**Hexaware Technical Training Program**

**Day 7 - March 19 , 2025**

**MySQL**

**SQL Joins**

SQL joins are used to combine rows from two or more tables based on a related column.

**Types of Joins**

| **Join Type** | **Description** |
| --- | --- |
| **INNER JOIN** | Returns only matching rows between tables. |
| **LEFT JOIN (LEFT OUTER JOIN)** | Returns all rows from the left table and matching rows from the right table. |
| **RIGHT JOIN (RIGHT OUTER JOIN)** | Returns all rows from the right table and matching rows from the left table. |
| **FULL JOIN (FULL OUTER JOIN)** | Returns all rows when there is a match in either table. |
| **SELF JOIN** | Joins a table with itself. |
| **CROSS JOIN** | Returns the Cartesian product of two tables. |

**1. INNER JOIN (Returns only matching rows)**

SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName

FROM Employees

INNER JOIN Departments

ON Employees.DepartmentID = Departments.DepartmentID;

* Returns employees who are assigned to a department.

**2. LEFT JOIN (Returns all rows from the left table and matching rows from the right table)**

SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName

FROM Employees

LEFT JOIN Departments

ON Employees.DepartmentID = Departments.DepartmentID;

* Returns all employees, even if they don’t belong to any department (NULL for unmatched departments).

**3. RIGHT JOIN (Returns all rows from the right table and matching rows from the left table)**

SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName

FROM Employees RIGHT JOIN Departments

ON Employees.DepartmentID = Departments.DepartmentID;

* Returns all departments, even if they don’t have employees (NULL for unmatched employees).

**4. FULL JOIN (Returns all rows when there is a match in either table)**

SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName

FROM Employees

FULL OUTER JOIN Departments

ON Employees.DepartmentID = Departments.DepartmentID;

* Returns all employees and all departments, even if they don’t have a match.

**5. SELF JOIN (Joins a table with itself)**

SELECT A.EmployeeID, A.Name AS Employee, B.Name AS Manager

FROM Employees A

JOIN Employees B

ON A.ManagerID = B.EmployeeID;

* Retrieves employees and their managers from the same table.

**6. CROSS JOIN (Returns all possible combinations)**

SELECT Employees.Name, Departments.DepartmentName

FROM Employees

CROSS JOIN Departments;

* Returns every employee paired with every department (Cartesian product).

**Subqueries in MySQL**

A **subquery** is a query inside another query. It is used to fetch data that will be used by the main query.

**Types of Subqueries**

1. **Single-Row Subquery** (Returns one value)
2. **Multi-Row Subquery** (Returns multiple values)
3. **Correlated Subquery** (Depends on the outer query)

**1. Single-Row Subquery Example**

**Find employees who earn more than the average salary**

**SELECT Name, Salary**

**FROM Employees**

**WHERE Salary > (SELECT AVG(Salary) FROM Employees);**

The **subquery calculates the average salary**, and the outer query **selects employees who earn more than that value**.

**2. Multi-Row Subquery Example**

**Find employees who work in departments that have more than 2 employees**

**SELECT Name, DepartmentID**

**FROM Employees**

**WHERE DepartmentID IN (**

**SELECT DepartmentID**

**FROM Employees**

**GROUP BY DepartmentID**

**HAVING COUNT(\*) > 2**

**);**

* The **subquery finds departments with more than 2 employees**.
* The **outer query retrieves employees from those departments**.

**3. Correlated Subquery Example**

**Find employees who earn the highest salary in their department**

**SELECT Name, Salary, DepartmentID**

**FROM Employees e1**

**WHERE Salary = (**

**SELECT MAX(Salary)**

**FROM Employees e2**

**WHERE e1.DepartmentID = e2.DepartmentID**

**);**

* The **subquery runs for each employee**, finding the **maximum salary in their department**.

**4. Using Subquery in FROM Clause (Derived Table)**

**Find the department with the highest total salary expense**

**SELECT DepartmentID, TotalSalary**

**FROM (**

**SELECT DepartmentID, SUM(Salary) AS TotalSalary**

**FROM Employees**

**GROUP BY DepartmentID**

**) AS SalaryByDept**

**ORDER BY TotalSalary DESC**

**LIMIT 1;**

* The **inner query calculates total salary per department**.
* The **outer query fetches the department with the highest total salary**.

**5. Subquery in UPDATE Statement**

**Increase the salary of employees who earn less than the department average by 10%**

**UPDATE Employees e**

**SET Salary = Salary \* 1.10**

**WHERE Salary < (**

**SELECT AVG(Salary)**

**FROM Employees**

**WHERE DepartmentID = e.DepartmentID**

**);**

* The **subquery finds the department-wise average salary**.

**Types of Subqueries in SQL**

Subqueries in SQL can be categorized based on the type of output they return. The four main types are:

1. **Scalar Subquery** (Returns a single value)
2. **Row Subquery** (Returns a single row)
3. **Column Subquery** (Returns a single column)
4. **Table Subquery** (Returns multiple rows and columns)

**1. Scalar Subquery (Returns a Single Value)**

A **scalar subquery** returns only **one value (one row, one column)** and can be used where a single value is expected (e.g., in SELECT, WHERE, HAVING).

**Example: Find employees who earn more than the average salary**

**SELECT Name, Salary**

**FROM Employees**

**WHERE Salary > (SELECT AVG(Salary) FROM Employees);**

* The **subquery calculates the average salary** and returns a **single value**.
* The **outer query retrieves employees whose salary is above the average**.

**Usage:** SELECT, WHERE, HAVING, CASE WHEN, ORDER BY

**2. Row Subquery (Returns a Single Row)**

A **row subquery** returns **one row with multiple columns**.

**Example: Find the employee with the highest salary**

**SELECT Name, Salary**

**FROM Employees**

**WHERE (Salary, EmployeeID) = (**

**SELECT MAX(Salary), EmployeeID**

**FROM Employees**

**);**

* The **subquery returns a row with MAX(Salary) and the corresponding EmployeeID**.
* The **outer query selects the employee with the highest salary**.

**Usage:** In WHERE or FROM clauses when the condition involves multiple columns.

**3. Column Subquery (Returns a Single Column with Multiple Rows)**

A **column subquery** returns **a single column but multiple rows**, often used with IN or ANY.

**Example: Find employees who work in departments that have more than 2 employees**

**SELECT Name, DepartmentID**

**FROM Employees**

**WHERE DepartmentID IN (**

**SELECT DepartmentID**

**FROM Employees**

**GROUP BY DepartmentID**

**HAVING COUNT(\*) > 2**

**);**

* The **subquery returns a list of DepartmentIDs with more than 2 employees**.
* The **outer query fetches employees working in those departments**.

**Usage:** In WHERE with IN, ANY, ALL.

**4. Table Subquery (Returns Multiple Rows and Columns)**

A **table subquery** returns multiple rows and columns and is used in the FROM clause.

**Example: Find the department with the highest total salary expense**

**SELECT DepartmentID, TotalSalary**

**FROM (**

**SELECT DepartmentID, SUM(Salary) AS TotalSalary**

**FROM Employees**

**GROUP BY DepartmentID**

**) AS SalaryByDept**

**ORDER BY TotalSalary DESC**

**LIMIT 1;**

* The **subquery calculates total salary per department** (DepartmentID, TotalSalary).
* The **outer query selects the department with the highest total salary**.

**Usage:** FROM clause, also known as a **Derived Table**.

**Summary Table**

| **Type** | **Returns** | **Common Use Cases** |
| --- | --- | --- |
| **Scalar Subquery** | Single value (1 row, 1 column) | WHERE, HAVING, SELECT, ORDER BY, CASE WHEN |
| **Row Subquery** | Single row, multiple columns | WHERE (with multiple column conditions) |
| **Column Subquery** | Single column, multiple rows | IN, ANY, ALL in WHERE |
| **Table Subquery** | Multiple rows and columns | Used in FROM as a derived table |