

**Assignment - 2**

**SAVEETHA SCHOOL OF ENGINEERING**



Submitted by

**PADMANABHAN S (192324076)**

Submitted to

**Dr. Christy Melwyn**

Professor

Course Code: **CSA0556**

Course Name: **Database Management Systems for Relational Database**

**Question1:**

**Top 3 Departments with Highest Average Salary**

**Write a SQL query to find the top 3 departments with the highest average salary of employees. Ensure departments with no employees show an average salary of NULL.**

QUERY :

SELECT d.DepartmentID, d.DepartmentName, AVG(e.Salary) AS AvgSalary

FROM Departments d

LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID

GROUP BY d.DepartmentID, d.DepartmentName

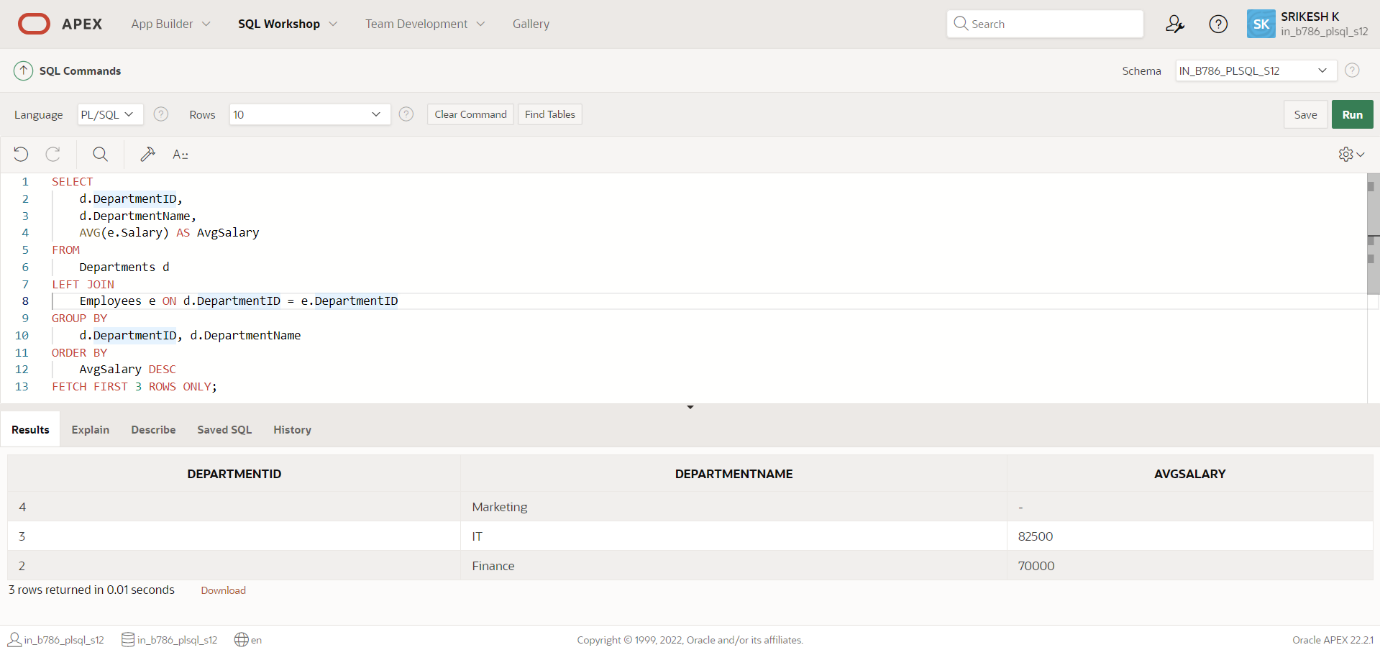
ORDER BY AvgSalary DESC

FETCH FIRST 3 ROWS ONLY;

EXPLANATION:

* selecting the Department ID and Department Name from the Departments table.
* It also calculates the average salary (AVG(e.Salary)) of employees in each department. If a department has no employees, the average salary will be NULL due to the LEFT JOIN.
* Specifies that the main table for the query is the Departments table, aliased as d.
* This performs a left join between the Departments table (d) and the Employees table (e).
* The join condition is d.DepartmentID = e.DepartmentID, meaning it links departments to their respective employees.
* A LEFT JOIN ensures that all departments are included in the result set, even those without any employees. If there are no employees in a department, the corresponding e.Salary will be NULL.
* This groups the results by DepartmentID and DepartmentName, ensuring that the average salary is calculated for each department.
* This sorts the results by the average salary (AvgSalary) in descending order, so departments with the highest average salaries come first.
* This limits the result set to the top 3 rows, effectively returning only the top 3 departments with the highest average salaries.

**TABLES :**



**Question 2:**

**Retrieving Hierarchical Category Paths**

**Write a SQL query using recursive Common Table Expressions (CTE) to retrieve all categories along with their full hierarchical path (e.g., Category > Subcategory > Sub-subcategory).**

**QUERY :**

SELECT CategoryID, CategoryName,SYS\_CONNECT\_BY\_PATH(CategoryName, ' > ')

AS HierarchicalPath

FROM ProductCategories

START WITH ParentCategoryID IS NULL

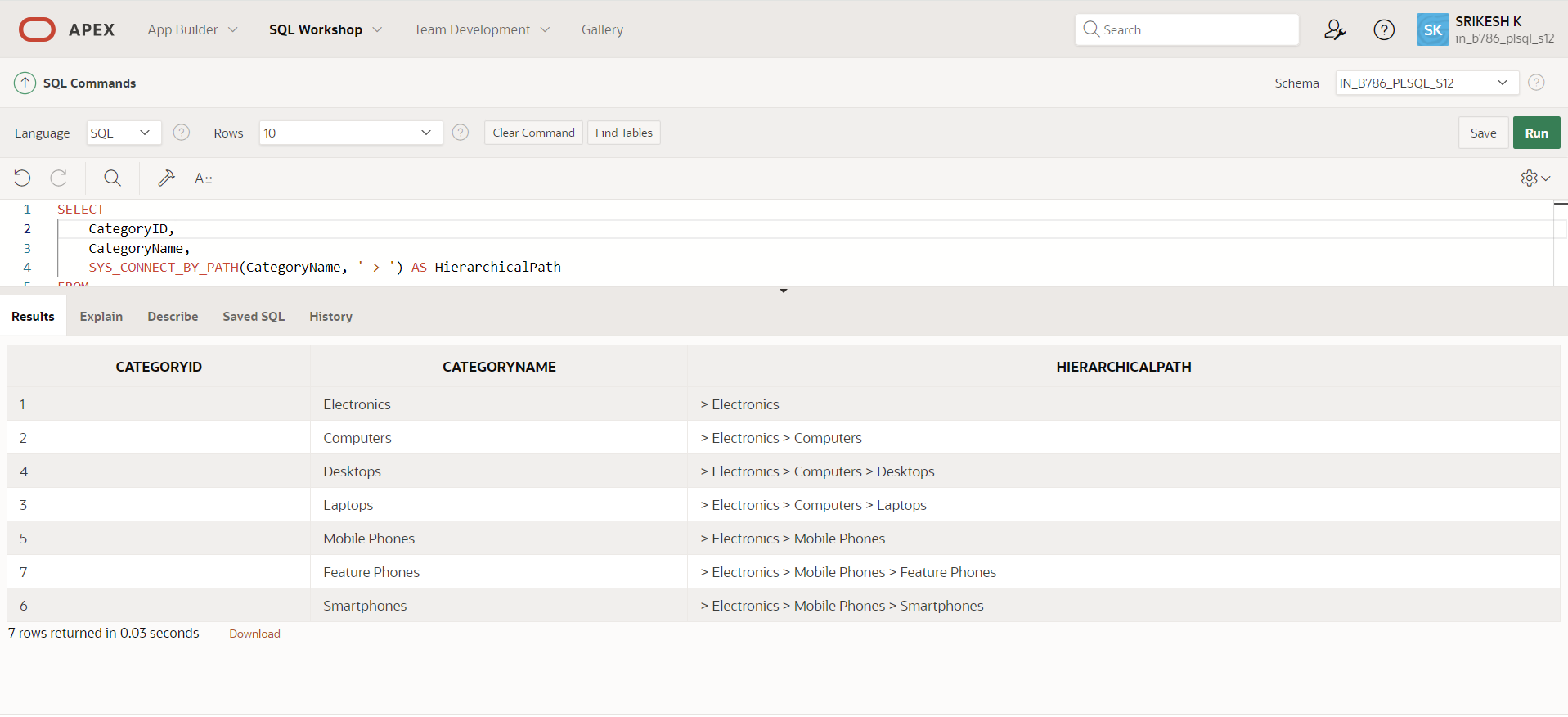
CONNECT BY PRIOR CategoryID = ParentCategoryID

ORDER BY HierarchicalPath;

EXPLANATION:

1. **LINE 1:**
   * This part selects the CategoryID and CategoryName from the ProductCategories table.
   * It also creates a string (HierarchicalPath) showing the full path of the category hierarchy, with each level separated by ' > '.
2. **LINE 2: FROM ProductCategories:**
   * Specifies that the query is working with the ProductCategories table.
3. **LINE 3: START WITH ParentCategoryID IS NULL:**
   * This specifies the starting point of the hierarchy, which is the root categories that do not have a parent (i.e., ParentCategoryID is NULL).
4. **FINALIZING: CONNECT BY PRIOR CategoryID = ParentCategoryID:**
   * This part defines the relationship between parent and child categories. It connects each category (CategoryID) to its parent (ParentCategoryID).
5. **ORDER BY HierarchicalPath:**
   * This sorts the results by the hierarchical path.

**TABLE :**



**Question 3:**

**Total Distinct Customers by Month**

**Design a SQL query to find the total number of distinct customers who made a purchase in each month of the current year. Ensure months with no customer activity show a count of 0.**

**QUERY :**

WITH Months AS (

SELECT TO\_CHAR(ADD\_MONTHS(TRUNC(SYSDATE, 'YEAR'), LEVEL - 1), 'Month') AS MonthName,

LEVEL AS MonthNumber

FROM DUAL

CONNECT BY LEVEL <= 12

),

CustomerCounts AS (

SELECT TO\_CHAR(PurchaseDate, 'Month') AS MonthName,

COUNT(DISTINCT CustomerID) AS CustomerCount

FROM Purchases

WHERE EXTRACT(YEAR FROM PurchaseDate) = EXTRACT(YEAR FROM SYSDATE)

GROUP BY TO\_CHAR(PurchaseDate, 'Month'), TO\_CHAR(PurchaseDate, 'MM')

)

SELECT m.MonthName,

NVL(c.CustomerCount, 0) AS CustomerCount

FROM Months m

LEFT JOIN CustomerCounts c ON m.MonthName = c.MonthName

ORDER BY m.MonthNumber;

**EXPLANATION:**

**Segment 1:**

* TRUNC(SYSDATE, 'YEAR') truncates the current date to the start of the year.
* ADD\_MONTHS(..., LEVEL - 1) adds months to the start of the year, where LEVEL is a pseudo-column in Oracle that generates a series of numbers.
* LEVEL <= 12 ensures it generates 12 months.
* TO\_CHAR(..., 'Month') formats the month as its full name.
* LEVEL provides the month number (1 for January, 2 for February, etc.).

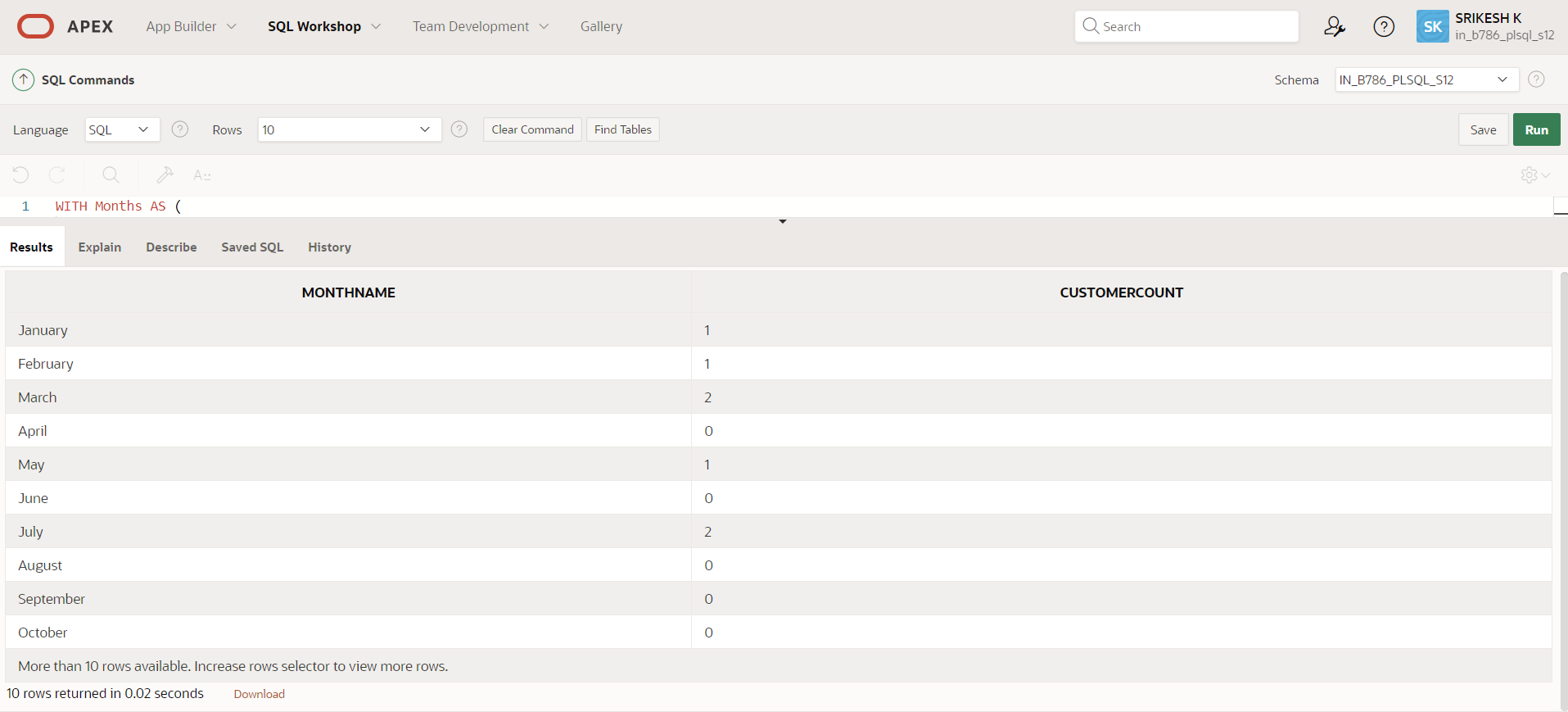
**Segment 2:**

* TO\_CHAR(PurchaseDate, 'Month') formats the purchase date as its full month name.
* COUNT(DISTINCT CustomerID) counts the unique customers.
* WHERE EXTRACT(YEAR FROM PurchaseDate) = EXTRACT(YEAR FROM SYSDATE) filters the purchases to the current year.
* GROUP BY TO\_CHAR(PurchaseDate, 'Month'), TO\_CHAR(PurchaseDate, 'MM') groups the results by month name and month number to ensure correct sorting.

**Segment 3:**

* LEFT JOIN CustomerCounts c ON m.MonthName = c.MonthName joins all months with customer counts, ensuring all months are included.
* NVL(c.CustomerCount, 0) replaces NULL values with 0 for months with no purchases.
* ORDER BY m.MonthNumber sorts the results by the month number.

**TABLE :**



**Question 4: Finding Closest Locations**

**Write a SQL query to find the closest 5 locations to a given point specified by latitude and longitude. Use spatial functions or advanced mathematical calculations for proximity**.

**QUERY :**

WITH input AS (SELECT34.052235 AS latitude,-118.243683 AS longitude

FROM dual)

SELECT \*FROM (

SELECT loc.LocationID, loc.LocationName, loc.Latitude, loc.Longitude, (6371 \* ACOS(

LEAST(1, GREATEST(-1, COS(input.latitude \* (3.141592653589793 / 180)) \*

COS(loc.Latitude \* (3.141592653589793 / 180)) \*

COS(loc.Longitude \* (3.141592653589793 / 180) - input.longitude \* (3.141592653589793 / 180)) +

SIN(input.latitude \* (3.141592653589793 / 180)) \*

SIN(loc.Latitude \* (3.141592653589793 / 180)) )) )) AS Distance

FROM Locations loc, input

ORDER BY Distance ) WHERE ROWNUM <= 5;

**EXPLANATION:**

Segment 1:

* 34.052235 AS latitude and -118.243683 AS longitude specify the coordinates.
* FROM dual is a dummy table in Oracle used for selecting a constant value.

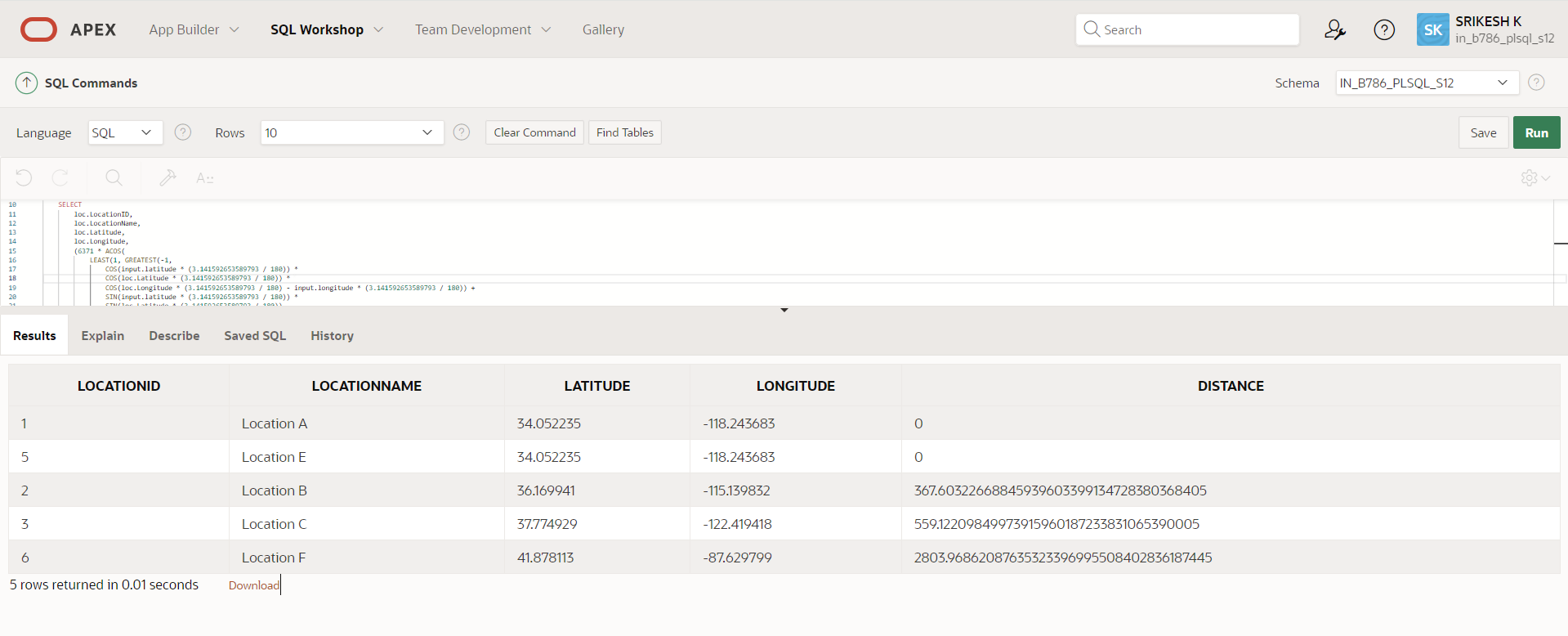
Segment 2:

* The Haversine formula is used to calculate the distance between two points on the Earth's surface.
* 6371 is the radius of the Earth in kilometers.
* COS and SIN functions are used to compute the trigonometric values.
* LEAST(1, GREATEST(-1, ...)) ensures the value passed to ACOS is within the valid range of -1 to 1, preventing errors.

Segment 3:

* ROWNUM is a pseudo-column in Oracle that assigns a unique number to each row returned by the query.
* ROWNUM <= 5 ensures only the first five rows (i.e., the closest locations) are selected.

**TABLE :**



**Question 5:**

**Optimizing Query for Orders Table**

**Write a SQL query to retrieve orders placed in the last 7 days from a large Orders table, sorted by order date in descending order**.

**QUERY :**

CREATE INDEX idx\_orderdate ON Orders(OrderDate);

SELECT OrderID, OrderDate, CustomerID, TotalAmount

FROM Orders

WHERE OrderDate >= SYSDATE - 7

ORDER BY OrderDate DESC;

**EXPLANATION:**

Segment 1:

**Creating the Index:**

* **Index:** An index is a database object that improves the speed of data retrieval operations on a table at the cost of additional storage space and maintenance overhead during data modification operations.
* **idx\_orderdate:** The name of the index being created.
* **Orders(OrderDate):** Specifies that the index is created on the OrderDate column of the Orders table.

**Benefit:** By creating this index, queries filtering or sorting by OrderDate will be faster because the database can quickly locate the relevant rows.

**2. Retrieving Recent Orders**

* SELECT OrderID, OrderDate, CustomerID, TotalAmount: Specifies the columns to be retrieved: order ID, order date, customer ID, and total amount.
* FROM Orders: Indicates that the data is coming from the Orders table.
* WHERE OrderDate >= SYSDATE - 7: Filters the results to include only orders where the OrderDate is within the last 7 days from the current system date (SYSDATE).
* ORDER BY OrderDate DESC: Orders the results by OrderDate in descending order, so the most recent orders appear first.

**TABLE :**

