|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Database Management System Laboratory** | **Semester:** | **IV** |
| **Date of Performance:** | **14 / 03 / 2023** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Prof. Shila Dhande** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 6**

**Title: Queries based on Join and Views**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **Objective:** To be able to use SQL JOIN clause to extract data from 2 (or more) tables, we need a relationship between certain columns in these tables. |

|  |
| --- |
| **COs to be achieved:** |
| **CO3:** Use SQL for Relational database creation, maintenance and query processing  **CO4:** Applying normalization to design database |

|  |
| --- |
| **Books / Journals / Websites Referred:** |
| 1. G. K. Gupta:” Database Management Systems”, McGraw – Hill 2. Korth, Slberchatz, Sudarshan: “Database Systems Concept”, 6th Edition, McGraw Hill 3. Elmasri and Navathe, “Fundamentals of Database Systems”, 5th Edition, PEARSON Education. |

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

|  |
| --- |
| **Theory:** |
| **Join** is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied. Or JOINS are used to retrieve data from multiple tables. A JOIN is performed whenever two or more tables are joined in a SQL statement.  There are different types of Joins:   * The CROSS JOIN * The INNER JOIN * The LEFT OUTER JOIN * The RIGHT OUTER JOIN * The FULL OUTER JOIN   A **CROSS JOIN** matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have x+y columns. Because CROSS JOINs have the potential to generate extremely large tables, care must be taken to use them only when appropriate.  **Example:**  SELECT EMP\_ID, NAME, DEPT FROM COMPANY CROSS JOIN DEPARTMENT;  An **INNER JOIN** creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows, which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of table1 and table2 are combined into a result row.  **Example:**  SELECT EMP\_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP\_ID;  The **OUTER JOIN** is an extension of the INNER JOIN. SQL standard defines three types of OUTER JOINs: LEFT, RIGHT, and FULL and PostgreSQL supports all of these.  In case of **LEFT OUTER JOIN**, an inner join is performed first. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.  **Example:**  SELECT EMP\_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP\_ID;  In the **RIGHT OUTER JOIN**, first, an inner join is performed. Then, for each row in table T2 that does not satisfy the join condition with any row in table T1, a joined row is added with null values in columns of T1. This is the converse of a left join; the result table will always have a row for each row in T2.  **Example:**  SELECT EMP\_ID, NAME, DEPT FROM COMPANY RIGHT OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP\_ID;  In **THE FULL OUTER JOIN**, first, an inner join is performed. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. In addition, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.  **Example:**  SELECT EMP\_ID, NAME, DEPT FROM COMPANY FULL OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP\_ID;  **VIEWS** are pseudo-tables. That is, they are not real tables; nevertheless, appear as ordinary tables to SELECT. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables. Because views are assigned separate permissions, you can use them to restrict table access so that the users see only specific rows or columns of a table.  A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.  Views, which are kind of virtual tables, allow users to do the following −   * Structure data in a way that users or classes of users find natural or intuitive. * Restrict access to the data such that a user can only see limited data instead of complete table. * Summarize data from various tables, which can be used to generate reports.   Since views are not ordinary tables, you may not be able to execute a DELETE, INSERT, or UPDATE statement on a view. However, you can create a RULE to correct this problem of using DELETE, INSERT or UPDATE on a view.  **Syntax:**  CREATE [TEMP | TEMPORARY] VIEW view\_name AS  SELECT column1, column2....  FROM table\_name  WHERE [condition];  **Example:**  CREATE VIEW COMPANY\_VIEW AS  SELECT ID, NAME, AGE  FROM COMPANY;  Dropping Views  **Syntax:**  DROP VIEW view\_name; |

|  |
| --- |
| **Implementation Details (Problem Statement, Query and Screenshots of Results):** |
| **Implementation Code –**  -- Cross Join  SELECT \*  FROM Customer  CROSS JOIN Rooms;  -- Inner Join  SELECT Customer.customer\_name, Customer.customer\_phone, Booking.check\_in\_date  FROM Booking  INNER JOIN Customer ON Booking.customer\_id = Customer.customer\_id;  -- Left Outer Join  SELECT Customer.customer\_id, Customer.customer\_name, Customer.customer\_phone  FROM Customer  LEFT OUTER JOIN Customer\_Services ON Customer.customer\_id = Customer\_Services.customer\_id  WHERE Customer\_Services.service\_id = 222;  -- Right Outer Join  SELECT Customer.customer\_name, Customer.customer\_phone, Customer.customer\_email  FROM Customer  RIGHT OUTER JOIN Booking ON Customer.customer\_id = Booking.customer\_id  WHERE Booking.check\_in\_date > '2024-01-01';  -- Full Outer Join  SELECT \*  FROM Customer  FULL OUTER JOIN Booking ON Customer.customer\_id = Booking.customer\_id;  DROP VIEW IF EXISTS Booking\_Details;  -- Creating View  CREATE VIEW Booking\_Details AS  SELECT b.booking\_id, c.customer\_name, c.customer\_email, c.customer\_phone,         r.room\_number, r.room\_type, r.room\_price,         b.check\_in\_date, b.check\_out\_date, b.booking\_date  FROM Booking b  INNER JOIN Customer c ON b.customer\_id = c.customer\_id  INNER JOIN Rooms r ON b.room\_id = r.room\_id;  SELECT \* FROM Booking\_Details;  **Query Results –** |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **What is a view?****A view is a special stored procedure executed when certain event occurs****A view is a virtual table which results of executing a pre-compiled query****A view is a database diagram****None of the Mentioned****What type of join is needed when you wish to include rows that do not have matching values?****Equi-join****Natural join****Outer join****All of the mentioned****Write SQL query including join operator to get following output:** **Input Tables –**    **Output Tables –**    SELECT c.id, c.name, ci.id AS class\_info\_id, ci.address  FROM class c  FULL OUTER JOIN class\_info ci ON c.id = ci.id  ORDER BY c.id; |

|  |
| --- |
| **Conclusion:** |
| In conclusion, delving into database fundamentals such as joins and views has equipped us with essential skills for efficient data handling and analysis. |

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