Sub Queries and Groups of Data

LAB MANUAL 04

Group By Statement:

The GROUP BY statement group's rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
GROUP BY column_name(s)
```

Group By:

```
SELECT
   AVG(salary) average_salary
FROM
   employees
GROUP BY department_id
```

Group by (Having)

HAVING Clause is used with GROUP BY Clause to restrict the groups of returned rows where condition is TRUE.

Syntax:

```
1. SELECT expression1, expression2, ... expression_n,
2.  aggregate_function (aggregate_expression)
3. FROM tables
4. WHERE conditions
5. GROUP BY expression1, expression2, ... expression_n
6. HAVING having_condition;
```

```
1. SELECT item, SUM(sale) AS "Total sales"
2. FROM salesdepartment
3. GROUP BY item
4. HAVING SUM(sale) < 1000;
```

Sub Queries:

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

NOTE:

Subqueries are useful when a query is based on unknown values.

Sub Queries with SELECT Statement:

Syntax:

```
1. SELECT column_name
2. FROM table_name
3. WHERE column_name expression operator
4. ( SELECT column_name  from table_name WHERE ... );
```

Types of Subqueries:

Single Row Sub Query: Sub query which returns single row output. They mark the usage of single row comparison operators, when used in WHERE conditions.

Multiple row sub query: Sub query returning multiple row output. They make use of multiple row comparison operators like IN, ANY, ALL. There can be sub queries returning multiple columns also.

Correlated Sub Query: Correlated subqueries depend on data provided by the outer query. This type of subquery also includes subqueries that use the EXISTS operator to test the existence of data rows satisfying specified criteria.

Single Row Sub Queries:

•Return only one row

•Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<> , =!	Not equal to

```
SELECT ename, job FROM EMP WHERE job = ( SELECT job FROM emp
WHERE empno=7369 )
```

Single Row Functions:

Finds the employees who have the highest salary:

```
SELECT
   employee_id, first_name, last_name, salary
FROM
   employees
WHERE
   salary = (SELECT
         MAX(salary)
      FROM
         employees)
```

Finds all employees who salaries are greater than the average salary of all employees:

```
SELECT
   employee_id, first_name, last_name, salary
FROM
   employees
WHERE
   salary > (SELECT
         AVG(salary)
      FROM
         employees)
```

Multiple row sub query:

Return more than one row

• Use multiple-row comparison operators

- [> ALL] More than the highest value returned by the subquery
- [< ALL] Less than the lowest value returned by the subquery
- [< ANY] Less than the highest value returned by the subquery
- [> ANY] More than the lowest value returned by the subquery
- [= ANY] Equal to any value returned by the subquery (same as IN)

IN:

```
SELECT          first_name,department_id
FROM employees
WHERE department_id IN (SELECT department_id
            FROM departments
            WHERE LOCATION_ID =100)
```

ANY:

```
SELECT empno, ename, job FROM emp WHERE sal< ANY
 ( SELECT sal FROM emp WHERE job = 'CLERK' );
```

```
SELECT empno, ename, job FROM emp WHERE sal< ANY
 ( SELECT sal FROM emp WHERE job = 'CLERK' ) AND job <> 'CLERK' ;
```

ALL:

```
SELECT empno, ename, job FROM emp WHERE sal> ALL
 ( SELECT sal FROM emp WHERE job = 'CLERK' ) AND job <> 'CLERK' ;
```

HAVING Example: (with GROUP BY MIN function)

```
1. SELECT department,
2. MIN(salary) AS "Lowest salary"
3. FROM employees
4. GROUP BY department
5. HAVING MIN(salary) < 15000;
```

HAVING Example: (with GROUP BY MAX function)

```
1. SELECT department,
2. MAX(salary) AS "Highest salary"
3. FROM employees
4. GROUP BY department
5. HAVING MAX(salary) > 30000;
```

Group By and HAVING IN SUB QUERIES:

```
SELECT department_name, avg(salary)
FROM emp_details_view
GROUP BY department_name
HAVING avg(salary) > (
        SELECT avg(salary)
        FROM employees
);
```

SUBQUERIES AND DML:

Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
1. INSERT INTO table_name (column1, column2, column3....)
2. SELECT *
3. FROM table_name
4. WHERE VALUE OPERATOR
```

Example:

```
1. INSERT INTO EMPLOYEE_BKP
2.   SELECT * FROM EMPLOYEE
3.   WHERE ID IN (SELECT ID
4.   FROM EMPLOYEE);
```

Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

Example:

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
UPDATE EMPLOYEE
  SET SALARY = SALARY * 0.25
  WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
    WHERE AGE >= 29);
```

Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

```
DELETE FROM TABLE_NAME
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

Example:

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

```
1. DELETE FROM EMPLOYEE
2.   WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
3.     WHERE AGE >= 29 );
```

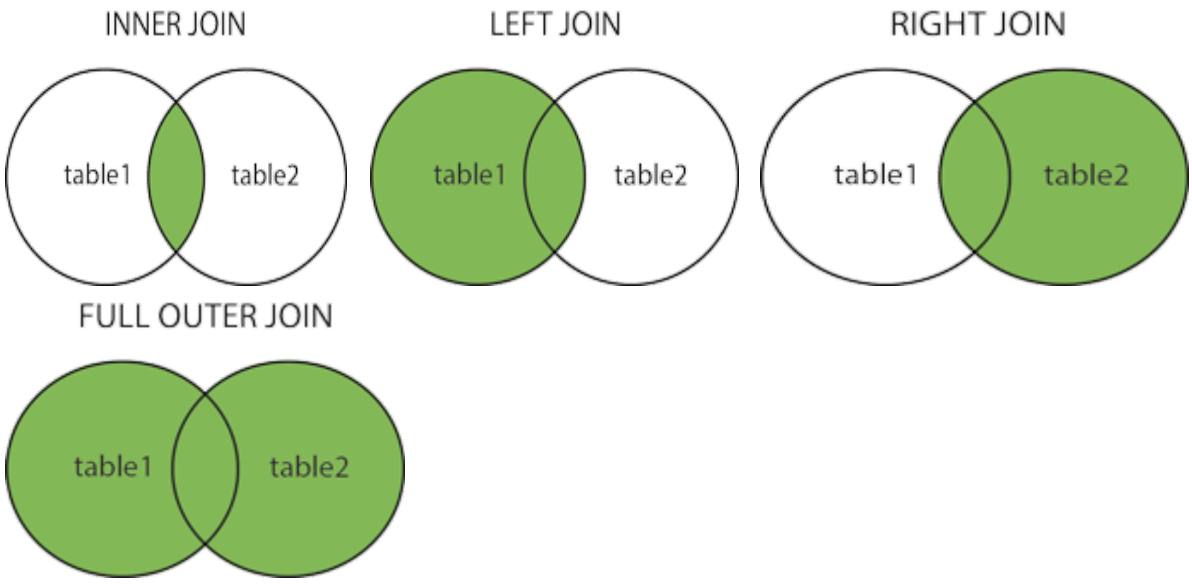
SQL JOIN:

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables.
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT
   first_name,
   last_name,
   employees.department_id,
   departments.department_id,
   department_name
FROM
   employees
      INNER JOIN
   departments ON departments.department_id = employees.department_id
```

SQL LEFT JOIN Keyword

A LEFT JOIN statement returns all rows from the left table along with the rows from the right table for which the join condition is met.

```
1. SELECT t1.emp_id, t1.emp_name, t1.hire_date, t2.dept_name
2. FROM employees AS t1 LEFT JOIN departments AS t2
3. ON t1.dept_id = t2.dept_id ORDER BY emp_id;
```

SQL RIGHT JOIN Keyword

The RIGHT JOIN is the exact opposite of the LEFT JOIN. It returns all rows from the right table along with the rows from the left table for which the join condition is met.

```
1. SELECT t1.emp_id, t1.emp_name, t1.hire_date, t2.dept_name
2. FROM employees AS t1 RIGHT JOIN departments AS t2
3. ON t1.dept_id = t2.dept_id ORDER BY dept_name;
```

Full Joins

A FULL JOIN returns all the rows from the joined tables, whether they are matched or not i.e. you can say a full join combines the functions of a LEFT JOIN and a RIGHT JOIN. Full join is a type of outer jointhat's why it is also referred as *full outer join*.

```
1. SELECT t1.emp_id, t1.emp_name, t1.hire_date, t2.dept_name
2. FROM employees AS t1 FULL JOIN departments AS t2
3. ON t1.dept_id = t2.dept_id ORDER BY emp_name;
```