### SQL Query Execution Order

1. **FROM clause**: Specifies the tables involved in the query. Joins between tables are processed here.
2. **WHERE clause**: Filters rows based on specified conditions.
3. **GROUP BY clause**: Groups rows that have the same values in specified columns into summary rows.
4. **HAVING clause**: Filters groups based on specified conditions.
5. **SELECT clause**: Specifies the columns to be returned by the query.
6. **ORDER BY clause**: Sorts the result set.
7. **LIMIT/OFFSET clause**: Limits the number of rows returned by the query and specifies an offset to skip a number of rows

**Query Execution Order in General**

Here's the general order of operations in SQL query processing:

1. **FROM**: Identify the tables.
2. **JOIN**: Join tables and create a virtual table.
3. **WHERE**: Apply filters to the rows.
4. **GROUP BY**: Group the rows.
5. **HAVING**: Apply filters to the groups.
6. **SELECT**: Select the columns and apply any expressions.
7. **DISTINCT**: Remove duplicate rows.
8. **ORDER BY**: Sort the result set.
9. **LIMIT/OFFSET**: Limit the number of rows and specify an offset.

**Visualization of Execution Order**

Think of SQL query execution like building a data pipeline:

1. **FROM** → **JOIN** → **WHERE**: Data loading and initial filtering.
2. **GROUP BY** → **HAVING**: Grouping and filtering groups.
3. **SELECT**: Final column selection.
4. **DISTINCT** → **ORDER BY**: Final sorting and removing duplicates.
5. **LIMIT/OFFSET**: Paginating results.

**Example with Incorrect Alias Usage**

If you try to use an alias in the GROUP BY clause directly, it would not work because the GROUP BY clause is processed before the SELECT clause, where the alias is defined.

**LIMIT Clause**

The LIMIT clause restricts the number of rows returned by a query.

### OFFSET Clause

The OFFSET clause skips a specified number of rows before starting to return rows.

**Example:** Skip the first 10 rows and then retrieve all remaining rows from the employees table.

SELECT \*

FROM employees

OFFSET 10;

**Example:** Retrieve rows 11 to 20 from the employees table.

SELECT \*

FROM employees

LIMIT 10

OFFSET 10;

**Alias use:**

Aliases are defined by writing the table name followed by the alias. Here’s the basic syntax:

SELECT column\_name(s)

FROM table\_name AS alias\_name

The AS keyword is optional, so you can also write it as:

SELECT column\_name(s)

FROM table\_name alias\_name

**Question 1: Employee Salary Analysis**

**Scenario:** You have a **table** employees with **columns** employee\_id, first\_name, last\_name, department\_id, and salary. Write a query to find the department with the highest average salary.

SELECT department\_id

FROM employees

GROUP BY department\_id

ORDER BY AVG(salary) DESC

LIMIT 1;

**Question 2: Sales Data Aggregation**

**Scenario:** You have a table sales with columns sale\_id, product\_id, sale\_date, and amount. Write a

**query to find the total sales amount for each month in the year 2023.**

SELECT DATE\_TRUNC('month', sale\_date) AS month, SUM(amount) AS total\_sales

FROM sales

WHERE EXTRACT(YEAR FROM sale\_date) = 2023

GROUP BY DATE\_TRUNC('month', sale\_date)

ORDER BY month;

**SELECT DATE\_TRUNC('month', sale\_date) AS month, SUM(amount) AS total\_sales**

* DATE\_TRUNC('month', sale\_date) AS month: The DATE\_TRUNC function truncates (shortened) the sale\_date to the beginning of the month, effectively grouping all dates within the same month to a single value (e.g., '2023-01-01' for January 2023). This new value is aliased as month.
* SUM(amount) AS total\_sales: The SUM function calculates the total sales amount for each group of truncated dates (i.e., each month). This sum is aliased as total\_sales.

**The SUM(amount) function calculates the total sales amount for each month group.**

**(** so , sum() ta date trunc er porei hbe,)

**GROUP BY DATE\_TRUNC('month', sale\_date)**

Groups the rows by the truncated month value. This means that all sales within the same month are aggregated together.

ORDER BY month;

**Question 3: Customer Purchase Frequency**

**Scenario:** You have a table purchases with columns customer\_id, purchase\_date, and amount.

**Write a query to find the customers who made more than 5 purchases in the last 6 months.**

SELECT customer\_id, COUNT(\*) AS purchase\_count

FROM purchases

WHERE purchase\_date >= CURRENT\_DATE - INTERVAL '6 months'

GROUP BY customer\_id

HAVING COUNT(\*) > 5;

**Question 4: Inventory Management**

**Scenario:** You have two tables: products with columns product\_id, product\_name, and price, and inventory with columns product\_id and quantity.

**Write a query to find the total value of inventory for each product.**

SELECT p.product\_id, p.product\_name, p.price, i.quantity, (p.price \* i.quantity) AS total\_inventory\_value

FROM products p

JOIN inventory i ON p.product\_id = i.product\_id;

**Question 5: Order Fulfillment**

**Scenario:** You have two tables: orders with columns order\_id, customer\_id, and order\_date, and order\_items with columns order\_id, product\_id, and quantity.

**Write a query to find the top 5 products that have been ordered the most.**

SELECT oi.product\_id, SUM(oi.quantity) AS total\_quantity

FROM order\_items oi

GROUP BY oi.product\_id

ORDER BY total\_quantity DESC

LIMIT 5;

**Question 6: Revenue from New Customers**

**Scenario:** You have two tables: customers with columns customer\_id, first\_name, last\_name, and registration\_date, and sales with columns sale\_id, customer\_id, sale\_date, and amount. Write a query to find the total revenue from customers who registered in the year 2023.

SELECT SUM(s.amount) AS total\_revenue

FROM sales s

JOIN customers c ON s.customer\_id = c.customer\_id

WHERE EXTRACT(YEAR FROM c.registration\_date) = 2023;

**Question 7: Employee Tenure**

**Scenario:** You have a table employees with columns employee\_id, first\_name, last\_name, hire\_date, and department\_id. Write a query to find the average tenure of employees in each department.

sql

Copy code

SELECT department\_id, AVG(CURRENT\_DATE - hire\_date) AS avg\_tenure

FROM employees

GROUP BY department\_id;

**Question 8: Product Sales Trends**

**Scenario:** You have a table sales with columns sale\_id, product\_id, sale\_date, and amount. Write a query to find the product with the highest sales growth (percentage increase) between two consecutive months.

sql

Copy code

WITH monthly\_sales AS (

SELECT product\_id, DATE\_TRUNC('month', sale\_date) AS month, SUM(amount) AS total\_sales

FROM sales

GROUP BY product\_id, DATE\_TRUNC('month', sale\_date)

),

sales\_growth AS (

SELECT product\_id, month, total\_sales,

LAG(total\_sales) OVER (PARTITION BY product\_id ORDER BY month) AS prev\_month\_sales,

(total\_sales - LAG(total\_sales) OVER (PARTITION BY product\_id ORDER BY month)) / NULLIF(LAG(total\_sales) OVER (PARTITION BY product\_id ORDER BY month), 0) AS growth

FROM monthly\_sales

)

SELECT product\_id, month, growth

FROM sales\_growth

WHERE growth IS NOT NULL

ORDER BY growth DESC

LIMIT 1;

**Question 9: Duplicate Records**

**Scenario:** You have a table transactions with columns transaction\_id, customer\_id, transaction\_date, and amount. **Write a query to find duplicate transactions made on the same day by the same customer.**

SELECT customer\_id, transaction\_date, COUNT(\*) AS transaction\_count

FROM transactions

GROUP BY customer\_id, transaction\_date

HAVING COUNT(\*) > 1;

**Question 10: Order Completion Time**

**Scenario:** You have two tables: orders with columns order\_id, customer\_id, and order\_date, and deliveries with columns order\_id, delivery\_date.

Write a query to **find the average time taken to complete an order (from order date to delivery date).**

SELECT AVG(delivery\_date - order\_date) AS avg\_completion\_time

FROM orders o

JOIN deliveries d ON o.order\_id = d.order\_id;

Git commands:

C:\Users\SDO>cd C:\Users\SDO\Desktop\Project\Pd-Np\git – to enter in a specific file

Git – to check wheather git is installed or not

Git –version – 2.44.0…..

Git config -l 🡪 user name/ pass dekha jbe if it was set previously

Git config –global user.name ‘soumik’ 🡪 to set username

Git config –global user.email ‘….@gmail.com’

Git config -l 🡪 to check wheather the name is set or not

Git init🡪 to initalise a file as git repo

Git status🡪 wheather the file is tracked or not, …etc

Wildcard = if we want to stage some particular files but not all the files , and without the naming of all files we can stage them . let suppose there are 10 files with the starting name is ‘prod’ then we just need to run

Git add prod\*

Git add . 🡪 all the files will staged for commit

Git reset <file name> or git rm –cached <file name> 🡪 for unstage the file which is already staged

Git diff 🡪 [what is changed but not staged] modified file a ki ki change hoyeche (add, delete)

Git diff –staged 🡪 changes staged but not commited

Git commit -m “ file added to master branch”

Git commit -a -m “add and commit both at a single command”

Git rm <file name > = to untrack the file

Git revert 🡪 delete

[primary key](https://www.geeksforgeeks.org/primary-key-constraint-in-sql/): A [primary key](https://www.geeksforgeeks.org/primary-key-constraint-in-sql/) is used to ensure that data in the specific column is unique. A column cannot have NULL values.

| **PRIMARY KEY** | **FOREIGN KEY** |
| --- | --- |
| A primary key is used to ensure data in the specific column is unique. | A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. |
| It uniquely identifies a record in the relational database table. | It refers to the field in a table which is the primary key of another table. |
| Only one primary key is allowed in a table. | Whereas more than one foreign key is allowed in a table. |
| It is a combination of UNIQUE and Not Null constraints. | It can contain duplicate values and a table in a relational database. |
| It does not allow NULL values. | It can also contain NULL values. |
| Its value cannot be deleted from the parent table. | Its value can be deleted from the child table. |
| It constraint can be implicitly defined on the temporary tables. | It constraint cannot be defined on the local or global temporary tables. |

Aggregate () is used for max, min, avg, sum, count cases.

Q-1)If i want to update all the elements of a column let suppose column name 'dept\_id' values (101,102,103,104) I want to update it with (1,2,3,4)

Existing tables

select \* from employees

inner join department

on employees.dept\_id = department.dept\_id;

UPDATE employees

SET dept\_id = CASE

    WHEN dept\_id = 101 THEN 1

    WHEN dept\_id = 102 THEN 2

    WHEN dept\_id = 103 THEN 3

    WHEN dept\_id = 104 THEN 4

    ELSE dept\_id

END;

* The **ELSE dept\_id** clause ensures that if **dept\_id** does not match any of the specified values (101, 102, 103, 104), it remains unchanged.
* The **END** keyword marks the end of the **CASE** expression.

CREATE TABLE employee (

    ID SERIAL PRIMARY KEY,

    EMPNO VARCHAR(10) UNIQUE NOT NULL,

    NAME VARCHAR(100) NOT NULL,

    DEPT\_ID INT,

);

INSERT INTO employee (EMPNO, NAME, DEPT\_ID, MANAGER\_ID, PHONE\_NO, HIRE\_DATE, SALARY, BONUS, COMM)

VALUES

('E001', 'John Doe', 1, NULL, '123-456-7890', '2020-01-15', 60000, 5000, 2000),

('E002', 'Jane Smith', 1, 1, '234-567-8901', '2019-05-20', 65000, 5500, 2500),

('E003', 'Emily Johnson', 2, 1, '345-678-9012', '2021-03-10', 70000, 6000, 3000),

('E004', 'Michael Brown', 2, 2, '456-789-0123', '2020-07-05', 75000, 6500, 3500),

**Some important Quaries:**

-- SELECT Statement

SELECT \* FROM employees;

-- INSERT Statement

INSERT INTO employees (first\_name, last\_name, email) VALUES ('John', 'Doe', 'john.doe@example.com');

-- UPDATE Statement

UPDATE employees SET email = 'johndoe@example.com' WHERE employee\_id = 1;

-- DELETE Statement

DELETE FROM employees WHERE employee\_id = 1;

-- JOIN Operations

SELECT employees.first\_name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;

-- GROUP BY and Aggregation Functions

SELECT department\_id, COUNT(\*) as employee\_count

FROM employees

GROUP BY department\_id;

-- ORDER BY

SELECT \* FROM employees

ORDER BY last\_name ASC, first\_name ASC;

-- LIMIT and OFFSET

SELECT \* FROM employees

LIMIT 10 OFFSET 5;

-- Subqueries

SELECT first\_name, last\_name

FROM employees

WHERE department\_id IN (SELECT department\_id FROM departments WHERE department\_name = 'IT');

-- CTE (Common Table Expressions)

WITH EmployeeCTE AS (

    SELECT first\_name, last\_name

    FROM employees

    WHERE department\_id = 1

)

SELECT \* FROM EmployeeCTE;

Some more…

-- DISTINCT

SELECT DISTINCT department\_id FROM employees;

-- Retrieves unique department IDs from the employees table, eliminating duplicates.

-- LIKE Operator

SELECT \* FROM employees WHERE first\_name LIKE 'J%';

-- Retrieves employees whose first names start with the letter 'J'.

-- BETWEEN Operator

SELECT \* FROM employees WHERE salary BETWEEN 50000 AND 80000;

-- Retrieves employees with salaries between $50,000 and $80,000.

-- IN Operator

SELECT \* FROM employees WHERE department\_id IN (1, 2, 3);

-- Retrieves employees from departments with IDs 1, 2, or 3.

-- NOT Operator

SELECT \* FROM employees WHERE department\_id NOT IN (4, 5);

-- Retrieves employees not belonging to departments with IDs 4 or 5.

-- COUNT Function

SELECT COUNT(\*) FROM employees;

-- Counts the total number of records in the employees table.

-- MAX and MIN Functions

SELECT MAX(salary) AS max\_salary, MIN(salary) AS min\_salary FROM employees;

-- Finds the maximum and minimum salaries from the employees table.

-- AVG Function

SELECT AVG(salary) AS avg\_salary FROM employees;

-- Calculates the average salary of employees.

-- SUM Function

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

-- Calculates the total salary of all employees.

-- GROUP BY with HAVING

SELECT department\_id, COUNT(\*) as employee\_count

FROM employees

GROUP BY department\_id

HAVING COUNT(\*) > 10;

-- Groups employees by department and filters departments with more than 10 employees.

-- JOIN with Multiple Tables

SELECT employees.first\_name, departments.department\_name, locations.city

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id

INNER JOIN locations ON departments.location\_id = locations.location\_id;

-- Joins employees, departments, and locations tables to retrieve employee names, department names, and city locations.

-- UNION Operator

SELECT first\_name, last\_name FROM employees

UNION

SELECT first\_name, last\_name FROM managers;

-- Combines unique records from the employees and managers tables.

-- EXISTS Operator

SELECT first\_name, last\_name

FROM employees e

WHERE EXISTS (

    SELECT 1 FROM managers m WHERE e.employee\_id = m.employee\_id

);

-- Retrieves employees who are also managers based on the existence of matching IDs in the managers table.

-- CASE Statement

SELECT first\_name,

       CASE

           WHEN salary > 70000 THEN 'High'

           WHEN salary BETWEEN 50000 AND 70000 THEN 'Medium'

           ELSE 'Low'

       END AS salary\_category

FROM employees;

-- Categorizes employees based on salary ranges as High, Medium, or Low.

4. What Is The SQL Query Used To Find The 2nd /

3rd / nth Highest Salary

select salary from employee

order by salary limit 2 offset 1;

5. What Is The SQL Query Used To Find All Employees

Who Also Hold The Managerial Position?

select e1.name as employeename, e2.name as managername

from employee e1

join employee e2 on e2.id = e1.manager\_id;

**Inner join**

## SQL INNER JOIN Syntax

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

Ex-1)

SELECT ProductID, ProductName, CategoryName  
FROM Products  
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;

|  |  |  |
| --- | --- | --- |
| **ProductID** | **ProductName** | **CategoryName** |
| 39 | Chartreuse verte | Beverages |
| 2 | Chang | Beverages |
| 24 | Guaraná Fantástica | Beverages |

SELECT \* FROM Employee

INNER JOIN Employee\_new ON Employee.ID = Employee\_new.ID;

6. What Is The SQL Query Used To Find The Names Of

The Employees That Begin With ‘E’ & ‘J’ ?

select name from employee

where name like 'E%' or name like 'J%';

*To Fetch The Even Number Records:-*

**SELECT \* FROM Employee WHERE id%2=0;**