**USMAN INSTITUTE OF TECHNOLOGY**

**Department of Computer Science**

**CS311 Introduction to Database Systems**

Lab#7

**Objective:**

**-** **Subqueries and compound queries in SQL**

**Name of Student: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Sec. \_\_\_\_\_\_\_\_\_\_\_**

**Date of Experiment: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Marks Obtained/Remarks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Why use subqueries?**

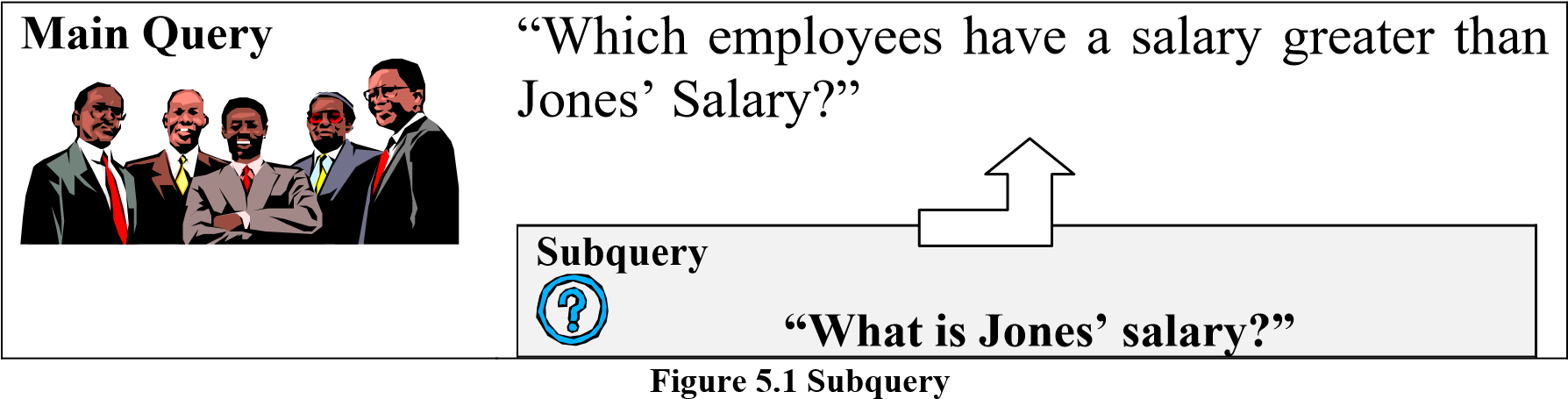
S

uppose we want to write a query to find out who earns a salary greater than Jones’ salary.

To solve this problem, we need two queries: one query to find what Jones earns and a second query to find who earns more than that amount.

The above problem can be solved by combining the two queries, placing one query inside the other query.

The inner query or the *subquery* returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the *search value* in the second query.



A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. They can be very useful when we need to select rows from a table with a condition that depends on the data in the table itself. The subquery generally executes first and its output is used to complete the query condition for the main or outer query.

The subquery can be placed in a number of SQL clauses:

* WHERE clause
* HAVING clause
* FROM clause

The syntax of SELECT statement using subqueries is

SELECT *select\_list*

FROM *table*

WHERE *expr operator*

(SELECT *select\_list*

FROM *table*);

**Note**: In the syntax, operator means comparison operator. Comparison operators fall into two clauses: single-row operators (>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL).

For example, to display the names of all employees who earn more than employee with number 7566.

SELECT ename

FROM emp

WHERE sal >

(SELECT sal

FROM emp

WHERE empno = 7566);

**Types of Subqueries**

**Single-row subquery**: Query that returns only one row from the inner SELECT statement. **Multiple-row subquery**: Query that returns more than one row form the inner SELECT statement.

**Multiple-column subquery**: Query that returns more than one column from the inner SELECT statement.

**Single-Row Subqueries Examples**

1. To display the employees whose job title is the same as that of employee 7369.

SELECT ename, job FROM emp

WHERE job =

(SELECT job

FROM emp

WHERE empno = 7369);

1. To display employees whose job title is the same as that of employee 7369 and whose salary is greater than that of employee 7876.

SELECT ename, job

FROM emp

WHERE job =

(SELECT job

FROM emp

WHERE empno = 7369)

AND sal >

(SELECT sal

FROM emp

WHERE empno = 7876);

1. We can display data from a main query by using a group function in a subquery to return a single row. e.g. to display the employee name, job title and salary of all employees whose salary is equal to the minimum salary.

SELECT ename, job, sal

FROM emp

WHERE sal =

(SELECT MIN(sal) FROM emp);

1. We can use subqueries not only in the WHERE clause, but also in the HAVING clause. The Oracle server executes the subquery and the results are returned into the HAVING clause of the main query. E.g. to display all departments that have a minimum salary greater than that of department 20.

SELECT deptno, MIN(sal)

FROM emp

GROUP BY deptno

HAVING MIN(sal) >

(SELECT MIN(sal)

FROM emp

WHERE deptno = 20);

**Multiple-Row Subqueries**

Multiple-row subqueries return more than one row. We use multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values. Following table illustrates multiple row operators.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| IN | Equal to any member in the list |
| ANY | Compare value to each value returned by the subquery |
| ALL | Compare value to every value returned by the subquery |

**Table 5.1**

**Note**: The **NOT** operator can be used with IN, ANY, and ALL operators.

**Examples**

1. Find the employees who earn the same salary as the minimum salary for departments.

SELECT ename, sal, deptno

FROM emp

WHERE sal IN (SELECT MIN(sal)

FROM emp

GROUP BY deptno);

1. To display employees whose salary is less than any clerk and who are not clerks.

SELECT empno, ename, job

FROM emp

WHERE sal < ANY

(SELECT sal

FROM emp

WHERE job = ‘CLERK’)

AND JOB <> ‘CLERK’;

1. To display employees whose salary is greater than the average salary of all the departments.

SELECT empno, ename, job

FROM emp

WHERE sal > ALL

(SELECT avg(sal)

FROM emp

GROUP BY deptno);

**Multiple-Column Subqueries**

If we want to compare two or more columns, we must write a compound WHERE clause using logical operators. Multiple column subqueries enable us to combine duplicate WHERE conditions into a single WHERE clause.

For example, to display the name of all employees who have done their present job somewhere before in their career.

SELECT ENAME

FROM EMP

WHERE (EMPNO, JOB)

IN

(SELECT EMPNO, JOB

FROM JOB\_HISTORY)

**COMPOUND QUERIES**

In SQL, we can use the normal set operators of Union, Intersection and Set Difference to combine the results of two or more component queries into a single result table. Queries containing SET operators are called *compound* queries. The following table shows the different set operators provided in Oracle SQL.

|  |  |
| --- | --- |
| **Operator** | **Returns** |
| UNION | All distinct rows selected by either query |
| UNION ALL | All rows selected by either query including all duplicates |
| INTERSECT | All distinct rows selected by both queries |
| MINUS | All distinct rows that are selected by the first SELECT statement and that are not selected in the second SELECT statement |

**Table 5.2**

**Restrictions on using set Operators**

There are restrictions on the tables that can be combined using the set operations, the most important one being that the two tables have to be union-compatible; that is, they have the same structure. This implies that the two tables must contain the same number of columns, and that their corresponding columns contain the same data types and lengths. It is the user’s responsibility to ensure that values in corresponding columns come from the same domain. For example, it would not be sensible to combine a column containing the age of staff with the number of rooms in a property, even though both columns may have the same data type i.e. NUMBER.

**The UNION Operator**

The UNION operator returns rows from both queries after eliminating duplicates. By default, the output is sorted in ascending order of the first column of the SELECT clause.

For example to display all the jobs that each employee has performed, the following query will be given. (NOTE: If an employee has performed a job multiple times, it will be shown only once)

SELECT EMPNO, JOB FROM JOB\_HISTORY UNION

SELECT EMPNO, JOB

FROM EMP;

**The UNION ALL Operator**

The UNION ALL operator returns rows from both queries including all duplicates. For example to display the current and previous jobs of all employees, the following query will be given. (NOTE: If an employee has performed a job multiple times, it will be shown separately)

SELECT EMPNO, JOB FROM JOB\_HISTORY UNION ALL

SELECT EMPNO, JOB

FROM EMP;

**The INTERSECT Operator**

The INTERSECT operator returns all rows that are common to both queries. For example, to display all employees and their jobs those have already performed their present job somewhere else in the past.

SELECT EMPNO, JOB FROM JOB\_HISTORY INTERSECT

SELECT EMPNO, JOB

FROM EMP;

**The MINUS Operator**

The MINUS operator returns rows from the first query that is not present in the second query. For example to display the ID of those employees whose present job is the first one in their career.

SELECT EMPNO, JOB

FROM EMP

MINUS

SELECT EMPNO, JOB

FROM JOB\_HISTORY;

**EXERCISE**

1. Why are subqueries needed in accessing data from the database?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write down the restrictions on using set operators in SQL.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write down SQL queries to perform following functions: -

* 1. To display the employee number and name for all employees who earn more than the average salary. Sort the results in descending order of salary.

* 1. To display the employee name and salary of all employees who report to *king*.

* 1. To display the department number, name and job for all employees in the *Sales* department.

* 1. To display the name, hiredate and salary for all employees who have both the same salary and commission as *scott*.

* 1. To display the employee name, department number and job title for all employees whose location is *Dallas*.

* 1. Write a query to display the employee name and hiredate for all employees in the same department as Blake. Exclude Blake.

* 1. Display the employee number, name and salary for all employees who earn more than the average salary and who work in department with any employee with a T in their name.