

# **Subquery Basics :**

### **Fetching the Highest Scorer from students table**

Objective: You have a student table , you need to Find the student details who got highest score.

Student Table:

| **id** | **name** | **score** |
| --- | --- | --- |
| 1 | Alice | 95 |
| 2 | Bob | 87 |
| 3 | Charlie | 78 |

Step 1: Find the highest score.

SELECT MAX(score) FROM students;

Output : 9

Step 2: Now we know the highest score , with that we can get students data (hardcoding values)

SELECT \* FROM students WHERE score = 95;

Output :

| **id** | **name** | **score** |
| --- | --- | --- |
| 1 | Alice | 95 |

Step 3: We can use subquery to make it dynamic

SELECT \* FROM students WHERE score = (SELECT MAX(score) FROM students);

Output :

| **id** | **name** | **score** |
| --- | --- | --- |
| 1 | Alice | 95 |

### **Find students who score more than Average Score of the batch**

Sample Table :

| **id** | **name** | **score** |
| --- | --- | --- |
| 1 | Alice | 95 |
| 2 | Bob | 87 |
| 3 | Charlie | 78 |

Objective: Identify students scoring above the class average using subqueries.

SELECT \* FROM students WHERE score > (SELECT AVG(score) FROM students);

## **Conclusion**

* SQL subqueries allow for more dynamic and flexible SQL queries by using the result of one query as the input for another.
* They allow SQL statements to be more flexible and powerful, enabling sophisticated data analysis without hardcoding values.

# **Defination and Syntax of Subquery :**

### **Definition of Subqueries**

A subquery is a query within another query, allowing you to nest SQL queries. It's used to perform operations that usually involve multiple queries in a single, more complex query.

### **Syntax Overview**

Subqueries can appear in various parts of a query, including the SELECT, FROM, WHERE, and HAVING clauses. The basic syntax with WHERE is as follows: Example :

SELECT \* FROM students WHERE score = (SELECT MAX(score) FROM students);

Syntax :

SELECT column\_name(s)

FROM table\_name

WHERE column\_name OPERATOR (SELECT column\_name FROM table\_name);

#### **How Subquery executes under the hood**

* When you execute a query with a subquery, MySQL first executes the inner subquery. This inner query generates a result set or a value that is then used by the outer query to complete its execution.
* First, it calculates the maximum score from the students table by executing the subquery SELECT MAX(score) FROM students. This finds the highest score across all entries.
* The MAX function identifies the largest score value by scanning the score column.
* The main query then filters the students table to retrieve rows where the score matches this maximum value determined by the subquery.
* The query returns all columns for these rows, capturing every student with the highest score. If there are multiple top-scoring students, all are included in the output.

# **Examples of Subquery :**

### **Subquery in SELECT Clause**

A subquery in the SELECT clause can be used to return additional, calculated columns.

Problem Statement : Retrieve a list of employees from a database, including each employee's ID, first name, salary, and department, along with the average salary for each employee's department

SELECT

e.id,

e.first\_name,

e.salary,

e.department,

(

SELECT AVG(salary)

FROM emp

WHERE department = e.department

) AS average\_department\_salary

FROM

emp e;

### **Subquery in WHERE Clause**

Subqueries within the WHERE clause can filter rows based on a condition that involves a calculation or comparison against a set of values returned by the subquery.

Problem Statement : Finding employees who earn more than the average salary in their department:

SELECT

id,

first\_name,

salary

FROM

emp

WHERE

salary > (SELECT AVG(salary) FROM emp WHERE department = emp.department);

Problem Statement : Finding the maximum average salary among all departments:

SELECT

id,

first\_name,

salary

FROM

emp

WHERE

salary > (SELECT AVG(salary) FROM emp WHERE department = emp.department);

### **Subquery in HAVING Clause**

The HAVING clause is used with the GROUP BY statement to filter groups or aggregates. A subquery within HAVING allows complex aggregate filtering.

Problem Statement : Finding departments whose average salary is above the overall average:

SELECT

department,

AVG(salary) AS average\_salary

FROM

emp

GROUP BY

department

HAVING

AVG(salary) > (SELECT AVG(salary) FROM emp);

### **Sample Data for the avove problems**

[Emp Data Link](https://drive.google.com/file/d/1ukkmk4RcOuu_epjJrQLihbvAK7W3c6RZ/view?usp=sharing)

# **View in SQL**

### **Description :**

* A view in SQL is a virtual table based on the result-set of a SQL query.
* It consists of a set of rows and columns, just like a real database table.
* The fields in a view are fields from one or more real tables in the database.
* You define a view with a SQL query that selects fields from one or more tables.

### **Syntax for creating a View :**

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

### **Droping a view :**

DROP VIEW view\_name;

### **Example :**

#### **Craete a view for all the female employee**

CREATE VIEW female\_emp

AS

SELECT \* FROM emp

WHERE gender = "Female";

SELECT \* FROM female\_emp;

#### **Create a view for High salary employee**

CREATE OR REPLACE VIEW female\_high\_salary\_emp

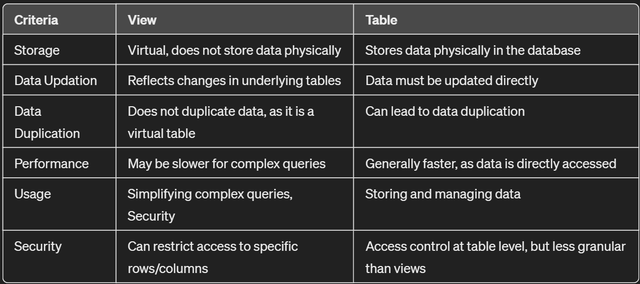
AS

SELECT name , email FROM emp

WHERE gender = "Female" AND salary > 60000;

SELECT \* FROM female\_high\_salary\_emp;

### **Difference between Table and Views in SQL :**

****

### **Advantages of Views**

* Simplification of Complex Queries: Views can encapsulate complex queries, simplifying the interaction with the database by hiding the complexity from users.
* Security: Views can provide a level of security by restricting access to a predetermined set of rows and columns, thereby preventing users from seeing certain data.
* Logical Data Independence: Views can serve as a layer of abstraction over table structures, allowing underlying tables to be modified without affecting access to the data through the view.
* Data Integrity: By using views, you can enforce constraints on data entry, ensuring data integrity. This is particularly useful in complex databases where data consistency is crucial.
* Reusability: Once a view is created, it can be used in any number of SQL queries, enhancing code reusability and consistency.

# **Common Table Expression (CTE) in SQL**

### **Descriptions :**

* Common Table Expressions (CTE) in SQL are a powerful feature that allow you to create temporary result sets which can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement.
* CTEs can make your queries more readable and maintainable by breaking them down into simpler, modular components.
* They are especially useful for recursive queries, which are queries that refer to themselves.

### **Syntax of CTE :**

* WITH cte\_name: Introduces the CTE, assigning a name (e.g., recent\_orders).
* column1, column2, ...: Optional column names for the CTE result set.
* SELECT: Clause defining the data to be included in the CTE.
* FROM: Clause referencing tables involved in the CTE.
* WHERE (optional): Clause filtering data for the CTE.
* Main SELECT: The main query that utilizes the CTE by name (cte\_name).

WITH cte\_name (column1, column2, ...) AS (

SELECT expression1, expression2, ...

FROM table\_name

[WHERE condition]

)

SELECT \*

FROM cte\_name;

### **Structure of a CTE**

A CTE consists of two main parts:

* WITH Clause: This clause introduces the CTE by specifying a name and a subquery defining the result set.
* Main Query: This is the main SQL statement that references the CTE by its name and uses the data from the CTE's result set.

### **Example :**

#### **Give all the products which use "Nvidia" gpu in it.**

WITH nvidia\_gpu\_products AS (

SELECT Company , Product , Cpu , Ram , Memory , Gpu

FROM laptop

WHERE LOWER(Gpu) LIKE "%nvidia%"

)

SELECT Company , Product FROM nvidia\_gpu\_products;

#### **Give count of products for each company wwhich have "nvidia" Gpu in it**

WITH nvidia\_gpu\_products (Company , count\_of\_products) AS (

SELECT Company , COUNT(\*) AS count\_of\_products

FROM laptop

WHERE LOWER(Gpu) LIKE "%nvidia%"

GROUP BY Company

ORDER BY count\_of\_products

)

SELECT \* FROM nvidia\_gpu\_products;

### **Advantage of CTE :**

* Improved Readability: By breaking down complex logic into smaller, named CTEs, the overall query becomes easier to understand and maintain.
* Modularization: CTEs allow you to reuse complex calculations or transformations throughout the query without repetitive subqueries.
* Hierarchical Data Processing: Recursive CTEs can be used to process hierarchical data structures, enabling tasks like traversing tree-like relationships.

## **Resources - Official Documentation and Other Resources**

* SubQueries in SQL - <https://www.javatpoint.com/dbms-sql-sub-queries>
* Views in SQL - <https://www.w3schools.com/sql/sql_view.asp>
* Common Table Expression in MySQL - <https://dev.mysql.com/doc/refman/8.0/en/with.html>

--

USE b35\_db101;

SHOW DATABASES;

SELECT \* FROM emp;

-- Give the highest salary

SELECT MAX(salary) FROM emp;

-- Give the emp details for highest salary emp

SELECT \* FROM emp WHERE salary = 99967;

-- Give the emp details for highest salary emp (Dynamic)

SELECT \* FROM emp WHERE salary = (SELECT MAX(salary) FROM emp);

-- Find avg salary

SELECT AVG(salary) FROM emp;

-- Find all the emp whose salary is greater than average salary

SELECT \* FROM emp WHERE salary > ( SELECT AVG(salary) FROM emp);

-- How sub query Executes under the hood ?

USE b35\_db101;

SELECT \* FROM emp;

-- Subquery with SELECT

-- id , name , salary , department , average\_salary

SELECT id , name , salary , department , (SELECT AVG(salary) FROM emp) AS avg\_salary

FROM emp;

-- id , name , salary , department , average\_salary fr the respective department

SELECT id , name , salary , department , (SELECT AVG(salary) FROM emp WHERE department = e.department) AS avg\_salary

FROM emp AS e;

-- Subquery with where Clause

-- Find all the emp whose salary is greater than average salary

SELECT \* FROM emp WHERE salary > ( SELECT AVG(salary) FROM emp);

-- Finding all the emp where salary is greater than average salary of the respective department.

SELECT \* FROM emp WHERE salary > ( SELECT AVG(salary) FROM emp WHERE department = emp.department);

-- Sub query with having

-- Find those department where avg salary for the department is greater than the avg salary fo all the emp

SELECT department , AVG(salary)

FROM emp

GROUP BY department ;

SELECT department , AVG(salary)

FROM emp

GROUP BY department

HAVING AVG(salary) > (SELECT AVG(salary) FROM emp);

USE b35\_db101;

-- SQL View

-- Syntax

-- CREATE VIEW view\_name

-- Your\_Sweet\_Query;

-- Craete a view for all the female employee with the view name - female\_emp

CREATE VIEW female\_emp

AS

SELECT \* FROM emp

WHERE gender = "Female";

-- Get all the data from - female\_emp

SELECT \* FROM female\_emp;

-- Give the average salary ffrom female\_emp

SELECT AVG(salary) FROM female\_emp;

-- Give all department and total salary for the same

SELECT department , SUM(salary)

FROM female\_emp

GROUP BY department;

-- How to delate a view

DROP VIEW female\_emp;

-- Create a view for high salary feamle emp (salary > 60000)

CREATE OR REPLACE VIEW female\_high\_salary\_emp

AS

SELECT name , email FROM emp

WHERE gender = "Female" AND salary > 60000;

SELECT \* FROM female\_high\_salary\_emp;

DROP VIEW female\_high\_salary\_emp;

USE b35\_db101;

-- Syntax

-- WITH cte\_name AS (

-- SQL\_SUERY

-- )

-- SQL\_QUERY\_WITH\_cte\_name

-- Give all the emp where name starts with "a" or "A" using CTE

SELECT \* FROM emp;

WITH cte AS (

SELECT \* FROM emp

WHERE name LIKE "a%"

)

SELECT \* FROM cte;

-- Calculate avg salary of the emp

WITH cte AS (

SELECT \* FROM emp

WHERE name LIKE "a%"

)

SELECT AVG(salary) FROM cte;

--

SHOW TABLES;

SELECT \* FROM laptop;

-- Give all the products which use "Nvidia" gpu in it.

WITH nvidia\_gpu\_products AS (

SELECT Company , Product , Cpu , Ram , Memory , Gpu

FROM laptop

WHERE LOWER(Gpu) LIKE "%nvidia%"

)

SELECT Company , Product FROM nvidia\_gpu\_products;

-- Give count of products for each company which have "nvidia" Gpu in it

WITH nvidia\_gpu\_products AS (

SELECT Company , Product , Cpu , Ram , Memory , Gpu

FROM laptop

WHERE LOWER(Gpu) LIKE "%nvidia%"

)

SELECT Company , COUNT(\*)

FROM nvidia\_gpu\_products

GROUP BY Company

ORDER BY COUNT(\*);

WITH nvidia\_gpu\_products (Company , count\_of\_products) AS (

SELECT Company , COUNT(\*) AS count\_of\_products

FROM laptop

WHERE LOWER(Gpu) LIKE "%nvidia%"

GROUP BY Company

ORDER BY count\_of\_products

)

SELECT \* FROM nvidia\_gpu\_products;