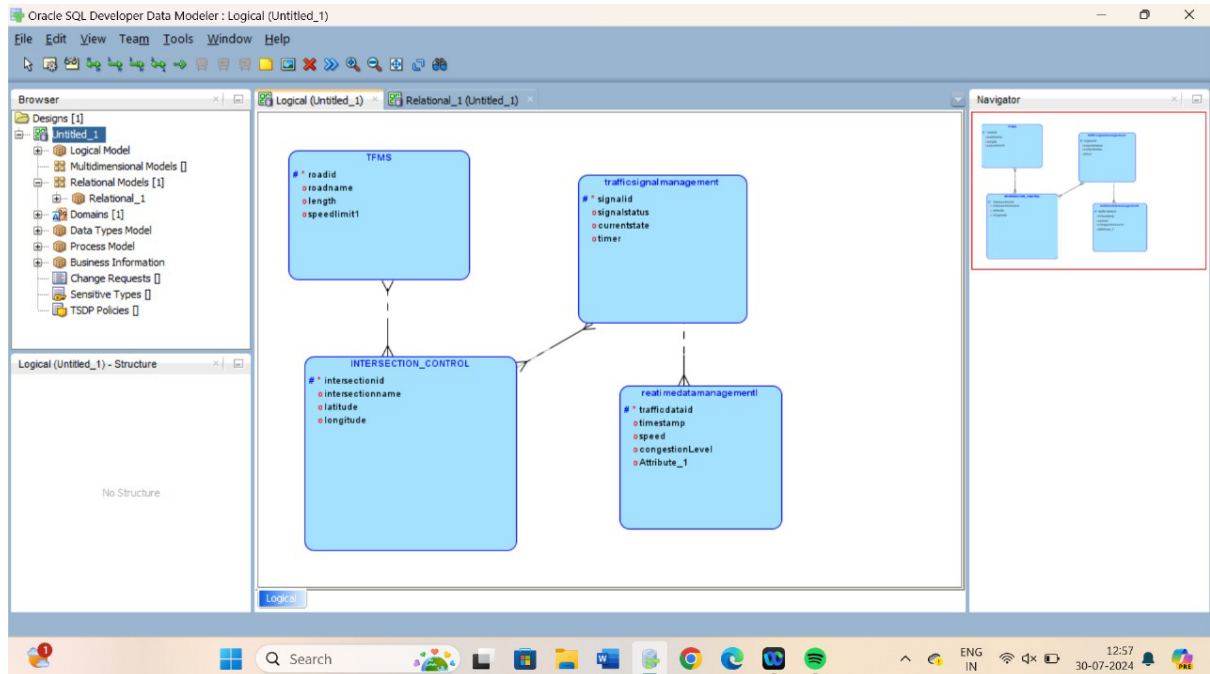


QUESTION 1

1.ER diagram



2. Entity Definitions

Roads

- **Attributes:**
 - Road_ID (Primary Key): Unique identifier for each road.
 - Road_Name: The name of the road.
 - Road_Length: The length of the road in kilometers.
 - Number_of_Lanes: The number of lanes on the road.

Intersections

- **Attributes:**
 - Intersection_ID (Primary Key): Unique identifier for each intersection.
 - Intersection_Name: The name or description of the intersection.
 - Intersection_Type: The type of intersection (e.g., 3-way, 4-way).

Traffic_Signals

- **Attributes:**
 - Signal_ID (Primary Key): Unique identifier for each traffic signal.
 - Signal_Status: Current status of the signal (e.g., Red, Green, Yellow).



- **Signal_Type:** The type of signal (e.g., Pedestrian, Vehicle).
- **Intersection_ID (Foreign Key):** Identifier linking the signal to an intersection.

Traffic_Data

- **Attributes:**
 - **Data_ID (Primary Key):** Unique identifier for each traffic data record.
 - **Timestamp:** The time and date when the data was recorded.
 - **Traffic_Flow:** Number of vehicles passing a point per minute.
 - **Road_ID (Foreign Key):** Identifier linking the data to a road.
 - **Intersection_ID (Foreign Key):** Identifier linking the data to an intersection.

3. Relationship Descriptions

Roads and Intersections

- **Relationship: Roads connect to Intersections.**
 - **Type:** One-to-Many
 - **Cardinality:** One road can connect to multiple intersections.
 - **Optionality:** Mandatory for both entities.

Intersections and Traffic_Signals

- **Relationship: Intersections host Traffic Signals.**
 - **Type:** One-to-Many
 - **Cardinality:** One intersection can host multiple traffic signals.
 - **Optionality:** Mandatory for both entities.

Roads and Traffic_Data

- **Relationship: Roads have Traffic Data.**
 - **Type:** One-to-Many
 - **Cardinality:** One road can have multiple traffic data entries.
 - **Optionality:** Mandatory for both entities.

Intersections and Traffic_Data

- **Relationship: Intersections have Traffic Data.**
 - **Type:** One-to-Many
 - **Cardinality:** One intersection can have multiple traffic data entries.
 - **Optionality:** Mandatory for both entities.

4. Justification Document



Design Choices

Scalability:

- The design supports adding new roads, intersections, and signals without significant changes.
- Efficiently handles extensive traffic data collection and analysis.

Real-Time Data Processing:

- The Traffic_Data entity includes timestamps for real-time data analysis and decision-making.
- Indexed attributes (e.g., Timestamp, Road_ID, Intersection_ID) optimize query performance.

Efficient Traffic Management:

- Tracks traffic signals and data at intersections for optimized traffic flow.
- Enables detailed traffic flow analysis and adaptive traffic control measures.

Normalization Considerations

1NF (First Normal Form):

- All attributes contain atomic values.
- Each entity has a primary key for unique record identification.

2NF (Second Normal Form):

- All non-key attributes are fully dependent on the primary key.
- Eliminates partial dependencies by ensuring each non-key attribute relates to the whole primary key.

3NF (Third Normal Form):

- Attributes are functionally dependent only on the primary key.
- Eliminates transitive dependencies, ensuring non-key attributes do not depend on other non-key attributes.

This structured approach minimizes redundancy and ensures data integrity, supporting TFMS's goals for accurate and efficient traffic management.

Question 2:

Question 1: Top 3 Departments with Highest Average Salary

1.SQL query:



Edit with WPS Office

The screenshot shows the Oracle APEX SQL Workshop interface. The SQL command window contains the following query:

```

1 SELECT
2   d.DepartmentID,
3   d.DepartmentName,
4   AVG(e.Salary) AS AvgSalary
5 FROM
6   Departments d
7 LEFT JOIN
8   Employees e ON d.DepartmentID = e.DepartmentID
9 GROUP BY
10  d.DepartmentID,
11  d.DepartmentName
12 ORDER BY
13  AvgSalary DESC

```

The Results window shows the following data:

DEPARTMENTID	DEPARTMENTNAME	AVGSALARY
2	Engineering	85000
1	HR	50000

2 rows returned in 0.03 seconds. Download

2. Explanation

SQL Query Breakdown:

1. FROM Clause:

- **FROM Departments d:** Selects the **Departments** table and assigns it an alias **d**.
- **LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID:** Performs a left join between the **Departments** table (**d**) and the **Employees** table (**e**). This means all rows from the **Departments** table will be included in the result, even if there are no matching rows in the **Employees** table. If no matching rows are found, the columns from the **Employees** table will contain **NULL**.

2. SELECT Clause:

- **d.DepartmentID:** Selects the **DepartmentID** from the **Departments** table.
- **d.DepartmentName:** Selects the **DepartmentName** from the **Departments** table.
- **AVG(e.Salary) AS AvgSalary:** Calculates the average salary of employees in each department. If there are no employees in a department, **AVG(e.Salary)** will return **NULL**.

3. GROUP BY Clause:

- **GROUP BY d.DepartmentID, d.DepartmentName:** Groups the results by **DepartmentID** and **DepartmentName**. This ensures that the average salary is calculated for each department.

4. ORDER BY Clause:



Edit with WPS Office

- **ORDER BY AvgSalary DESC:** Orders the results by the **AvgSalary** in descending order, so the departments with the highest average salaries appear first.

5. **LIMIT Clause:**

- **LIMIT 3:** Limits the result set to the top 3 departments based on the highest average salary.

Handling Departments with No Employees:

- **Left Join:** The **LEFT JOIN** ensures that all departments are included in the result set, even if there are no employees in some departments. For these departments, the **Salary** column from the **Employees** table will be **NULL**.
- **AVG Function:** The **AVG(e.Salary)** function calculates the average salary for each department. If there are no employees in a department, the function will return **NULL** because there are no salary values to average. This handles departments with no employees appropriately by showing **NULL** for the average salary.

Question 2: Retrieving Hierarchical Category Paths

1.SQL QUERY:



Edit with WPS Office

The screenshot shows the Oracle APEX SQL Workshop interface. The SQL command window contains the following query:

```

1 SELECT
2   CategoryID,
3   CategoryName,
4   LEVEL AS Depth,
5   SYS_CONNECT_BY_PATH(CategoryName, '> ') AS Path
6 FROM
7   Categories
8 START WITH
9   ParentCategoryID IS NULL
10 CONNECT BY
11   PRIOR CategoryID = ParentCategoryID
12 ORDER BY
13   Path;

```

The Results tab shows the following data:

CATEGORYID	CATEGORYNAME	DEPTH	PATH
1	Electronics	1	> Electronics
2	Computers	2	> Electronics > Computers

2 rows returned in 0.07 seconds. Download

2.Explanation:

- **START WITH:** Starts the hierarchy with categories that have no parent.
- **CONNECT BY:** Establishes the parent-child relationship between categories.
- **SYS_CONNECT_BY_PATH:** Constructs the hierarchical path for each category.
- **LEVEL:** Provides the depth of each category in the hierarchy.
- **ORDER BY Path:** Orders the output based on the constructed path.

Question 3: Total Distinct Customers by Month

1.SQL QUERY:



Edit with WPS Office

The screenshot shows the Oracle APEX SQL Workshop interface. The SQL command is as follows:

```

WITH MONTHS AS (
  SELECT TO_CHAR(ADD_MONTHS(TRUNC(SYSDATE, 'YEAR'), LEVEL - 1), 'YYYY-MM') AS Month,
         TO_CHAR(ADD_MONTHS(TRUNC(SYSDATE, 'YEAR'), LEVEL - 1), 'Month') AS MonthName
  FROM dual
)
SELECT MonthName, COUNT(*) AS CustomerCount
FROM MONTHS
LEFT JOIN Purchases ON MONTHS.Month = Purchases.Month
GROUP BY MonthName
ORDER BY MonthName;

```

The results table shows the following data:

MONTHNAME	CUSTOMERCOUNT
January	2
February	0
March	0
April	0
May	0
June	0
July	0
August	0
September	0

Explanation

- Months CTE:**
 - Generates a list of all months for the current year. `TO_CHAR(ADD_MONTHS(TRUNC(SYSDATE, 'YEAR'), LEVEL - 1), 'YYYY-MM')` provides the month in YYYY-MM format, and `TO_CHAR(ADD_MONTHS(TRUNC(SYSDATE, 'YEAR'), LEVEL - 1), 'Month')` provides the full month name.
- CustomerPurchases CTE:**
 - Retrieves distinct customer IDs and the corresponding purchase months for the current year from the **Purchases** table.
- LEFT JOIN:**
 - Joins the **Months** CTE with the **CustomerPurchases** CTE on the month field. This ensures that all months are included, even if there are no purchases in some months.
- COALESCE(COUNT(cp.CustomerID), 0):**
 - Counts the distinct customers for each month. **COALESCE** ensures that if there are no customers for a particular month, it shows **0** instead of **NULL**.
- GROUP BY:**
 - Groups the results by month and month name to aggregate the customer counts.
- ORDER BY:**
 - Orders the results by month to display them in chronological order.

Question 4: Finding Closest Locations

1.SQL QUERY



Edit with WPS Office

The screenshot displays the Oracle APEX SQL Workshop interface. The SQL command window contains the following query:

```

1 Define the value of pi as a constant
2 WITH DistanceCalc AS (
3     SELECT
4         LocationID,
5         LocationName,
6         Latitude,
7         Longitude,
8         Value of pi
9     FROM
10        Locations,
11        DUAL
12     WHERE
13        Value of pi = 3.141592653589793
14 )
15 SELECT
16     LocationID,
17     LocationName,
18     Latitude,
19     Longitude,
20     Distance
21 FROM
22     DistanceCalc
23 WHERE
24     Distance <= 10000
25 ORDER BY
26     Distance

```

The Results tab shows the following data:

LOCATIONID	LOCATIONNAME	LATITUDE	LONGITUDE	DISTANCE
4	Location D	51.14929	-122.41918	0
5	Location F	39.099778	94.578568	2419.862097614381147313579402916593532

2 rows returned in 0.02 seconds

2. Explanation

1. pi Value:

- 3.141592653589793 is used as the value of π (pi).

2. Conversion to Radians:

- Latitude and Longitude are converted from degrees to radians by multiplying with π and dividing by 180.

3. Haversine Formula:

- Uses the converted radians to compute the distance.

This query calculates the distance between the given point and each location in the **Locations** table and then returns the 5 closest locations based on that distance.

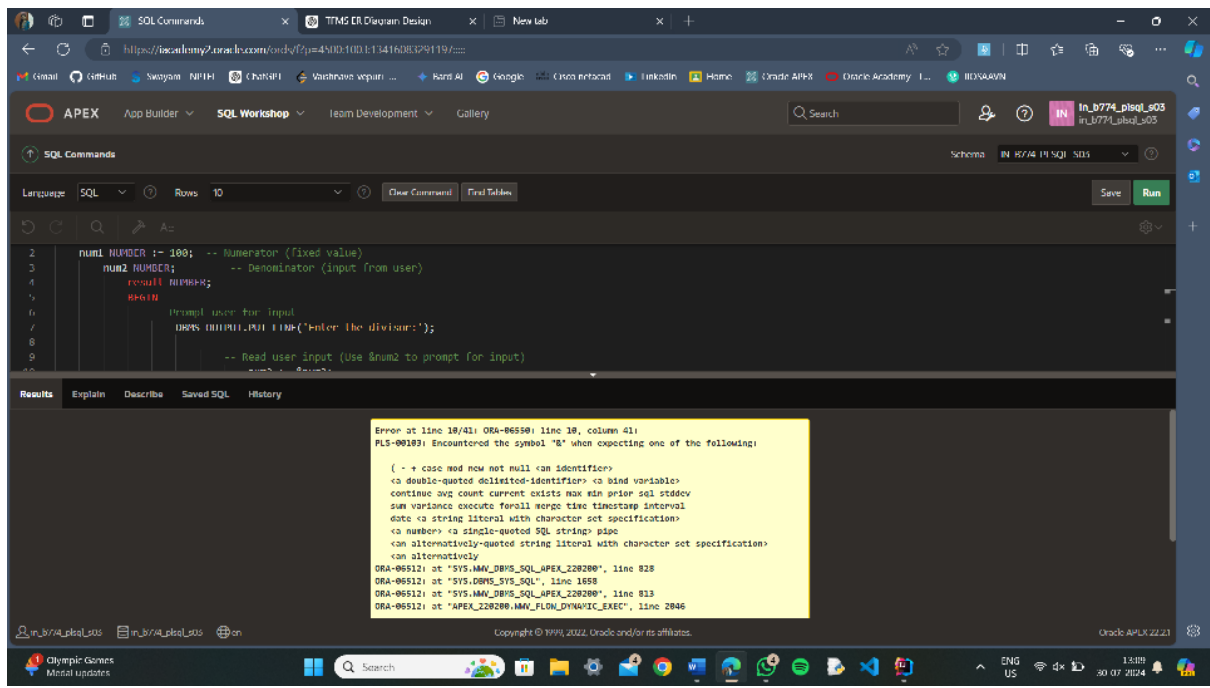
QUESTION3:

Question 1: Handling Division Operation

1. SQL QUERY



Edit with WPS Office



1. Exception Handling:

- The **EXCEPTION** block is used to catch and handle exceptions that occur during the execution of the PL/SQL block.
- The specific exception **ZERO_DIVIDE** is handled within the **EXCEPTION** block. This exception is raised when there is an attempt to divide by zero.

2. Graceful Error Message:

- When a **ZERO_DIVIDE** exception is caught, a user-friendly error message is displayed using **DBMS_OUTPUT.PUT_LINE**.
- This ensures that the program doesn't terminate abruptly and provides feedback to the user about the nature of the error.

3. User Input:

- The **&num2** placeholder is used to prompt the user for input when running the script in an environment that supports user input (e.g., SQL*Plus, Oracle APEX).
- This allows the user to provide the divisor dynamically, demonstrating how the exception handling works with different inputs.

4. Structured Block:

- The **DECLARE** section is used to define and initialize variables.
- The **BEGIN** section contains the main logic for performing the division and handling normal execution.

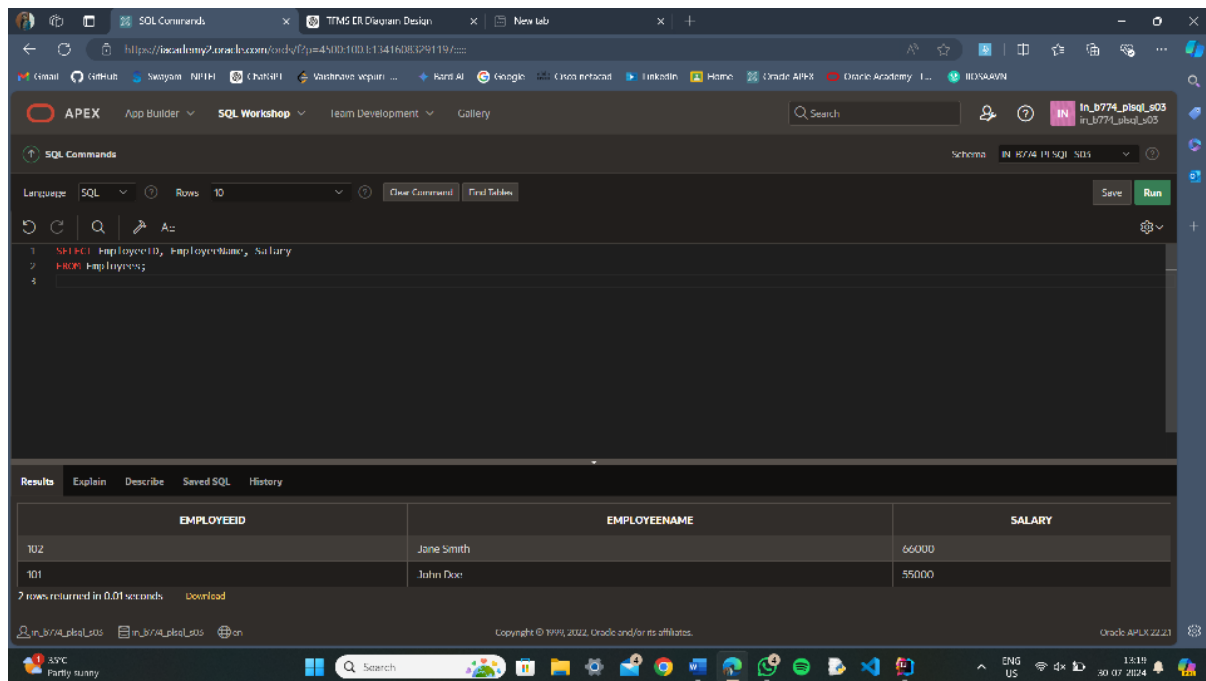


Edit with WPS Office

- The **EXCEPTION** section handles any exceptions that occur during the execution of the **BEGIN** section, ensuring the program can respond to errors appropriately.

Question 2: Updating Rows with FORALL

1. SQL QUERY:



1. Creating the Employees Table:

- This SQL command creates the **Employees** table and populates it with some initial sample data.

2. PL/SQL Block for Bulk Update:

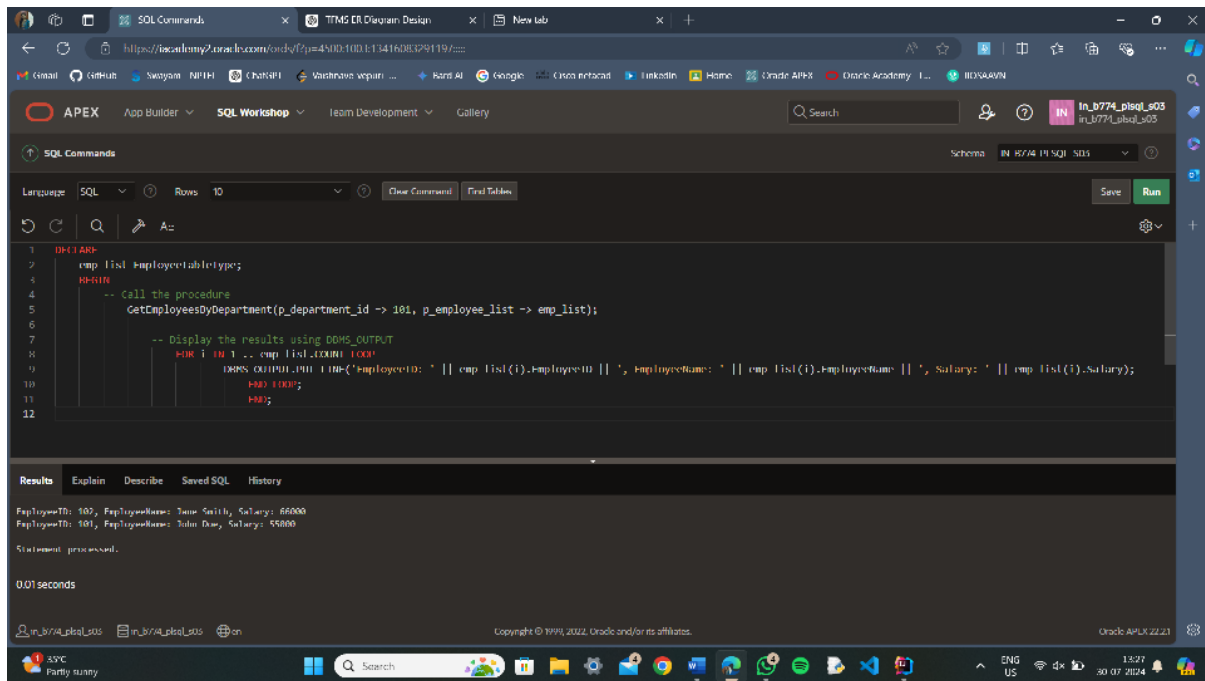
- **Collections Definition:** Defines the **EmployeeIDArray** and **SalaryIncrementArray** types.
- **Populating Arrays:** Adds sample employee IDs and corresponding salary increments to the arrays.
- **Consistency Check:** Ensures both arrays have the same number of elements to avoid errors during the update.
- **FORALL Statement:** Executes the **UPDATE** statement in bulk, updating salaries for all employees listed in the array.
- **Transaction Handling:** Commits the changes if successful; otherwise, rolls



back and prints an error message.

Question 3: Implementing Nested Table Procedure

1. SQL QUERY:



The screenshot shows the Oracle APEX SQL Workshop interface. The 'SQL Commands' tab is active, displaying an anonymous PL/SQL block. The block defines a nested table type, calls a procedure, and uses a FOR loop to print results. The 'Run' button has been clicked, and the 'Results' tab shows the output of the execution.

```
1 DECLARE
2   emp_list EmployeeTableType;
3 BEGIN
4   -- Call the procedure
5   GetEmployeesByDepartment(p_department_id => 101, p_employee_list => emp_list);
6
7   -- Display the results using DBMS_OUTPUT
8   FOR i IN 1 .. emp_list.COUNT LOOP
9     DBMS_OUTPUT.PUT_LINE('EmployeeID: ' || emp_list(i).EmployeeID || ', EmployeeName: ' || emp_list(i).EmployeeName || ', Salary: ' || emp_list(i).Salary);
10  END LOOP;
11 END;
```

Results Explain Describe Saved SQL History

EmployeeID: 102, EmployeeName: Tom Smith, Salary: 60000
EmployeeID: 101, EmployeeName: John Doe, Salary: 55000

Statement processed.
0.01 seconds

2.Explanation:

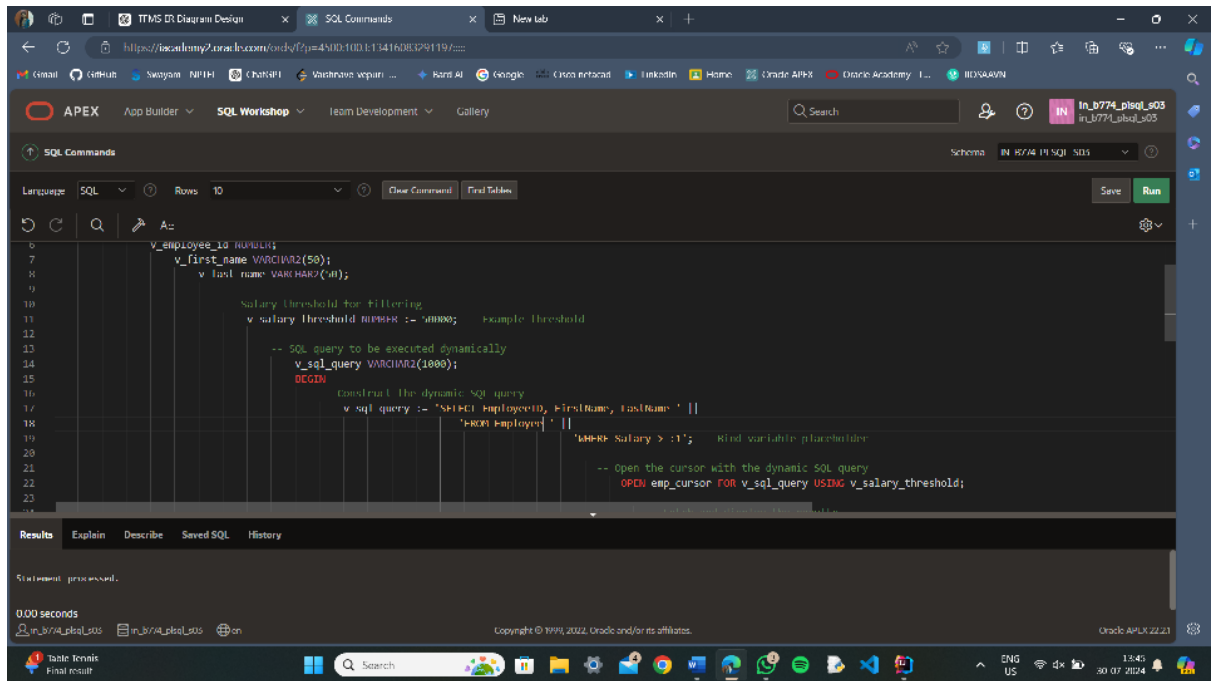
1. **Creating Types:** Define **EmployeeRecord** and **EmployeeTableType** to hold and manage employee data.
2. **Procedure:** **GetEmployeesByDepartment** fetches data for a specific department and populates the nested table.
3. **Testing:** An anonymous block tests the procedure by retrieving data for department ID 101 and printing it.

Question4:

1. SQL QUERY:



Edit with WPS Office



2.Explanation

1.create the REF CURSOR Package:

2.Execute the CREATE OR REPLACE PACKAGE statement in SQL Workshop.

3.Create the Employees Table:

4.Ensure the Employees table exists with the columns EmployeeID, FirstName, LastName, and Salary.

5.Execute the PL/SQL Block:

Run the PL/SQL block in SQL Workshop or SQL Commands.

6.Enable DBMS_OUTPUT:

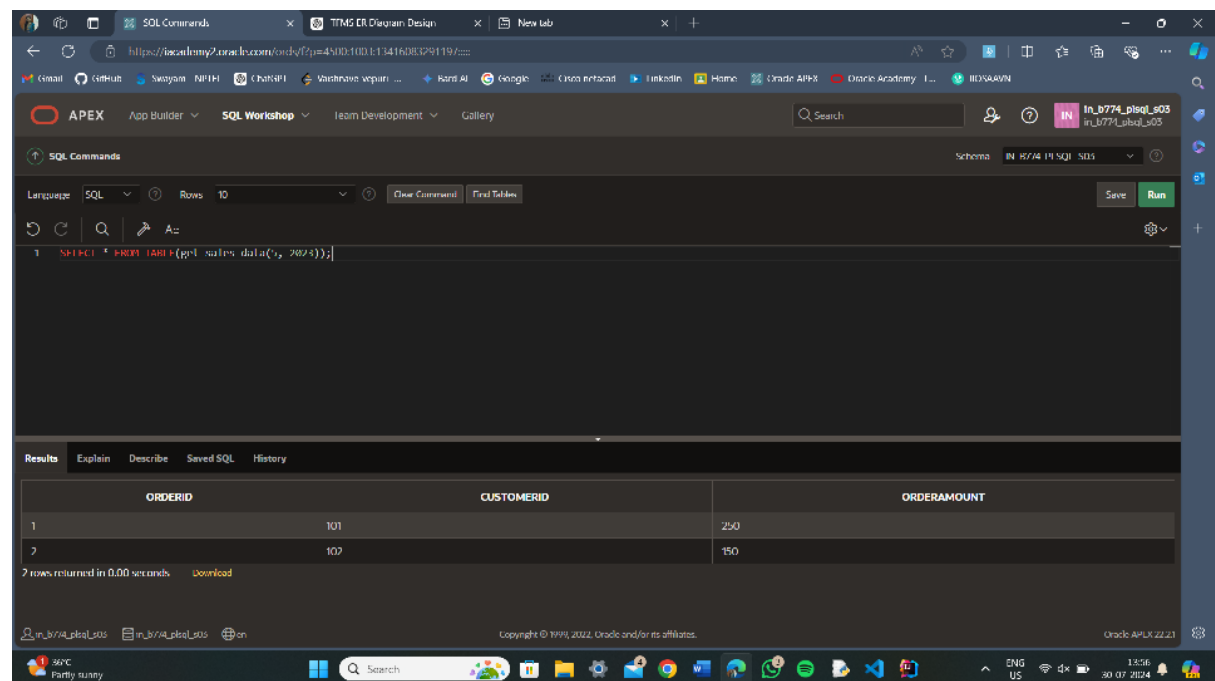
Make sure DBMS_OUTPUT is enabled in your environment to view the output from DBMS_OUTPUT.PUT_LINE.



Edit with WPS Office

Question 5: Designing Pipelined Function for Sales Data

1. SQL QUERY:



2.Explanation

Explanation of Pipelined Table Functions

Pipelined Table Functions are used in Oracle to improve the efficiency of querying large datasets by allowing data to be processed and returned in chunks rather than as a single large result set. Here's how they work:

1. **Data Processing in Chunks:** Instead of processing the entire dataset in one go, pipelined functions return rows incrementally as they are processed. This is especially beneficial for large datasets where processing and returning the entire set at once would be inefficient.
2. **Improved Performance:** By streaming the results as they are produced, pipelined functions reduce memory consumption and improve performance. This method is more efficient for querying and processing large amounts of data as it avoids creating a large intermediate result set.
3. **Integration with SQL Queries:** Pipelined table functions can be used directly in SQL queries. For example, you can use `SELECT * FROM TABLE(get_sales_data(5, 2023))` to retrieve sales data for May 2023. This integration makes it easy to use pipelined functions as part of regular SQL operations.





Edit with WPS Office