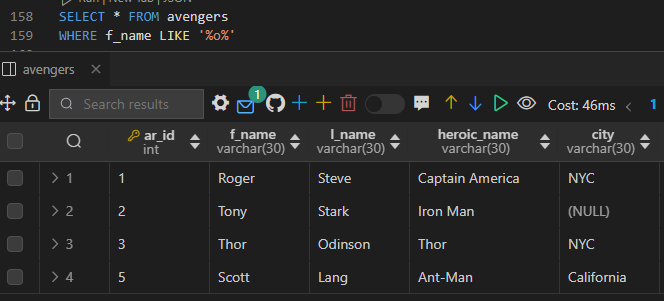
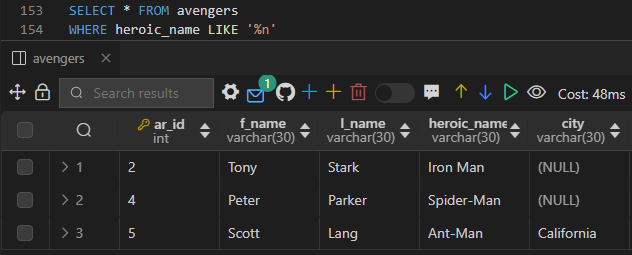
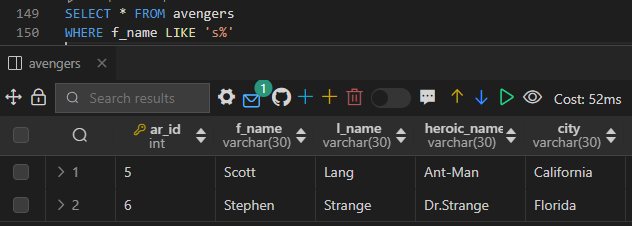
**LIKE 'a%'** matches any value that starts with 'a'.

**LIKE '%a'** matches any value that ends with 'a'.

**LIKE '%or%'** matches any value that contains 'or' anywhere in the string.

**LIKE '\_r%'** matches any value that has 'r' as the second character.

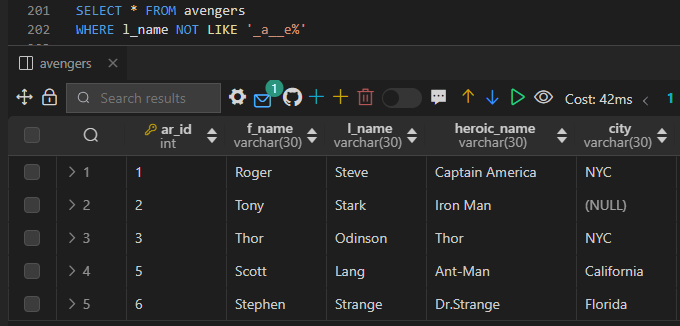
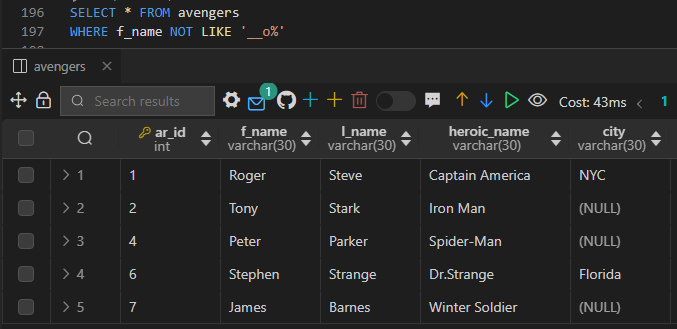
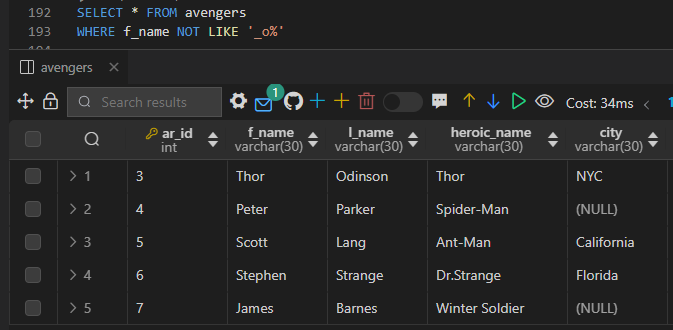
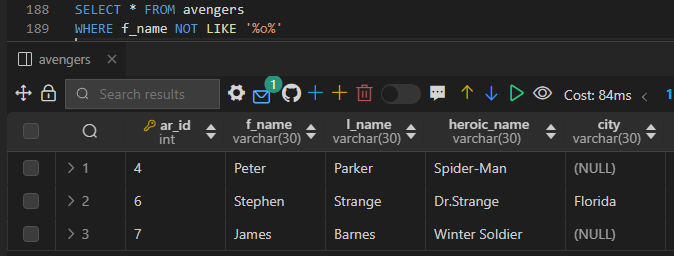
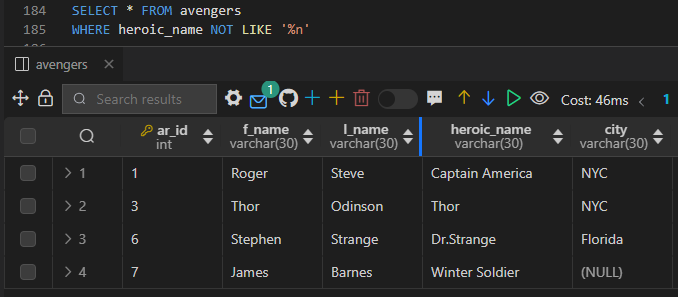
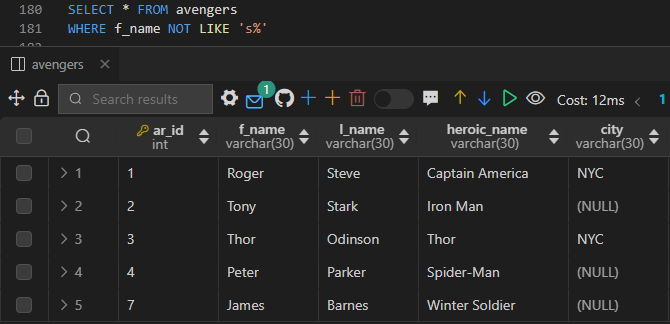
**The LIKE operator is often used in combination with % and \_ wildcard characters to perform pattern matching in SQL queries.**

****

## 

## **17. NOT LIKE Operator:**

* The NOT LIKE operator in SQL is used to **retrieve rows where a specified pattern does not match.**
* It **works similarly to the LIKE operator but negates the matching condition.**

****

# 

# 

# ***ORDER BY:***

* The ORDER BY clause in SQL is used to **sort the result set of a query based on one or more columns.**
* **It sorts the rows returned by a SELECT statement in ascending or descending order, depending on the specified criteria.**

**Syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**ORDER BY column1 [ASC|DESC], ...;**

**+**

# ***Constraint:***

Constraint **is a rule or condition that is applied to a table column or a set of columns** to maintain the integrity, accuracy, and consistency of the data stored in the database.

## **1.Not Null Constraint:**

Ensures that a column **cannot have a NULL value**. It requires that every row in the table has a value for that column

## **2.Unique Constraint:**

Ensures that **all values in a column (or a set of columns) are unique**. Multiple unique constraints can be defined per table, and **they allow NULL values** (except for columns defined as NOT NULL).

## **3.Default Constraint:**

Default constraints are useful for ensuring that a **column always contains a value**, even if it's not explicitly provided during an INSERT operation.

## **4.Check Constraint:**

Defines a condition that **each row in the table must satisfy**. It restricts the range of values that a column can contain.

## **5.Primary Key Constraint:**

* Ensures that **each record in a table is uniquely identified.**
* It **prevents duplicate and null values** in the specified column(s).
* **Only one primary key constraint can be defined per table.**
* **Primary key is a combination of NOT NULL and UNIQUE constraints.**

## **6.Foreign Key Constraint:**

Defines a **relationship between two tables.**

It ensures **referential integrity** by enforcing that **values in a column (or a set of columns) of one table match the values in another table's column** (usually the primary key of the referenced table).

# ***AUTO\_INCREMENT:***

* Auto-increment is a feature in SQL used to **automatically generate unique, sequential values for a column typically used as a primary key.**
* It simplifies the process of **inserting new records** into a table by automatically assigning a value to the column **without requiring manual input.**
* Other **databases might have similar functionality but with different syntax**. For example, **in SQL Server, you can use IDENTITY** property to achieve the same functionality.

# **Distinct:**

**DISTINCT** is useful when you want to **identify unique values** within a specific column or combination of columns in your query result.

**Syntax:**

**SELECT DISTINCT column**

**FROM table\_name;**

# ***ALTER TABLE:***

**ALTER TABLE** statement is used to modify an existing table's structure, such as:

* Add Column
* Change Column
* Modify Column
* Drop Column
* Disable Constraints

## ***ADD Column:***

### ***1.Adding columns without constraints.***

**Syntax:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size);**

### ***2.Adding multiple columns without constraints.***

**Syntax:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size),**

**ADD COLUMN column\_name data\_type(size);**

### ***3.Adding column to specific position without constraint.***

Use ALTER TABLE statement along with the ADD COLUMN clause, specifying the desired position using the **FIRST or AFTER keywords**.

By default column is added to the last.

**Syntax for FIRST keyword:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size) FIRST**

***The FIRST keyword will add a column to the starting of the table.***

**Syntax for AFTER keyword:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size) AFTER column\_name**

***The AFTER keyword will add a column to the starting of the table.***

### ***4.Adding columns with constraints.***

**Syntax:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size) [column\_constraints];**

### ***5.Adding multiple columns with constraints.***

**Syntax:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size) [column\_constraints],**

**ADD COLUMN column\_name data\_type(size) [column\_constraints];**

### ***6.Adding column to specific position with constraint.***

**Syntax for FIRST keyword:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size) [constraint\_name] FIRST**

***The FIRST keyword will add a column to the starting of the table.***

**Syntax for AFTER keyword:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name data\_type(size) [constraint\_name] AFTER column\_name**

***The AFTER keyword will add a column to the starting of the table.***

## ***CHANGE Column:***

### ***1.Changing column Name and Datatype without constraint.***

**Syntax:**

**ALTER TABLE table\_name**

**CHANGE COLUMN old\_column\_name new\_column\_name data\_type(size);**

If we change columns without datatype it will give an error.

### ***2.Changing column Name and Datatype of more than one column without constraint.***

**Syntax:**

**ALTER TABLE table\_name**

**CHANGE COLUMN old\_column\_name new\_column\_name data\_type(size),**

**CHANGE COLUMN old\_column\_name new\_column\_name data\_type(size);**

If we change columns without datatype it will give an error.

### ***3.Changing column Name and Datatype with constraint.***

**Syntax:**

**ALTER TABLE table\_name**

**CHANGE COLUMN old\_column\_name new\_column\_name data\_type(size) constraint\_name;**

## ***Modify Column:***

It allows you to alter properties of the column such as its data type, length, and constraints without changing the name of the column.

**Syntax:**

**ALTER TABLE table\_name**

**MODIFY COLUMN column\_name data\_type(size) [column\_constraints];**

## **Drop Column:**

To drop a column from a table in SQL, you **use the ALTER TABLE statement with the DROP COLUMN clause**.

**Syntax:**

**ALTER TABLE table\_name**

**DROP COLUMN column\_name;**

## ***Syntax for finding Constraint Name:***

**SELECT \* FROM INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS WHERE TABLE\_NAME = 'your\_table\_name';**

## ***Disable constraint:***

### **1. NOT NULL :-**

We can simply remove the NOT NULL constraint **by using MODIFY and CHANGE COLUMN clauses.**

**Syntax:**

**ALTER TABLE table\_name**

**MODIFY COLUMN column\_name Datatype(size)**

### **2. DEFAULT:**

**Syntax:**

**ALTER TABLE table\_name**

**ALTER COLUMN column\_name DROP DEFAULT;**

### **3. UNIQUE**:

We can remove UNIQUE constraint in two ways First through Index and second Constraint name

**SHOW INDEX FROM table\_name;**

**Syntax 1:**

**ALTER TABLE table\_name**

**DROP INDEX key\_name;**

**Syntax 2:**

**ALTER TABLE table\_name**

**DROP CONSTRAINT constraint\_name;**

### **4. Primary key:**

**ALTER TABLE table\_name**

**DROP PRIMARY KEY,**

**MODIFY COLUMN column\_name Datatype(size) ;**

**Syntax for finding Constraint Name:**

**SELECT FROM INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS WHERE TABLE\_NAME = 'your\_table\_name';**

**SHOW CREATE TABLE avengers; :-** This command will result in the query that has been used while creating a table.

# ***UPDATE:***

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

**Syntax:**

**UPDATE table\_name**

**SET column1 = value1, column2 = value2, ...**

**[WHERE condition];.**

* **table\_name:** The name of the table you want to update.
* **column1, column2, ...:** The columns you want to update.
* **value1, value2, ...:** The new values you want to set for the specified columns.
* **WHERE condition:** An optional clause that specifies which rows to update. **If omitted, all rows in the table will be updated.**

## **Update Case:**

We use CASE statements within an UPDATE statement in SQL to conditionally update values based on specific conditions.The case statements apply only on one column.

**Syntax:**

**UPDATE table\_name**

**SET column\_name =**

**CASE**

**WHEN condition1 THEN value1**

**WHEN condition2 THEN value2**

**...**

**ELSE default\_value**

**END**

**[WHERE condition];**

**WHERE condition:** **An optional clause** that specifies which rows to update. If omitted, all rows in the table will be updated.

## **Delete:**

The DELETE statement in SQL is used to **remove rows from a table based on specified conditions.**

**Syntax:**

**DELETE FROM table\_name**

**[WHERE condition];**

* **table\_name:** The name of the table from which you want to delete rows.
* **WHERE condition:** An optional clause that specifies which rows to delete. **If omitted, all rows in the table will be deleted.**

## **Truncate:**

The TRUNCATE TABLE statement in SQL is **used to quickly delete all rows from a table.**

Unlike the DELETE statement, which removes rows one by one and generates individual rollback logs for each deleted row, **TRUNCATE TABLE is faster because it removes all rows at once** and generates minimal transaction logs, making it more efficient for large tables.

Syntax:

**TRUNCATE TABLE table\_name;**

## ***Difference between DELETE and TRUNCATE:***

**DELETE:**

* It is used to remove specific rows from a table based on a specified condition.
* DELETE is a logged operation, meaning it is recorded in the transaction log and can be rolled back if necessary.
* It can be slower when dealing with a large number of rows.
* DELETE **does not reset identity or auto-increment columns.**

**TRUNCATE:**

* TRUNCATE command is used to remove all rows from a table in a more efficient way compared to DELETE **without the need for a WHERE condition.**
* TRUNCATE is not as logged as DELETE, **means data cannot be rolled back.**
* TRUNCATE **resets identity or auto-increment columns, effectively starting from the initial value.**

# ***Alias:***

An alias is a **temporary name assigned to a table or a column in a query to make the output more readable or to provide a shorter reference to the table or column**.

**Table Syntax:**

**SELECT alias.column1, alias.column2**

**FROM table\_name AS alias;**

**Column syntax:**

**SELECT column\_name AS alias\_name**

**FROM table\_name;**

**Aggregate Function Syntax:**

**SELECT AGG\_FUNC(column) AS alias\_name**

**FROM table\_name;**

# 

# ***Functions:***

## **String Function:**

String functions in SQL are used to manipulate character data, such as extracting substrings, converting case, concatenating strings, trimming whitespace, and more.

### **1. CONCAT:** Concatenates two or more strings together.

**SELECT CONCAT(first\_name, ' ', last\_name) AS full\_name**

**FROM employees;**

### **2. UPPER:** Converts a string to uppercase.

**SELECT UPPER(cloumn\_name) AS uppercase\_string FROM table\_name;**

### **3. LOWER:** Converts a string to lowercase.

**SELECT LOWER(‘Hello’) AS lowercase\_string;**

### **4. SUBSTRING / SUBSTR:** Extracts a substring from a string.

**SELECT SUBSTRING('abcdef', 2, 3) AS substring\_result;**

### **5.REPLACE:** Replaces occurrences of a substring within a string with another substring.

**SELECT REPLACE('hello world', 'world', 'SQL') AS replaced\_string;**

## **Math Function:**

Math functions in SQL are used to perform mathematical operations on numeric data.

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

**SELECT ABS(-10) AS absolute\_value;**

### **2. CEILING / CEIL:** Returns the smallest integer greater than or equal to a number.

**SELECT CEILING(4.3) AS ceiling\_value;**

### **3. FLOOR:** Returns the largest integer less than or equal to a number.

**SELECT FLOOR(4.9) AS floor\_value;**

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

**SELECT ROUND(4.567, 2) AS rounded\_value;**

### **5. POWER:** Raises a number to the power of another number.

**SELECT POWER(2, 3) AS power\_value;**

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

**SELECT SQRT(25) AS square\_root;**

## **Date Function:**

### **1. CURDATE():** Returns the current date.

**SELECT CURDATE()**

### **2.NOW():** Returns the current date and time at which the statement was executed.

**SELECT NOW()**

### **3.SYSDATE():** Returns the current date.

**SELECT SYSDATE()**

### **4.Last\_day(date):** returns the last day of the corresponding month for a date or datetime value

**SELECT LAST\_DAY('2024-02-12')**

### **5.DAYNAME:** Returns the Name of Day.

**SELECT DAYNAME('1947-08-15') AS day\_of\_week;**

### **6. MONTHNAME:** Returns Name of the month.

**SELECT MONTHNAME('1990-09-25') AS month\_name;**

### **7. YEAR:** Returns name of the year

**SELECT YEAR("2017-06-15") AS year;**

# **Aggregate function:**

Aggregate functions in SQL are used to **perform calculations on sets of values and return a single value as a result.**

These functions are **commonly used in conjunction with the GROUP BY clause** to calculate summary statistics for groups of rows in a table.

## **1. COUNT:** Returns the number of rows in a group.

SELECT COUNT() AS total\_rows FROM employees;

## **2. SUM:** Calculates the sum of values in a group.

**SELECT SUM(salary) AS total\_salary FROM employees;**

## **3. AVG:** Calculates the average of values in a group.

**SELECT AVG(salary) AS average\_salary FROM employees;**

## **4. MIN:** Returns the minimum value in a group.

**SELECT MIN(salary) AS min\_salary FROM employees;**

## **5. MAX:** Returns the maximum value in a group.

**SELECT MAX(salary) AS max\_salary FROM employees;**

# **GROUP BY clause:**

The GROUP BY clause is **used in conjunction with aggregate functions (such as SUM, AVG, COUNT, MAX, MIN) to perform calculations on groups of rows rather than individual rows.**

It organizes the result set into summary rows based on the specified grouping columns.

**Syntax:**

**SELECT column1, aggregate\_function(column2)**

**FROM table\_name**

**GROUP BY column1;**

**column1:** The column or expression by which you want to group the results.

**aggregate\_function(column2):** An aggregate function applied to a column.

**table\_name:** The name of the table you're querying.

# **HAVING Clause:**

* The HAVING clause in SQL is **used in conjunction with the GROUP BY clause to filter the results of aggregate functions applied to grouped rows**.
* It allows you to **apply a condition to groups of rows after they have been aggregated**.
* The **HAVING clause is similar to the WHERE clause, but it operates on groups defined by the GROUP BY clause rather than individual rows.**
* HAVING clause **can only be used in queries that include a GROUP BY clause.**

**Syntax:**

**SELECT column1, aggregate\_function(column2)**

**FROM table\_name**

**GROUP BY column1**

**HAVING condition;**

**- column1:** The column or expression by which you want to group the results.

**- aggregate\_function(column2):** An aggregate function applied to a column.

**- table\_name:** The name of the table you're querying.

**- condition:** The condition that must be satisfied by the aggregated values.

# **Foreign key:**

* A foreign key is a column or set of columns in one table that refers to the primary key in another table.
* This creates a link between the two tables.
* Values in foreign key columns can be NULL.
* It establishes a relationship between the two tables, known as a parent-child relationship.

***Referential Integrity:*** Foreign keys **enforce referential integrity,** which **ensures that relationships between tables remain valid**.

## ***Create Table Syntax:***

**Syntax:**

***1.With Constraint Name:***

**CREATE TABLE child\_table (**

**child\_id INT PRIMARY KEY,**

**parent\_id INT,**

**CONSTRAINT constraint\_name**

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id)**

**ON DELETE CASCADE**

**ON UPDATE CASCADE**

**);**

***2.Without Constraint Name:***

**CREATE TABLE child\_table (**

**child\_id INT PRIMARY KEY,**

**parent\_id INT,**

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id)**

**ON DELETE CASCADE**

**ON UPDATE CASCADE**

**);**

## ***Drop Foreign key:***

**Syntax 1:**

**ALTER Table table\_name**

**DROP constraint constraint\_name;**

**Syntax 2:**

**ALTER Table table\_name**

**DROP FOREIGN KEY constraint\_name;**

## ***Add foreign key to existing column:***

**Syntax :**

**ALTER Table table\_name**

**ADD constraint constraint\_name**

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id);**

## 

## ***Add New Column and Add foreign key to the new column:***

**Syntax :**

**ALTER Table table\_name**

**ADD COLUMN column\_name Datatype,**

**ADD constraint constraint\_name**

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id);**

## **ON DELETE Clause:**

The **ON DELETE** clause in SQL is **used when defining a foreign key constraint to specify the action that should be taken if a referenced row in the parent table is deleted.**

It allows you to **define the behavior of child rows when a corresponding parent row is deleted.**

### ***1. ON DELETE CASCADE:***

* When a referenced **row in the parent table is deleted**, all corresponding **rows in the child table are also deleted automatically.**

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id) ON DELETE CASCADE**

### ***2. ON DELETE SET NULL:***

* When a referenced **row in the parent table is deleted, the foreign key column(s) in the child table are set to NULL.**
* This is applicable **only if the foreign key column(s) allow NULL values.**

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id) ON DELETE SET NULL**

### ***3. RESTRICT (Default):***

* Prevents the deletion of a parent row if there are corresponding child rows. The delete operation on the parent row is rejected with an error.

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id) ON DELETE RESTRICT**

### ***4. ON DELETE NO ACTION:***

* **Similar to RESTRICT,** it also prevents the deletion of a parent row if there are corresponding child rows.

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id) ON DELETE NO ACTION**

## ***ON UPDATE Clause:***

### ***1. ON UPDATE CASCADE:***

* **When the value** of the referenced column **in the parent table is updated**, all corresponding foreign key values in the **child table are also updated** to match the new value.
* This ensures consistency between parent and child rows.

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id) ON UPDATE CASCADE**

### ***2. ON UPDATE SET NULL:***

* **When the value** of the referenced column **in the parent table is updated,** the foreign key column(s) in the **child table are set to NULL if the new value does not exist** in the parent table.
* **This is applicable only if the foreign key column(s) allow NULL values.**

**FOREIGN KEY (parent\_id) REFERENCES parent\_table(parent\_id) ON UPDATE SET NULL**

### ***3. ON UPDATE RESTRICT (Default):***

### ***4. ON UPDATE NO ACTION:*** Similar to RESTRICT

# **JOINS:**

Joins in SQL are used to **combine rows from two or more tables based on a related column between them.**

Joins allow you to **retrieve data from multiple tables in a single query** by specifying how the rows from each table should be matched.

## **1. CROSS JOIN:**

Returns the Cartesian product of the two tables, resulting in every possible combination of rows from both tables. It doesn't require a join condition.

**Syntax 1:**

**SELECT**

**FROM table1**

**CROSS JOIN table2;**

**Syntax 2:**

**SELECT FROM table1,table2;**

**Syntax 3:**

**SELECT table1.column\_name,table2.column\_name,**

**FROM table1,table2;**

## **2. INNER JOIN:**

Returns rows from both tables where there is a match based on the specified join condition.

### ***EQUI JOIN:***

An equi join is a type of join in SQL that **combines rows from two or more tables based on matching values in specified columns**.

In an equi join, the join condition typically **uses the equality operator (=) to compare values in the specified columns**.

**Rows from the tables are combined if their values in the specified columns match.**

**Syntax 1:**

**SELECT FROM table1**

**INNER JOIN table2**

**ON table1.column\_name = table2.column\_name;**

**Syntax 2:**

**SELECT table1.name,table2.name FROM table1**

**INNER JOIN table2**

**ON table1.column\_name = table2.column\_name;**

### ***NON EQUI JOIN:***

A non-equi join is a type of join in SQL where the join condition involves a comparison other than equality between the columns of the joined tables.

**Syntax 1:**

**SELECT FROM table1**

**INNER JOIN table2**

**ON table1.column\_name = table2.column\_name;**

**Syntax 2:**

**SELECT table1.name,table2.name FROM table1**

**INNER JOIN table2**

**ON table1.column\_name = table2.column\_name;**

### ***NATURAL JOIN:***

A natural join is a type of join in SQL that **automatically joins tables based on columns with the same name and data type**.

Unlike other types of joins where you explicitly specify the columns to join on, a **natural join automatically matches columns with identical names in both tables.**

***Note:*** **Having more than one column with the same name in both tables doesn’t show the result.**

**Syntax:**

**SELECT**

**FROM table1**

**NATURAL JOIN table2;**

## **3. OUTER JOIN:**

### 

### ***LEFT OUTER JOIN (or LEFT JOIN):***

**Returns all rows from the left table (table1), and the matched rows from the right table (table2).** If there is no match, NULL values are returned for the columns from the right table.

**Syntax:**

**SELECT FROM table1**

**LEFT JOIN table2**

**ON table1.column\_name = table2.column\_name;**

### **RIGHT OUTER JOIN (or RIGHT JOIN):**

**Returns all rows from the right table (table2), and the matched rows from the left table (table1)**. If there is no match, NULL values are returned for the columns from the left table.

**Syntax:**

**SELECT FROM table1**

**RIGHT JOIN table2**

**ON table1.column\_name = table2.column\_name;**

### ***FULL OUTER JOIN:***

**Returns all rows from both tables, combining the results of both LEFT and RIGHT joins.** If there is no match, NULL values are returned for the columns from the table that lacks a matching row.

**Syntax:**

**SELECT FROM table1**

**FULL OUTER JOIN table2**

**ON table1.column\_name = table2.column\_name;**

## ***4. SELF JOIN:***

A self join is a type of join in SQL **where a table is joined with itself**.

It's **used when you want to compare rows within the same table** based on some condition.

In a self join, you **treat the table as if it were two separate tables, each with its own alias**, and then you join these "virtual" tables together.

**Syntax:**

**SELECT t1.column1, t2.column2 FROM table\_name t1**

**JOIN table\_name t2**

**ON t1.columnX = t2.columnY;**

# ***UNION and UNION ALL:***

The UNION operator in SQL is **used to combine the results of two or more SELECT statements into a single result set.**

By default, **UNION removes duplicate rows from the combined result** set.

**For duplicate rows we can use UNION ALL.**

1. Each SELECT statement within the UNION **must have the same number of columns** in the result sets.
2. The corresponding columns must have **compatible data types.**
3. The column names in the result set are taken from the first SELECT statement.

**Syntax:**

**SELECT column1, column2, ...FROM table1**

**UNION**

**SELECT column1, column2, ...FROM table2;**

**Syntax:**

**SELECT column1, column2, ...FROM table1**

**UNION ALL**

**SELECT column1, column2, ...FROM table2;**

# **IFNULL():**

The IFNULL() function in SQL is **used to handle NULL values**.

**Syntax:**

**IFNULL(expression, value\_if\_null)**

**expression:** The value that you want to check for NULL.

**value\_if\_null:** The value to return if the expression is NULL.

If the expression is not NULL, the IFNULL() function returns the value of the expression.

**Example:**

**SELECT IFNULL(salary, 100) AS adjusted\_salary**

**FROM employees;**

* The IFNULL(salary, 100) function checks each salary value. If a salary is NULL, it returns 100; otherwise, it returns the original salary value.
* The AS adjusted\_salary alias is given to the resulting column.

# **NULLIF():**

The NULLIF() function returns NULL if two expressions are equal; otherwise, it returns the first expression.

**Syntax:**

**NULLIF(expression1, expression2)**

**expression1** is the first expression to compare.

**expression2** is the second expression to compare.

**If expression1 is equal to expression2, the NULLIF() function returns NULL**. If expression1 is not equal to expression2, it returns expression1.

Example:

**SELECT NULLIF(age, 10) AS adjusted\_age**

**FROM employees;**

The NULLIF(age, 0) function checks each age value. **If an age is 10, it returns NULL; otherwise, it returns the original age value.**

The AS adjusted\_age alias is given to the resulting column.

**For Multiple row:**

**SELECT ,**

**CASE**

**WHEN age IN (10, 15) THEN NULL**

**ELSE age**

**END AS adjusted\_age**

**FROM employees;**

# **Subquery:**

A subquery, also **known as a nested query or inner query**, is a SQL query embedded within another SQL statement.

The **result of the inner query (subquery) is used as a part of the outer query.**

Subqueries can be used in various parts of a SQL statement, such as the SELECT, FROM, WHERE, and HAVING clauses.

**Types of Subqueries:**

## **1. Single-Row Subquery:**

A subquery that **returns only one row and one column**. It's typically used in conditions where a single value is expected.

**Syntax:**

**SELECT column1 FROM table1**

**WHERE column2 = (SELECT column2 FROM table2 WHERE condition);**

## **2. Multiple-Row Subquery:**

A subquery that returns multiple rows but only one column. It's used with operators like IN, ANY, or ALL.

**SELECT column1 FROM table1**

**WHERE column2 IN (SELECT column2 FROM table2 WHERE condition);**

# **VIEW:**

* A view in SQL is a **virtual table that is based on the result of a SELECT query.**
* **It behaves like a table, but its contents are dynamically generated when it is queried.**

## ***1. Creating a view:***

**Syntax:**

**CREATE VIEW view\_name AS**

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

**WHERE condition;**

Once a view is created, you **can query it like a regular table using SELECT statements.**

**Example:**

**SELECT \* FROM employee\_view;**

## ***2. Updating a View:***

In most databases, views are **read-only by default**, meaning you **cannot directly modify the underlying data** through the view. However, you **can update the data in the underlying tables, and those changes will be reflected in the view.**

## ***3. Dropping a View:***

To remove a view from the database, you use the DROP VIEW statement followed by the name of the view.

**Syntax:**

**DROP VIEW view\_name;**

**Advantages of Using Views:**

**1. Simplified Querying:** Views can encapsulate complex logic or joins into a single, easy-to-use object.

**2. Security:** Views can restrict access to certain columns or rows, providing an additional layer of security.

**3. Data Abstraction:** Views can hide the underlying table structure, making it easier to work with the data.