|  |  |
| --- | --- |
| Sl. No. | QUERY |
| 1. | CREATE VIEW WestRegionOrders AS  SELECT  "Order ID",  "Order Date",  "Customer Name",  Sales  FROM sql\_assignment2.Orders  WHERE Region = 'WEST'; |
|  |  |
| 2. | CREATE OR REPLACE FUNCTION GetOrdersForMonth(last\_day\_of\_month DATE)  RETURNS TABLE (  Order\_ID TEXT,  Order\_Date DATE,  Customer\_Name TEXT,  Sales FLOAT  ) AS $$  BEGIN  RETURN QUERY  SELECT  "Order ID",  "Order Date",  "Customer Name",  Sales  FROM sql\_assignment2.Orders  WHERE  "Order Date" >= date\_trunc('MONTH', last\_day\_of\_month) AND  "Order Date" <= last\_day\_of\_month;  END;  $$ LANGUAGE plpgsql; |
|  |  |
| 3. | CREATE OR REPLACE FUNCTION GetOrdersForMonth(last\_day\_of\_month DATE)  RETURNS TABLE (  Order\_ID TEXT,  Order\_Date DATE,  Customer\_Name TEXT,  Sales\_Value FLOAT  ) AS $$  BEGIN  RETURN QUERY  SELECT  o."Order ID",  o."Order Date",  o."Customer Name",  o.Sales  FROM sql\_assignment2.Orders o  WHERE  o."Order Date" >= date\_trunc('MONTH', last\_day\_of\_month) AND  o."Order Date" <= last\_day\_of\_month;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM GetOrdersForMonth('2021-12-31');  CREATE OR REPLACE FUNCTION CalculateProfitMargin(sales FLOAT, profit FLOAT)  RETURNS FLOAT AS $$  BEGIN  RETURN CASE  WHEN sales = 0 THEN NULL -- Handle division by zero  ELSE profit / sales  END;  END;  $$ LANGUAGE plpgsql;  SELECT CalculateProfitMargin(1000, 200); |

**Interview Questions:**

|  |  |
| --- | --- |
| Sl. No. | Answers |
| 1. | Primary Key  A primary key is a unique identifier for a record in a database table. Every table can have only one primary key, and no two rows can have the same primary key value. It ensures that each record in the table can be uniquely identified.  Foreign Key  A foreign key is a column or set of columns in one table that refers to the primary key column(s) in another table. It establishes a relationship between the two tables and ensures referential integrity, meaning that the foreign key value must match a primary key value in another table or be null. |
| 2. | Union  1. Combines the results of two or more SELECT statements.  2. Removes duplicate rows.  3. Sorts the result set by default.  Union All  1. Combines the results of two or more SELECT statements.  2. Retains duplicate rows.  3. Does not sort the result set by default. |
| 3. | Delete  1. Removes rows from a table based on a condition.  2. Can delete specific rows.  3. Consumes more resources, as it logs individual row deletions.  4. Fires triggers for the affected rows.  Drop  1. Removes an entire table or database from the system.  2. Cannot delete specific rows; deletes the entire table.  3. Releases the memory and storage used by the table.  4. Drops associated indexes, triggers, constraints, and permissions.  Truncate  1. Removes all rows from a table, but the table structure remains.  2. Cannot delete specific rows; deletes all rows.  3. More efficient than DELETE, as it doesn't log individual row deletions.  4. Doesn't fire triggers. |
| 4. | Inner Join  Returns rows when there is a match in both the left and right tables.  Left (Outer) Join  Returns all rows from the left table and the matched rows from the right table. Unmatched rows from the left table are returned with NULL values for columns from the right table.  Right (Outer) Join  Returns all rows from the right table and the matched rows from the left table. Unmatched rows from the right table are returned with NULL values for columns from the left table.  Full (Outer) Join  Returns all rows when there is a match in one of the tables. Unmatched rows from either table are returned with NULL values for columns from the opposite table.  Cross Join  Returns the Cartesian product of the two tables, i.e., it combines each row from the first table with every row from the second table.  Self Join  Joins a table with itself, comparing rows within the same table.  Anti Join  Returns rows from one table where no matches are found in the other table. |
| 5. | The order of execution in SQL determines the sequence in which the components of a SQL statement are processed. The order is:  1. FROM and JOIN: Retrieve and join tables.  2. WHERE: Apply row-level filtering.  3. GROUP BY: Group rows based on specified columns.  4. HAVING: Filter groups based on a condition.  5. SELECT. Select specific columns.  6. DISTINCT: Remove duplicate rows.  7. ORDER BY: Sort the result set.  8. LIMIT (or equivalent in other RDBMS): Limit the number of returned rows. |
| 6. | Use Case: School Exam Results  Imagine students in a school have taken an exam, and we want to rank them based on their scores.  Data  Student Score  ------- -----  Alice 85  Bob 90  Charlie 90  David 80  Eva 88  Frank 88  Grace 70  RANK():  `RANK()` provides a unique rank to each distinct score, leaving gaps between ranks for duplicate scores.  Ranking Based on Scores:  Alice - 4  Bob - 1  Charlie - 1  David - 5  Eva - 2  Frank - 2  Grace - 6  DENSE\_RANK():  `DENSE\_RANK()` provides a unique rank to each distinct score, but unlike `RANK()`, it does not leave gaps between ranks for duplicate scores.  Ranking Based on Scores:    Alice - 4  Bob - 1  Charlie - 1  David - 3  Eva - 2  Frank - 2  Grace - 5  ROW\_NUMBER():  `ROW\_NUMBER()` assigns a unique number to each row based on the order specified, regardless of duplicates.  Row Numbers Based on Scores:    Alice - 4  Bob - 1 (or 2, depending on the ordering of students with the same score)  Charlie - 2 (or 1)  David - 5  Eva - 3 (or 6)  Frank - 6 (or 3)  Grace - 7    In summary:  - `RANK()` gives the same rank for tied scores but leaves gaps.  - `DENSE\_RANK()` gives the same rank for tied scores without gaps.  - `ROW\_NUMBER()` always gives a unique number to each row, even if values are the same. |
| 7. | SELECT  amount,  SUM(amount) OVER (ORDER BY some\_column) AS cumulative\_sum  FROM sales\_data; |
| 8. | SELECT salary  FROM employees  ORDER BY salary DESC  LIMIT 1 OFFSET (N-1); |
| 9. | SELECT A.\*  FROM tableA A  LEFT JOIN tableB B ON A.id = B.id  WHERE B.id IS NULL;  OR  SELECT A.\*  FROM tableA A  WHERE NOT EXISTS (  SELECT 1  FROM tableB B  WHERE A.id = B.id  ); |
| 10. | SELECT column1, column2, COUNT(\*)  FROM tableA  GROUP BY column1, column2  HAVING COUNT(\*) > 1; |