**National University of Computer & Emerging Sciences, Karachi** 

**Computer Science Department**

**Fall 2023, Lab Manual - 04**

|  |  |
| --- | --- |
| **Course Code: CL-2005** | **Course: Database Systems Lab** |
| **Instructor(s):** | **Shakir Hussain** |

# Contents:

1. Groups of Data (Group by, Having)
2. Sub Queries (Single Row, Multiple and correlated)
3. Sub Queries and DML
4. Tasks

**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.

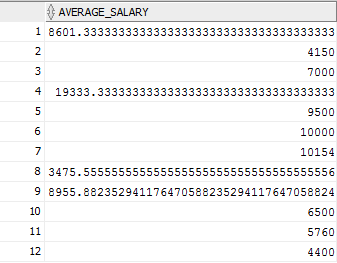
**Group by Syntax**

|  |
| --- |
| SELECT *column\_name(s)* FROM *table\_name* GROUP BY *column\_name(s)* |

**Group By:**

|  |
| --- |
| SELECT      AVG(salary) as “average\_salary”  FROM      employees  GROUP BY Department\_id |

**Sample Output:**

****

**Group by (Having)**

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

**Syntax:**

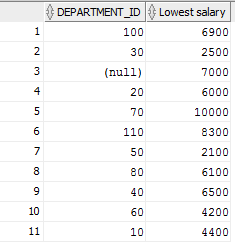
|  |
| --- |
| 1. **SELECT** expression1, expression2, ... expression\_n, 2. aggregate\_function (aggregate\_expression) 3. **FROM** [table](https://www.javatpoint.com/oracle-having-clause)s 4. **WHERE** conditions 5. **GROUP** **BY** expression1, expression2, ... expression\_n 6. **HAVING** having\_condition; |

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

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

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

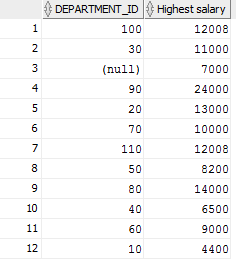
|  |
| --- |
| 1. **SELECT** Department\_ID, 2. **MIN**(salary) **AS** "Lowest salary" 3. **FROM** employees 4. **GROUP** **BY** Department\_ID 5. **HAVING** **MIN**(salary) < 15000; |

**Sample Output:**

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

|  |
| --- |
| 1. **SELECT** Department\_ID, 2. **MAX**(salary) **AS** "Highest salary" 3. **FROM** employees 4. **GROUP** **BY** Department\_ID 5. **HAVING** **MAX**(salary) > 3000; |

**Sample Output:**

****

**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:**

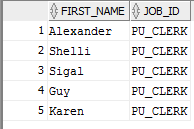
• Return only one row

• 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 First\_Name, Job\_ID FROM Employees WHERE job = ( SELECT job\_ID FROM Employees WHERE empno=7369 ) |

**Sample Output:**

****

**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) |

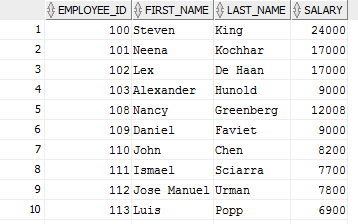
**Sample Output:**



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:**



**Multiple row sub query:**

Return more than one row

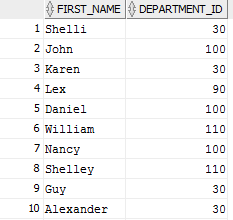
• Use multiple-row comparison operators

* [> 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) |

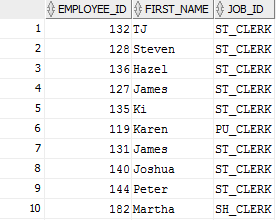
**Sample Output:**

****

**ANY:**

|  |
| --- |
| SELECT employee\_ID, First\_Name, job\_ID FROM EMPLOYEES WHERE SALARY < ANY  ( SELECT salary FROM EMPLOYEES WHERE JOB\_ID = 'PU\_CLERK' ); |

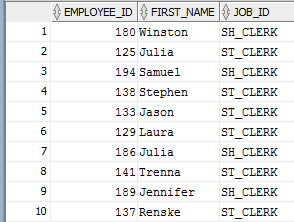
**Sample Output:**



**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:**

****

**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:**

## 

## 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 |

**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**

|  |
| --- |
| 1. UPDATE table 2. SET column\_name = new\_value 3. WHERE VALUE OPERATOR 4. (SELECT COLUMN\_NAME 5. FROM TABLE\_NAME 6. 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 statements mentioned above.

**Syntax**

|  |
| --- |
| 1. DELETE FROM TABLE\_NAME 2. WHERE VALUE OPERATOR 3. (SELECT COLUMN\_NAME 4. FROM TABLE\_NAME 5. 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\_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'); |

**Tasks:**

1. For each department, retrieve the department no, the number of employees in the department and their average salary.
2. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $15,000.
3. Write a Query to display the number of employees with the same job.
4. Display the manager number and the salary of the lowest paid employee of that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is 2000. Sort the output is descending order of the salary.
5. Write a Query to select Firstname and Hiredate of Employees Hired right after the joining of employee\_ID no 110.
6. Write a SQL query to select those departments where maximum salary is at least 15000.
7. Write a query to display the employee number, name (first name and last name) and job title for all employees whose salary is smaller than any salary of those employees whose job title is IT\_PROG.
8. Update salary of employees by 20% increment having minimum salary of 6000.

**Home Tasks**

1. Create a table named **Student** having attributes **name, age, gender, roll\_number, semester, warning, CGPA,** and **Department** (at least 3)**.**
2. Create another table named **Course** having attributes **course\_code, course\_name,** and **credit\_hours**.
3. Make **course\_code** primary key for Course table and **roll\_number** for Student.
4. Add a check on gender to make sure that it’s either Male or Female.
5. Make sure that the **course\_name** is never repeated.
6. Change column name **course\_name** to **course\_title**.
7. Make sure that if no value is assigned to **warning** column, it should be 0.
8. Fill the above table with your own data as the first-row entry. The remaining entries should be random and not include your classmates/friends.
9. Fill the above table with 7-9 Core courses from your CS program and add 3-4 random electives as well (at least 10 rows of data in total).
10. Write a query to retrieve the total number of courses in each department.
11. Write a query to Filter the Students with CGPA of more than 2.5.
12. Write a query to retrieve the bottom 3 students.
13. Retrieve details of the student(s) with the best CGPA.
14. Retrieve details of the student(s) with the worst CGPA.
15. Retrieve details of student(s) that belong to either **CS** or **EE** department (you can’t use **OR** operator).
16. Retrieve details of student(s) that have the same number of warnings as any of students with the worst CGPA.
17. Retrieve details of student(s) that are in the same department as the student with CGPA 3.6. If more than one department has students with 3.6 CGPA, it shouldn’t return anything.
18. Delete records of students whose warning count is less than the average warning count.