**SQL Notes**

**SQL Data Types**

In SQL, data types define the kind of data that can be stored in a column or variable.

Here are the Frequently used SQL data types:

1. **CHAR**

**Usage**: Suitable for storing fixed-length String data, such as codes, or identifiers, where the length is known and constant.

**Description:** String(0-255), can store characters of fixed length

**Example:**

CREATE TABLE Products (

ProductID INT,

ProductCode CHAR(10),

ProductName VARCHAR(100));

1. **VARCHAR**

**Usage:** VARCHAR is a variable-length character string data type. It allows you to store text data with a maximum length specified by 'n'. The 'n' can range from 1 to 65535 characters, depending on the database system.

**Description**: String(0-255), can store characters up to given length

**Example**:

CREATE TABLE Students (

StudentID INT,

Name VARCHAR(50),

Email VARCHAR(100));

**\*Note** – CHAR is for fixed length & VARCHAR is for variable length strings. Generally,

VARCHAR is better as it only occupies necessary memory & works more efficiently.

1. **BLOB** in oracle **/** **VARBINARY(MAX)** in ms sql

The BLOB (Binary Large Object) data type in SQL is used to store large binary data, such as images, videos, or documents. It can hold a variable amount of data, making it suitable for storing files and other binary content.

**Usage:** BLOB is commonly used when you need to store and retrieve large binary files within a database.

**Description**: String(0-65535), can store binary large object.

**Example:**

CREATE TABLE Images (

ImageID INT,

ImageName VARCHAR (100),

ImageData BLOB / VARBINARY(MAX) );

In this example, the ImageData column is defined as BLOB, allowing it to store binary data representing images.

1. **INT**

**Usage**: INT is commonly used for storing numeric data without decimal places.

**Description**: Integer ( -2,147,483,648 to 2,147,483,647)

**Example:**

CREATE TABLE Employees (

EmployeeID INT,

Name VARCHAR (50),

Age INT);

1. **TINYINT**

**Usage:** TINYINT is commonly used for storing small numeric values that don't require a lot of storage space.

**Description**: Integer (-128 to 127)

**Example:**

CREATE TABLE Students (

StudentID INT,

Name VARCHAR (50),

Grade TINYINT);

1. **BIGINT in ms sql & NUMBER in oracle**

**Usage:** BIGINT is commonly used for storing large numbers such as unique identifiers, timestamps, or quantities that require a wide range.

**Description**: Integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

**Example:**

CREATE TABLE Transactions (

TransactionID BIGINT/ NUMBER,

Amount DECIMAL (10, 2),

TransactionDate TIMESTAMP);

1. **BIT**

**Usage:** The BIT data type is used to store binary data, specifically representing boolean values of 0 or 1. It is commonly used for storing binary or boolean flags, such as true/false or yes/no indicators.

**Description**: Can store x-bit values. X can range from 1 to 64

**Example:**

CREATE TABLE Tasks (

TaskID INT,

TaskName VARCHAR (100),

IsCompleted BIT);

1. **FLOAT**

**Usage**: FLOAT is commonly used for storing numeric data that requires a floating-point representation, such as decimal values in scientific calculations or financial data.

**Description**: Decimal number – with precision to 23 digits

**Example:**

CREATE TABLE Measurements (

ID INT,

Temperature FLOAT,

Pressure FLOAT);

1. **DOUBLE** in ms sql & **BINARY\_DOUBLE** in oracle

**Usage:** DOUBLE is commonly used for storing numeric data with high precision, such as scientific calculations or financial data requiring accurate decimal representation.

**Description**: Decimal number – with 24 to 53 digits

**Example:**

CREATE TABLE StockPrices (

StockID INT,

StockSymbol VARCHAR (10),

Price DOUBLE);

1. **BOOLEAN**

**Usage**: BOOLEAN is commonly used for columns that need to hold binary or boolean values, such as yes/no or true/false indicators.

**Description**: Boolean values 0 or 1

**Example:**

CREATE TABLE Tasks (

TaskID INT,

TaskName VARCHAR (100),

IsCompleted BOOLEAN);

1. **DATE**

**Usage**: DATE is commonly used for columns that need to store specific dates, such as birthdates, appointment dates, or event dates. The DATE data type is used to store date values without any time components.

**Description**: Date format YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31

**Example:**

CREATE TABLE Appointments (

AppointmentID INT,

AppointmentDate DATE,

ClientName VARCHAR (100));

1. **YEAR**

**Usage:** YEAR is commonly used for columns that need to store specific year values, such as birth years, fiscal years, or historical events. It represents a specific year without any month or day components.

**Description:** Year in 4 digits format ranging from 1901 to 2155

**Example:**

CREATE TABLE Employees (

EmployeeID INT,

Name VARCHAR (50),

BirthYear YEAR);

1. **CLOB** in oracle

**CLOB** stands for Character Large Object. It is used to store large text data, such as long documents, HTML pages, or any text data that exceeds the storage capacity of the traditional VARCHAR2 data type.

**Example:**

CREATE TABLE documents (

id NUMBER PRIMARY KEY,

doc\_name VARCHAR2(100),

doc\_content CLOB);

**Types of SQL Commands**

1. **DDL (Data Definition Language)**: Used to create, alter, and delete database objects Like tables, indexes, etc. (*CREATE, DROP, ALTER, RENAME, TRUNCATE*)
2. **DQL (Data Query Language)**: Used to retrieve data from databases. (*SELECT*)
3. **DML (Data Manipulation Language)**: Used to modify the database. (*INSERT, UPDATE, DELETE*)
4. **DCL (Data Control Language)**: Used to grant & revoke permissions. (*GRANT, REVOKE)*
5. **TCL (Transaction Control Language)**: Used to manage transactions. (*COMMIT, ROLLBACK, START TRANSACTIONS, SAVEPOINT*)
6. **Data Definition Language (DDL):**

Data Definition Language (DDL) is a subset of SQL responsible for defining and managing the structure of databases and their objects.

DDL commands enable you to create, modify, and delete database objects like tables, Indexes, constraints, and more.

*Key DDL Commands are:*

1. **CREATE:**

"The CREATE command is used to create new database objects such as tables, views, indexes, or stored procedures"

**Example:**

*Create a database:*

CREATE DATABASE SampleDB;

*Create a table:*

CREATE TABLE employees (

Id INT PRIMARY KEY,

Salary DECIMAL (10,2));

*Create a index:*

CREATE INDEX idx\_DepartmentID ON Employees (DepartmentID);

*Create a view:*

CREATE VIEW EmployeeView AS

SELECT EmployeeID, FirstName, LastName

FROM Employees

WHERE DepartmentID = 1;

*Create a Stored Procedure (in ms sql)*

CREATE PROCEDURE GetEmployeesByDepartmentID

@DeptID INT

AS

BEGIN

SELECT EmployeeID, FirstName, LastName

FROM Employees

WHERE DepartmentID = @DeptID;

END;

1. **DROP:**

It is used to permanetaly Delete the Database objects such as Tables, Views, Indexes or Store Procedures, including all associated data and metadata. It is a powerful command and should be used with caution, as there is no way to undo the action once it's executed.

DROP DATABASE mydatabase;

DROP TABLE employees;

DROP VIEW employee\_view;

DROP INDEX idx\_employee\_id ON employees;

DROP PROCEDURE sp\_get\_employee\_info;

1. **ALTER:**

The ALTER command in SQL is used to modify existing database objects like tables, views, procedures, or functions. It allows you to make changes such as adding, modifying, or dropping columns in a table, changing the data type of a column, renaming objects, or even altering the structure of a table.

**Example:**

*Adding a New Column:*

ALTER TABLE employees

ADD email VARCHAR(100);

*Modifying Column Data Type:*

*(in oracle)*

ALTER TABLE employees

MODIFY salary DECIMAL(10, 2);

*(in ms sql)*

ALTER TABLE employees

ALTER COLUMN salary DECIMAL(10, 2);

*Delete Column:*

ALTER TABLE employees

DROP COLUMN email;

*Adding a Primary Key Constraint:*

ALTER TABLE employees

ADD CONSTRAINT pk\_employee\_id PRIMARY KEY (employee

\_id);

1. **TRUNCATE:**

It is used to quickly delete the all records from table without removing the table structure such as columns, constraints,indexes, etc.

TRUNCATE TABLE employees;

1. **RENAME:**

The RENAME command in used to change the name of an existing database object, such as a table or a column. It allows you to modify the name of the object without altering its structure or data.

*Rename Column name:*

*(In oracle)*

ALTER TABLE employees

RENAME COLUMN emp\_department TO department\_name;

*(In ms sql server)*

-- Rename Table

sp\_rename 'old\_name','new\_name'

--Rename Column

sp\_rename 'new\_name.id','stu\_ID','COLUMN'

1. **Data Manipulation Language (DML):**

DML stand for Data manipulation Language its allow to query edit, add, and remove data stored in database objects.

**1) INSERT**

It is used to Insert new Records into a table

INSERT INTO employee VALUES(2,'Abhijit')

**2) UPDATE**

It is Used to change the values of one or more columns in existing roes of a table based on specific conditions.

UPDATE employee SET stu\_ID = 50 WHERE stu\_ID = 2

**3) DELETE**

It is used to remove one or more rows from table based on a specific condition & all records can be removed in single go.

DELETE FROM employee WHERE stu\_id = 50

1. **Data Query Language (DQL):**

DQL stand for Data Query Language it is perform on the data in a database to retrieve the necessary information from it.

**1) SELECT**

A SELECT statement retrieves zero or more rows from one or more database table or views.

SELECT \* FROM employee

1. **Data Control Language (DCL):**

DCL is Stand for Data Control language it is like the authority you have to control who can do what with the data in a database. It's all about managing permissions and deciding who gets to access, modify, or delete data.

**1) GRANT**

it is used to give specific privileges or permissions to a user allows the perform, such as SELECT, INSERT, UPDATE, DELETE in the database.

**2) REVOKE**

It is used to take back or revoke previously granted privileges from a user or group of users in a database.

1. **Transaction Control Language (TCL):**

TCL Stand for Transaction Control language it is used to manage transactions within a database. These commands are used to control the transactions' start, end, and rollback points.

**1) COMMIT**

This command is used to permanently save the changes made in the current transaction to the database.

**2) ROLLBACK**

It is used to undo the changes made in the current transaction since the last COMMIT or SAVEPOINT command.

**3) SAVEPOINT**

This command sets a point in the transaction to which you can later roll back if needed. It allows for partial rollback within a transaction.

**Constraints**

The Constraints are the rules applied to columns in a database table that enforce data integrity and ensure data consistency.

They define limits or conditions that data entered into a column must meet.

**1) NOT NULL**

The NOT NULL constraint in a database ensures that a column cannot have a NULL value. It enforces that every row in the table must have a value for this column. This is useful for ensuring data integrity by making sure that critical fields always contain valid data.

**2) UNIQUE**

A UNIQUE constraint in a database ensures that all values in a column (or a combination of columns) are distinct, meaning no duplicate values are allowed. It enforces the uniqueness of the data stored in the specified columns.

**3) PRIMARY KEY**

A PRIMARY KEY constraint in a database uniquely identifies each record in a table. It ensures that no duplicate values exist and that no null values are allowed for the column or combination of columns it is applied to.

**4) FOREIGN KEY**

A FOREIGN KEY constraint in a database is a rule that ensures the values in a column match values in a column of another table, creating a link between the two tables. This helps maintain data integrity by ensuring that a value in one table corresponds to an existing value in another table, enforcing relationships between tables.

**5) CHECK**

A CHECK constraint is a rule applied to a column in a database table to ensure that all values in that column meet a specific condition. It helps maintain data integrity by restricting the type of data that can be stored in the column.

For example, a CHECK constraint can ensure that the values in an "age" column are always greater than 0.

**6) DEFAULT**

A DEFAULT constraint is used to set a default value for a column when no value is specified during the insertion of a new record.

Constraints play a crucial role in maintaining the accuracy, reliability, and security of database systems.

**Logical operator**

A logical operator is a symbol or keyword used to perform logical operations on expressions or conditions in sql queries. These operators are essential for creating complex conditions to filter data effectively.

1) AND

The AND operator returns true if both conditions on either side of the operator are true.

Eg. `condition1 AND condition2`.

2) OR

The OR operator returns true if at least one of the conditions on either side of the operator is true.

Eg. `condition1 OR condition2`.

3) NOT

The NOT operator is used to negate a condition. It returns true if the condition following NOT is false and vice versa.

Eg. `NOT condition1`.

4) IN

The IN operator is used to check if a value matches any value in a list. Eg. `value IN (list)`.

5) BETWEEN

The BETWEEN operator is used to check if a value falls within a specified range.

Eg. `value BETWEEN low AND high`.

6) LIKE

The LIKE operator is used for pattern matching with wildcard characters such as `%` and `\_`.

Eg. `column LIKE 'pattern'`.

7) IS NULL

The IS NULL operator checks if a value is NULL (missing or unknown).

Eg. `column IS NULL`.

8) IS NOT NULL

The IS NOT NULL operator checks if a value is not NULL.

Eg. `column IS NOT NULL`.

9) ALL

ALL means that the condition will be true only if the operation is true for all values in the range.

Eg. `where column = AND (subquery)`

10) ANY

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

Eg. `where column = ANY (subquery)`

11) EXISTS

he 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.

Eg. `where EXISTS (subquery)`

12) SOME

SOME op is same like ANY op it condition will be true if the operation is true for any of the values in the range.

Eg. `where column = SOME (subquery)`

These operators allow to create flexible and precise conditions to retrieve, update, or delete data from a database.

**Aggregate Function / Numerical Function**

1. COUNT

Counts the number of rows in a result set.

SELECT COUNT(Name) FROM employees;

1. AVG

Calculates the average of a set of values in a column.

SELECT AVG(Salary) FROM employees;

1. SUM

Calculates the sum of a set of values.

SELECT SUM(Salary) FROM employees;

1. MIN

Returns the minimum value in a set of values.

SELECT MIN(Salary) FROM employees;

1. MAX

Returns the maximum value in a set of values.

SELECT MAX(Salary) FROM employees;

1. STRING\_AGG (in ms sql) & LISTAGG (in oracle)

Concatenates values from multiple rows into a single string using a specified separator.

SELECT STRING\_AGG(Name, ', ') FROM Employees

SELECT LISTAGG(Name, ', ') WITHIN GROUP (ORDER BY Name) FROM Employees;

1. ROUND

The ROUND function is a fundamental tool that adjusts the precision of numerical data in SQL. It rounds values to a specified number of decimal places and simplifies data for analysis

SELECT AVG(salary) AS avg\_sal, ROUND(AVG(salary), 2) AS round\_avg\_sal FROM employee\_salary;

1. COUNT\_BIG (this func not valid in oracle you can use COUNT)

Similar to COUNT but returns a bigint data type.

Useful for counting large result sets where COUNT might overflow.

SELECT COUNT\_BIG(Salary) FROM employees;

1. GROUPING

Indicates whether a specified column expression in a GROUP BY list is aggregated or not. Used with GROUP BY ROLLUP or GROUP BY CUBE

SELECT region, product, SUM(sales\_amount) AS total\_sales,

GROUPING(region) AS is\_region\_total,

GROUPING(product) AS is\_product\_total

FROM sales

GROUP BY ROLLUP/CUBE(region, product);

1. GROUPING\_ID

Returns the bitwise-AND of the GROUPING values for the grouping sets.

Used with GROUP BY ROLLUP or GROUP BY CUBE to recognize different levels of grouping.

SELECT region, product, SUM(sales\_amount) AS total\_sales,

GROUPING\_ID(region, product) AS grouping\_id

FROM sales

GROUP BY ROLLUP(region, product);

1. APPROX\_COUNT\_DISTINCT

Returns an approximate count of distinct values in a column.

More efficient than COUNT(DISTINCT) for large datasets but may have a slight margin of error.

SELECT APPROX\_COUNT\_DISTINCT(Salary) FROM employees;

1. CHECKSUM\_AGG (this func not available in oracle)

Calculates a checksum value for a group of values.

Useful for detecting changes in data.

SELECT CHECKSUM\_AGG(Salary) FROM employees;

1. STDEV (ms sql) & STDDEV (oracle)

Calculates the sample standard deviation of a set of values.

SELECT STDEV(sales\_amount) AS st\_deviation FROM sales;

SELECT STDDEV(sales\_amount) AS st\_deviation FROM sales;

1. STDEVP (ms sql) & STDDEV\_POP (oracle)

Calculates the population standard deviation of a set of values.

SELECT STDEVP(sales\_amount) AS st\_deviation FROM sales;

SELECT STDDEV\_POP(sales\_amount) AS st\_deviation FROM sales;

1. VAR (ms sql) & VAR\_SAMP (oracle)

Calculates the sample variance of a set of values.

SELECT VAR(sales\_amount) AS variance FROM sales;

SELECT VAR\_SAMP(sales\_amount) AS variance FROM sales;

1. VARP (ms sql) & VAR\_POP (oracle)

Calculates the population variance of a set of values.

SELECT VARP(sales\_amount) AS p\_variance FROM sales;

SELECT VAR\_POP(sales\_amount) AS p\_variance FROM sales;

**String Function / Character Function**

1)\_UPPER

The UPPER function is used to convert all characters in a string to uppercase.

SELECT UPPER(first\_name) FROM employees;

2) LOWER

The LOWER function is used to convert all characters in a string to lowercase.

SELECT LOWER(first\_name) FROM employees;

3) CONCAT

The CONCAT function is used to concatenate two or more strings into a single string

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

FROM employees;

4) SUBSTRING (ms sql) & SUBSTR (oracle)

The SUBSTRING function is used to extract a portion of a string, starting at a specified position and for a specified length.

SELECT SUBSTRING(Cust\_Name, 1, 5) FROM Customers;

SELECT SUBSTR(Cust\_Name, 1, 5) FROM Customers;

5) LEN (ms sql) & LENGTH (oracle)

The LEN function in SQL is used to return the number of characters in a string, excluding trailing spaces.

SELECT \* FROM customers

WHERE LEN(phone\_number) <> 10;

SELECT \* FROM customers

WHERE LENGTH(phone\_number) <> 10;

6)\_LTRIM

The LTRIM function is used to remove the left side spaces from a string.

SELECT LTRIM(' Abhijit')

SELECT LTRIM(' Abhijit') FROM dual (oracle)

6)\_RTRIM

The RTRIM function is used to remove the right side spaces from a string.

SELECT RTRIM('Abhijit ')

SELECT RTRIM('Abhijit ') FROM dual (oracle)

\*NOTE: Dual is a dummy table already present in oracle

7) PATINDEX (this func not present in oracle)

The PATINDEX function is returns the starting position of the first occurrence of a specified pattern in a string, where the pattern can include wildcard characters. If the pattern is not found, it returns 0.

SELECT ProductName FROM Products

WHERE PATINDEX('Apple%Phone', ProductName) > 0;

SELECT PATINDEX('%[a-z]%', 'SQL123Server') AS Position;

8) CHARINDEX (ms sql) & INSTR (oracle)

The CHARINDEX function is used to find the starting position of a substring within a string.

SELECT \* FROM Products

WHERE CHARINDEX('apple', ProductName) > 0;

SELECT \* FROM employees

WHERE INSTR(full\_name, 'Abhijit') > 0;

\*NOTE:

* + **PATINDEX** supports wildcards (**%**, **\_**) and allows pattern matching similar to the **LIKE** operator.
  + **CHARINDEX** does not support wildcards and only searches for exact substring matches.

9) REPLACE

The REPLACE function is used to replace all occurrences of a specified substring within a string with another substring.

SELECT REPLACE(address, 'Pandharpur', 'Pune') FROM customers;

9) TRANSLATE

The TRANSLATE function is used to replace each character in a string with a corresponding character in another string or to remove characters from a string.

SELECT TRANSLATE ('Hi, i am Abhijet', 'e', 'i') FROM dual;

10) INITCAP (not present in ms sql)

The **INITCAP** function is used to capitalize the first letter of each word in a string while converting the remaining letters to lowercase.

SELECT INITCAP(FirstName) FROM Employees;

**Date Function**

1. GETDATE (ms sql) & SYSDATE (oracle)

Returns the current date and time.

SELECT GETDATE() AS CurrentDateTime;

SELECT SYSDATE AS CurrentDateTime FROM dual;

1. SYSDATETIME (ms sql) & SYSTIMESTAMP (oracle)

Returns the current date and time with more precision.

SELECT SYSDATETIME() AS CurrentDateTime;

SELECT SYSTIMESTAMP AS CurrentDateTime FROM dual;

1. DATEADD (ms sql) & ADD\_MONTHS (oracle)

Adds a specified number of units to a date. (oracle add only month)

SELECT DATEADD(day, 10, GETDATE()) AS NewDate;

SELECT ADD\_MONTHS(SYSDATE, 2) AS NewDate FROM dual;

1. DATEDIFF (ms sql) & MONTHS\_BETWEEN (oracle)

Returns the difference between two dates.

SELECT DATEDIFF(day, '2023-01-01', '2023-12-31')

SELECT MONTHS\_BETWEEN(TO\_DATE('2023-12-31', 'YYYY-MM-DD'), TO\_DATE('2023-01-01', 'YYYY-MM-DD')) FROM dual;

1. FORMAT (ms sql) & TO\_CHAR (oracle)

Formats a date value based on a format pattern.

SELECT FORMAT(GETDATE(), 'DD-MM-YYYY');

SELECT TO\_CHAR (SYSDATE, 'DD-MM-YYYY') FROM dual;

1. DATEPART (ms sql) & EXTRACT (oracle)

Returns a specified part of a date.

SELECT DATEPART(MONTH, OrderDate) FROM Orders

WHERE OrderID = 1001;

SELECT EXTRACT(MONTH FROM OrderDate) FROM Orders

WHERE OrderID = 1001;

1. EOMONTH (ms sql) & LAST\_DAY (oracle)

Returns the last day of the month containing a specified date.

SELECT EOMONTH('2024-05-15');

SELECT LAST\_DAY(TO\_DATE('2024-05-15', 'YYYY-MM-DD')) FROM dual;

**Clause**

1. WHERE

The WHERE clause is used to specify conditions that filter which records should be retrieved, updated, or deleted from a database table. It is most commonly used in SELECT, UPDATE, DELETE, and INSERT statements to control the selection of rows.

SELECT \* FROM Employees

WHERE department = 'HR';

1. ORDER BY

The ORDER BY clause allows you to specify the one or more columns by which you want to sort the result set, and you can control whether the sorting is ascending or descending.

SELECT \* FROM Employees

ORDER BY Salary ASC, Name ASC;

1. GROUP BY  
   The GROUP BY clause is used to arrange identical data into groups. This means that it aggregates data across multiple records and groups the results based on one or more columns. It's commonly used with aggregate functions such as COUNT(), SUM**()**, AVG**()**, MAX**()**, and MIN**()** to perform operations on each group of data.

SELECT Category, SUM(Revenue) AS TotalRevenue

FROM Sales

GROUP BY Category;

1. HAVING  
   The HAVING clause is used to apply a filter to the result set after the GROUP BY clause has grouped the rows. This clause is particularly useful when you want to apply conditions to aggregated data in grouped results.

SELECT product, SUM(quantity) AS total\_quantity

FROM sales

GROUP BY product

HAVING SUM(quantity) > 100;

\*NOTE:

WHERE & HAVING are same but they operate at different stages of the query.

* + **WHERE:** Applied before grouping and aggregation**.**
  + **HAVING:** Applied after grouping and aggregation**.**

**Like**

The LIKE command is used to search for a specified pattern in a column. It is often used with the WHERE clause to filter rows based on patterns.

The LIKE operator can be used with two wildcards:

1. **Percent sign (%)**: Represents zero, one, or multiple characters.
2. **Underscore (\_)**: Represents a single character.

SELECT \* FROM employee

WHERE emp\_Name LIKE 'A%';

**Between**

The BETWEEN operator is used to filter the result set within a specified range. It is inclusive, meaning it includes the start and end values in the range.

SELECT \* FROM employees

WHERE salary BETWEEN 50000 AND 100000;

**TOP**

TOP (ms sql) & ROWNUM (oracle)

The TOP command is used to limit the number of rows returned in a query result. It is often used with a SELECT statement to fetch a specified number of records from a table.

SELECT TOP 5 \* FROM Employees

order by salary

SELECT TOP 10 PERSENT \* FROM Employees

(PERSENT keyword return the specific percentage of rows in from the table)

SELECT \* FROM Employees

WHERE ROWNUM <= 5;

**Nested Command / Sub Query**

A subquery is a query nested inside another query. It is used to perform operations that require multiple steps, such as filtering, aggregating, or transforming data, within a single query.

SELECT EmployeeName

FROM Employees

WHERE DepartmentID = (SELECT DepartmentID

FROM Departments

WHERE DepartmentName = 'Sales');

**Distinct**

The DISTINCT keyword is used to remove duplicate records from the result set of a query. It ensures that the returned values are unique. (including null value)

SELECT DISTINCT department\_id

FROM employees;

**Set Operator**

**set operator** is used to combine the results of two or more SELECT queries into a single result set. The main set operators are UNION, UNIONALL, INTERSECT, and EXCEPT (or MINUS in some databases). Each operator has a specific function for combining query results.

1. UNION

The UNION operator is return the distinct values in two or more select queries including null value it’s eliminate duplicate rows by default. Each SELECT statement within the UNION must have the same number of columns in the result sets with similar data types.

SELECT name, email FROM customers

UNION

SELECT name, email FROM employees;

1. UNION ALL

The UNION ALL operator is used to combine the result sets of two or more SELECT queries. It is return the all rows including duplicates and null values.

SELECT name, email FROM customers

UNION ALL

SELECT name, email FROM employees;

1. INTERSECT

The INTERSECT operator is used to return the common records from two or more SELECT statements. the result will not include any duplicate rows. (null value include)

SELECT name, email FROM customers

INTERSECT

SELECT name, email FROM employees;

1. EXCEPT

The EXCEPT operator is retrieves only distinct records from the first query that are not present in the result of the second query. (null value including)

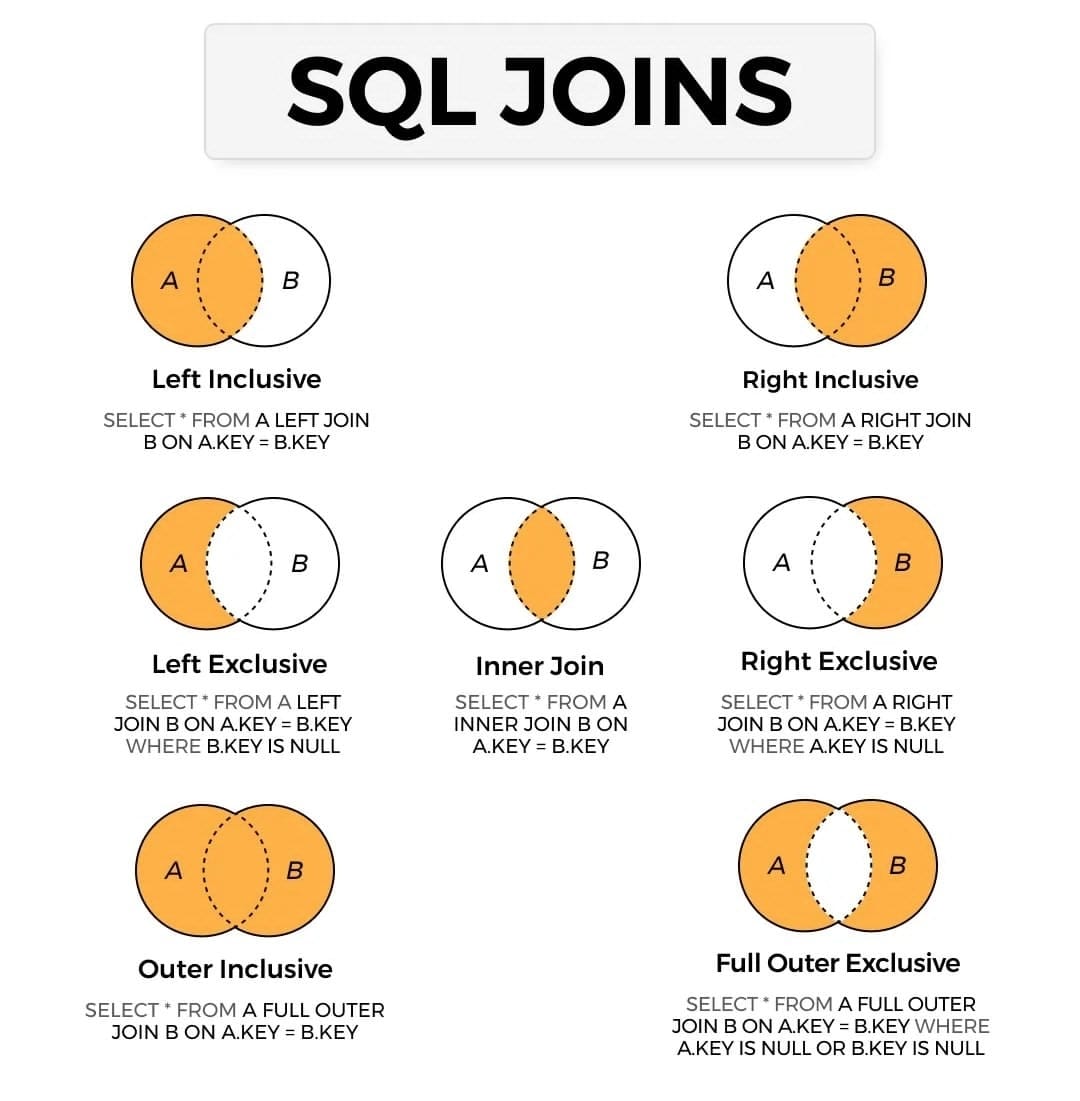
SELECT name, email FROM customers

EXCEPT

SELECT name, email FROM employees;

**JOINS**

The Joins operation combines data or rows from two or more tables based on a common field between them.



1. INNER JOIN

A INNER JOIN is retrieves records from multiple tables where the values in the specified columns match in both tables. (null value is not include)

SELECT \* FROM employees INNER JOIN departments

ON employees.dept\_id = departments.id;

1. LEFT JOIN

A Left join is returns all rows from the left table and the matched rows from the right table. If there is no match, the result is NULL from the right table. (null value including)

SELECT \* FROM employees LEFT JOIN departments

ON employees.dept\_id = departments.id;

1. RIGHT JOIN

A right join is returns all rows from the right table and the matched rows from the left table. If there is no match, NULL values are returned for columns from the left table. (null value including)

SELECT \* FROM employees RIGHT JOIN departments

ON employees.dept\_id = departments.id;

1. FULL JOIN

A full outer join is returns all records when there is a match in either the left or right table records. It combines the results of both left and right outer joins, including rows that do not have matches in the other table, with NULLs in the place of missing matches. (null value include)

SELECT \* FROM employees FULL OUTER JOIN departments

ON employees.emp\_id = departments.emp\_id;

1. CROSS JOIN

A cross join is returns the Cartesian product of two tables, meaning it combines all rows from the first table with all rows from the second table, without any condition.

SELECT \* FROM Employees CROSS JOIN Departments;

**INDEX**

A index is that helps speed up the retrieval of rows from a table It works like a book's index, allowing the database management system to quickly locate and access specific rows or ranges of rows within a table. And If primary key is created then indexes are automatically created.

1. Clustered Index

A clustered index determines the physical order of data in a table. Because the data rows are stored in the order of the clustered index key, there can be only one clustered index per table. This index is often used to improve the performance of queries that retrieve data within a range of values or sorted data.

CREATE CLUSTERED INDEX idx\_EmployeeID

ON Employees(EmployeeID);

1. Non-Clustered Index

A Non-clustered index determines the virtually order of data in a table. The data is stored in one place, and the index is stored in another place. There you can be create multiple non clustered index in a one table

CREATE NONCLUSTERED INDEX idx\_LastName

ON Employees (LastName);

1. Column Store Index
2. Filtered Index
3. Hash Index
4. Unique Index