**National University of Computer & Emerging Sciences, Karachi**
**Computer Science Department**
**Fall 2024, Lab Manual - 04**

| Course Code: CL-2005 | Course: Database Systems Lab |
|---|---|
| Instructor(s): | Sohail Ahmed Malik, Mr. Mubashir, Mr. Sameer Faisal, Ms. Mehak Mazhar, Ms. Fatima, Ms. Mahnoor Javed, Ms. Filza Akhlaq, Ms. Bushra Sattar, Ms. Syeda Ravia Ejaz, Ms. Yumna Asif |

**Contents:**

- Groups of Data (Group by, Having)
- Sub Queries (Single Row, Multiple and correlated)
- Sub Queries and DML
- Tasks

**Group by Statement:**

The GROUP BY statement group's rows that have the same values in 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)
```

```
SELECT
   AVG(salary) as "average_salary"
FROM
   employees
GROUP BY Department_id
```

Sample Output:

| | AVERAGE_SALARY |
|---|---|
| 1 | 8601.333333333333333333333333333333333333 |
| 2 | 4150 |
| 3 | 7000 |
| 4 | 19333.333333333333333333333333333333333333 |
| 5 | 9500 |
| 6 | 10000 |
| 7 | 10154 |
| 8 | 3475.555555555555555555555555555555555556 |
| 9 | 8955.882352941176470588235294117647058824 |
| 10 | 6500 |
| 11 | 5760 |
| 12 | 4400 |

**Group by (Having)**

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

**Syntax:**

```
SELECT expression1, expression2, ... expression_n,
 aggregate_function (aggregate_expression)
FROM table_name
WHERE conditions
```

```
GROUP BY expression1, expression2, ... expression_n
HAVING having_condition;
```

**HAVING Example: (with GROUP BY SUM function)**

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

**HAVING Example: (with GROUP BY MIN function)**

```
SELECT Department_ID,
MIN(salary) AS "Lowest salary"
FROM employees
GROUP BY Department_ID
HAVING MIN(salary) < 15000;
```

**Sample Output:**

| | DEPARTMENT_ID | Lowest salary |
|---|---|---|
| 1 | 100 | 6900 |
| 2 | 30 | 2500 |
| 3 | (null) | 7000 |
| 4 | 20 | 6000 |
| 5 | 70 | 10000 |
| 6 | 110 | 8300 |
| 7 | 50 | 2100 |
| 8 | 80 | 6100 |
| 9 | 40 | 6500 |
| 10 | 60 | 4200 |
| 11 | 10 | 4400 |

**HAVING Example: (with GROUP BY MAX function)**

```
SELECT Department_ID,
MAX(salary) AS "Highest salary"
FROM employees
GROUP BY Department_ID
HAVING MAX(salary) > 3000;
```

**Sample Output:**

| | DEPARTMENT_ID | Highest salary |
|---|---|---|
| 1 | 100 | 12008 |
| 2 | 30 | 11000 |
| 3 | (null) | 7000 |
| 4 | 90 | 24000 |
| 5 | 20 | 13000 |
| 6 | 70 | 10000 |
| 7 | 110 | 12008 |
| 8 | 50 | 8200 |
| 9 | 80 | 14000 |
| 10 | 40 | 6500 |
| 11 | 60 | 9000 |
| 12 | 10 | 4400 |

**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:

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
( SELECT column_name FROM table_name WHERE ... );
```

**Types of Subqueries**:

1. **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.
2. **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.
3. **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 First_Name, Job_ID FROM Employees WHERE job = ( SELECT job_ID FROM
    Employees WHERE empno=7369 )
```

Sample Output:

| | FIRST_NAME | JOB_ID |
|---|---|---|
| 1 | Alexander | PU_CLERK |
| 2 | Shelli | PU_CLERK |
| 3 | Sigal | PU_CLERK |
| 4 | Guy | PU_CLERK |
| 5 | Karen | PU_CLERK |

**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)
```

Sample Output:

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | SALARY |
|---|---|---|---|---|
| 1 | 100 | Steven | King | 24000 |

**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)
```

Sample Output:

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | SALARY |
|---|---|---|---|---|
| 1 | 100 | Steven | King | 24000 |
| 2 | 101 | Neena | Kochhar | 17000 |
| 3 | 102 | Lex | De Haan | 17000 |
| 4 | 103 | Alexander | Hunold | 9000 |
| 5 | 108 | Nancy | Greenberg | 12008 |
| 6 | 109 | Daniel | Faviet | 9000 |
| 7 | 110 | John | Chen | 8200 |
| 8 | 111 | Ismael | Sciarra | 7700 |
| 9 | 112 | Jose Manuel | Urman | 7800 |
| 10 | 113 | Luis | Popp | 6900 |

**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
  - [= A NY] 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)
```

**Sample Output:**

| | FIRST_NAME | DEPARTMENT_ID |
|---|---|---|
| 1 | Shelli | 30 |
| 2 | John | 100 |
| 3 | Karen | 30 |
| 4 | Lex | 90 |
| 5 | Daniel | 100 |
| 6 | William | 110 |
| 7 | Nancy | 100 |
| 8 | Shelley | 110 |
| 9 | Guy | 30 |
| 10 | Alexander | 30 |

**ANY:**

```
SELECT employee_ID, First_Name, job_ID
FROM EMPLOYEES
WHERE SALARY < ANY
 ( SELECT salary FROM  EMPLOYEES WHERE JOB_ID = 'PU_CLERK' );
```

**Sample Output:**

| | EMPLOYEE_ID | FIRST_NAME | JOB_ID |
|---|---|---|---|
| 1 | 132 | TJ | ST_CLERK |
| 2 | 128 | Steven | ST_CLERK |
| 3 | 136 | Hazel | ST_CLERK |
| 4 | 127 | James | ST_CLERK |
| 5 | 135 | Ki | ST_CLERK |
| 6 | 119 | Karen | PU_CLERK |
| 7 | 131 | James | ST_CLERK |
| 8 | 140 | Joshua | ST_CLERK |
| 9 | 144 | Peter | ST_CLERK |
| 10 | 182 | Martha | SH_CLERK |

**ALL:**

```
SELECT employee_ID, First_Name, job_ID
FROM EMPLOYEES WHERE SALARY >All
 ( SELECT salary FROM HR.EMPLOYEES WHERE JOB_ID = 'PU_CLERK' ) AND
    job_ID <> 'PU_CLERK' ;
```

Sample Output:

| | EMPLOYEE_ID | FIRST_NAME | JOB_ID |
|---|---|---|---|
| 1 | 180 | Winston | SH_CLERK |
| 2 | 125 | Julia | ST_CLERK |
| 3 | 194 | Samuel | SH_CLERK |
| 4 | 138 | Stephen | ST_CLERK |
| 5 | 133 | Jason | ST_CLERK |
| 6 | 129 | Laura | ST_CLERK |
| 7 | 186 | Julia | SH_CLERK |
| 8 | 141 | Trenna | ST_CLERK |
| 9 | 189 | Jennifer | SH_CLERK |
| 10 | 137 | Renske | ST_CLERK |

**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);
```

**Sample Output:**

| | DEPARTMENT_NAME | AVG(SALARY) |
|---|---|---|
| 1 | Accounting | 10154 |
| 2 | Executive | 19333.333333333333333333333333333333333 |
| 3 | Human Resources | 6500 |
| 4 | Public Relations | 10000 |
| 5 | Finance | 8601.333333333333333333333333333333333 |
| 6 | Sales | 8955.882352941176470588235294117647058824 |
| 7 | Marketing | 9500 |

**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:**

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

**You may login from a new user for DML sub Queries.**

**Example**: Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table having all the attributes of Employees table

```
INSERT INTO EMPLOYEE_BKP
SELECT * FROM EMPLOYEES
WHERE job_ID IN (SELECT job_id
FROM jobs WHERE job_title='Accountant');
```

**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:**

The given example updates the SALARY by 10 times in the EMPLOYEE table for all employee whose minimum salary is 3000.

```
Update employees
set salary= salary+(0.1*salary)
WHERE job_ID IN (SELECT job_ID
FROM jobs WHERE min_salary=3000);
```

**Subqueries with the DELETE Statement**

The subquery of SQL can be used in conjunction with the Delete statement just like any other statement 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 a backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE_BKP table for all EMPLOYEE whose end date is '31-DEC-06'.

```
Delete from employee_BKP
WHERE job_ID IN (SELECT job_ID
FROM job_History WHERE end_Date='31-Dec-06');
```

```
SELECT
  e.employee_id,
  e.first_name,
  e.last_name,
  (SELECT job_title FROM jobs WHERE job_id = e.job_id) AS job_title,
   (SELECT department_name FROM departments WHERE department_id = e.department_id)
AS department_name,
        (SELECT city FROM locations WHERE location_id = d.location_id) AS
department_location,
  (SELECT region_name FROM regions WHERE region_id = r.region_id) AS region_name
FROM
  employees e,
  departments d,
  locations l,
  regions r
```

```
WHERE
  e.department_id = d.department_id
  AND d.location_id = l.location_id;
```

**(ROWNUM) LIMIT Function:**

In SQL databases, limit function is used to restrict the number of rows returned by a query. Here's a simple explanation of how LIMIT function works:

**Example:**

**Display only the top 5 highest salaries from an employee's table**

```
SELECT salary FROM (
    SELECT salary FROM employees
    ORDER BY salary DESC
    )
    WHERE ROWNUM <= 5;
```

**Lab Activity:**

1. Display the average salary for each job, but only for jobs where the average salary is greater than $10,000. Display the job ID and the average salary.
2. For each department that has more than 2 employees, retrieve the department number and the number of its employees who are making more than $10,000.
3. Display the name of the department that has the most recent job start date. Use ROWNUM to ensure only the most recent record is considered.
4. Create indexes on employee_id in both employees and job_history tables. List employees who do not have any records in the job_history table.
5. Display the department number and the salary of the highest-paid employee in that department. Excluding departments where the maximum salary is below $2,000. Sort the results in descending order of the salary.
6. Display the department IDs and average salaries of employees where the average salary is above $6,000. Use ROWNUM to limit the results to the top 3 departments.
7. Retrieve the last name and job ID of employees who have the same job as the employee with Employee_ID 150.
8. Create table Job_History1 like the job_history table of HR user. Insert records into Job_History1 for jobs with an end date of '19-DEC-07' from hr.Job_History.
9. Insert 5 rows in the Job_History1 table and delete records from Job_History1 where the job_id is ' AC_ACCOUNT'.
10. Delete records from Job_History for departments with the name 'IT'.

**Lab Task:**

11. Display the names and salaries of employees who earn more than the average salary of their respective department.
12. Display the job ID and the salary of the lowest-paid employee in each job. Exclude any jobs where the minimum salary is below $1,000. Sort the results in ascending order of salary.
13. Select first name and department ID of employees working in the same department as the employee with Employee_ID 140.
14. List employees whose job title is the same as that of employee 7369 and whose salary is greater than that of employee 7876.
15. Create a replica of employees table and increase the salary of employees by 12% who have a salary between $5,000 and $10,000.
16. Display the names and salaries of employees whose salaries fall in the top 10% of all salaries. Use ROWNUM to limit the results.
17. Write a Query to display the number of departments with the same location.
18. Display the job ID and the salary of the lowest paid employee of that job. Exclude anyone whose job is not known. Exclude any groups where the minimum salary is 1500. Sort the output in descending order of the salary.

19. Write a Query to select Firstname and Department_ID of Employees who are working in the same department as employee_ID no 130.
20. List all employees who are not working in department 30 and who earn more than all employees working in department 30.
21. Write a query to display the department number, name (department name) and location name for all departments whose average salary is greater than any average salary of those departments whose location name is 'New York'.
22. Insert into employees_BKP as it should copy the record of the employee whose hire date is ’10-MAR-03’ from employees table.