|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Database Management System Laboratory** | **Semester:** | **IV** |
| **Date of Performance:** | **22 / 02 / 2023** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Prof. Shila Dhande** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 5**

**Title: Implementation of Database in SQL – DML II**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **Aim: DML – select, insert, update, delete**   1. Group by, Having clause 2. Aggregate functions, Set operations   **Objective:** To perform various DML Operations and executing clauses & set operator’s-based queries. |

|  |
| --- |
| **COs to be achieved:** |
| **CO3:** Use SQL for Relational database creation, maintenance and query processing. |

|  |
| --- |
| **Books / Journals / Websites Referred:** |
| 1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press www.db-book.com 2. Korth, Slberchatz, Sudarshan: “Database Systems Concept”, 6th Edition, McGraw Hill 3. Elmasri and Navathe, “Fundamentals of Database Systems”, 5th Edition, PEARSON Education.   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **PreLab / Prior Concepts:** |

|  |
| --- |
| **Tools Required:** |
| * Postgresql Software |

|  |
| --- |
| **Theory:** |
| **Clauses and Operators:**   1. Order by Clause –   The Order byclause allows you to sort the result set of a query based on one or more columns.  Basic Syntax:   * The ORDER BY clause is used after the SELECT statement to sort query results. - Syntax: SELECT column1, column2 FROM table\_name ORDER BY column1 [ASC|DESC];   Ascending and Descending Order:   * By default, the ORDER BY clause sorts in ascending order (smallest to largest). * You can explicitly specify descending order using the DESC keyword. * Example: SELECT product\_name, price FROM products ORDER BY price DESC;   Sorting by Multiple Columns:   * You can sort by multiple columns by listing them sequentially in the ORDER BY clause. * Rows are first sorted based on the first column, and for rows with equal values, * subsequent columns are used for further sorting. * Example: SELECT first\_name, last\_name FROM employees ORDER BY last\_name first\_name;   Sorting by Expressions:   * It's possible to sort by calculated expressions, not just column values. * Example: SELECT product\_name, price, price \* 1.1 AS discounted\_price FROM * products ORDER BY discounted\_price;   Sorting NULL Values:   * By default, NULL values are considered the smallest in ascending order and the largest in descending order. * You can control the sorting behaviour of NULL values using the NULLS FIRST or NULLS LAST options. * Example: SELECT column\_name FROM table\_name ORDER BY column\_name NULLS LAST;   Sorting by Position:   * Instead of specifying column names, you can sort by column positions in the ORDER BY clause. * Example: SELECT product\_name, price FROM products ORDER BY 2 DESC, 1 ASC;  1. Group by Clause –   These are circumstances where we would like to apply the aggregate functions to a single set of tuples but also to a group of sets of tuples we would like to specify this wish in SQL using the group by clause. The attributes or attributes given by the group by clause are used to form groups. Tuples with the same value on all attributes in the group by clause placed in one group.  Example:  Select<attribute\_name,avg(<attribute\_name>)as  <new\_attribute\_name>l From <table\_name>  Group by <attribute\_name>  Example:  select designation, sum( salary) as total\_salary from employee group by Designation;   1. Having Clause –   A having clause is like a where clause but only applies only to groups as a whole whereas the where clause applies to the individual rows. A query can contain both where clause and a having clause. In that case -   1. The where clause is applied first to the individual rows in the tables or table structures objects in the diagram pane. Only the rows that meet the conditions in the where clause are grouped. 2. The having clause is then applied to the rows in the result set that are produced by grouping. Only the groups that meet the having conditions appear in the query output.   Example:  select dept\_no from EMPLOYEE group\_by dept\_no  having avg (salary) >=all (select avg (salary)  from EMPLOYEE group by dept\_no);  **Aggregate Functions:**  Aggregate functions such as SUM, AVG, count, count (\*), MAX and MIN generate summary values in query result sets. An aggregate functions (with the exception of count (\*) processes all the selected values in a single column to produce a single result value.  Example:  select dept\_no,count (\*)  from EMPLOYEE group by dept\_no;  Example: select max (salary)as maximum from EMPLOYEE;  Example: select sum (salary) as total\_salary from EMPLOYEE;  Example: Select min (salary) as minsal from EMPLOYEE;  **Set Operations:**   1. Union –   UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.  Query:  SELECT \* FROM First  UNION  SELECT \* FROM Second;   1. Union All –   This operation is similar to Union. But it also shows the duplicate rows.  Query:  SELECT \* FROM First  UNION ALL  SELECT \* FROM Second;   1. Intersect –   Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.  Query:  SELECT \* FROM First  INTERSECT  SELECT \* FROM Second;   1. Minus –   The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.  Query:  SELECT \* FROM First  MINUS  SELECT \* FROM Second;  **Any and All:**  The ANY and ALL operators are used with a WHERE or HAVING clause. The ANY operator returns true if any of the subquery values meet the condition. The ALL operator returns true if all of the subquery values meet the condition.   1. Any –   SELECT *column\_name(s)* FROM *table\_name* WHERE *column\_name operator* ANY (SELECT *column\_name*FROM *table\_name* WHERE *condition*);  Example: The following SQL statement returns TRUE and lists the productnames if it finds ANY records in the OrderDetails table that quantity = 10:  SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);   1. All –   SELECT *column\_name(s)* FROM *table\_name* WHERE *column\_name operator* ALL (SELECT *column\_name*FROM *table\_name*WHERE *condition*);  Example: The following SQL statement returns TRUE and lists the productnames if ALL the records in the OrderDetails table has quantity = 10:  SELECT ProductName FROM Products WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10); |

|  |
| --- |
| **Implementation Details (Problem Statement, Query and Screenshots of Results):** |
| Codes for the all the SQL Queries below:  SELECT \* FROM customer  SELECT customer\_name, customer\_mobile, customer\_email  FROM customer  WHERE customer\_address = 'Mumbai'  UNION  SELECT customer\_name, customer\_mobile, customer\_email  FROM customer  WHERE pay\_amount > 5000;  SELECT customer\_name, customer\_mobile, customer\_email  FROM customer  WHERE customer\_address = 'Mumbai'  INTERSECT  SELECT customer\_name, customer\_mobile, customer\_email  FROM customer  WHERE pay\_amount > 5000;  SELECT customer\_address, SUM(pay\_amount)  FROM customer  GROUP BY customer\_address;  SELECT \*  FROM customer  WHERE pay\_amount > ANY (SELECT AVG(booking\_id) FROM customer);  SELECT customer\_name, customer\_mobile, customer\_address, customer\_email  FROM customer  WHERE customer\_address = 'Mumbai'  AND pay\_amount > 0  EXCEPT  SELECT customer\_name, customer\_mobile, customer\_address, customer\_email  FROM customer  WHERE customer\_address = 'Mumbai'  AND booking\_id IS NOT NULL;  SELECT \*  FROM customer  WHERE pay\_amount > ALL (SELECT MAX(pay\_amount) FROM customer WHERE customer\_address = 'Indore');  SELECT COUNT(\*)  FROM customer;  SELECT MAX(pay\_amount) AS Maximum\_Pay\_Amount from customer;  SELECT MIN(pay\_amount) AS Minimum\_Pay\_Amount from customer;  SELECT SUM(pay\_amount) AS Total\_Pay\_Amount from customer;  SELECT customer\_name  FROM customer  GROUP BY customer\_name, customer\_address  HAVING AVG(pay\_amount) >= ALL (      SELECT AVG(pay\_amount)      FROM customer      GROUP BY customer\_address  );  SELECT \*  FROM customer  ORDER BY room\_no ASC;  SELECT \*  FROM customer  ORDER BY customer\_address ASC, pay\_amount DESC;  SELECT \*,         pay\_amount \* 0.95 AS discounted\_total\_payment  FROM customer  ORDER BY discounted\_total\_payment DESC;  SELECT \*  FROM customer  ORDER BY 2 ASC;  SELECT \*  FROM customer  WHERE customer\_id NOT BETWEEN 1 AND 90;  SELECT \* FROM customer WHERE customer\_address NOT IN('Mumbai', 'Delhi', 'Kolkata', 'Chennai');  SELECT \*  FROM customer  WHERE customer\_name LIKE 'A%';  SELECT \*  FROM customer  ORDER BY (CASE customer\_address              WHEN 'Mumbai' THEN 1              WHEN 'Delhi' THEN 2              WHEN 'Pune' THEN 3              WHEN 'Lucknow' THEN 4              WHEN 'Indore' THEN 5              ELSE 100           END) ASC, customer\_address DESC; |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **Write SQL query for following statements:****Find all student sorted by their department name, if there** SELECT student\_id, student\_name, department\_name, total\_credit FROM students ORDER BY department\_name; **Are two students have the same department name, then sort them by** SELECT student\_id, student\_name, department\_name, total\_credit FROM students ORDER BY department\_name, total\_credit; **Total credit in ascending order, then by their “student name” alias.** SELECT student\_id, student\_name, department\_name, total\_credit FROM students ORDER BY department\_name, total\_credit, student\_name; |

|  |
| --- |
| **Conclusion:** |
| In this experiment, we practiced intermediate SQL operations like Aggregate Functions, Set Operations such as Union, Intersect, Minus, different Clauses, Group By as well as nested queries using various operators. These exercises help in understanding database manipulation techniques using SQL. |

**Signature of faculty in-charge with Date:**