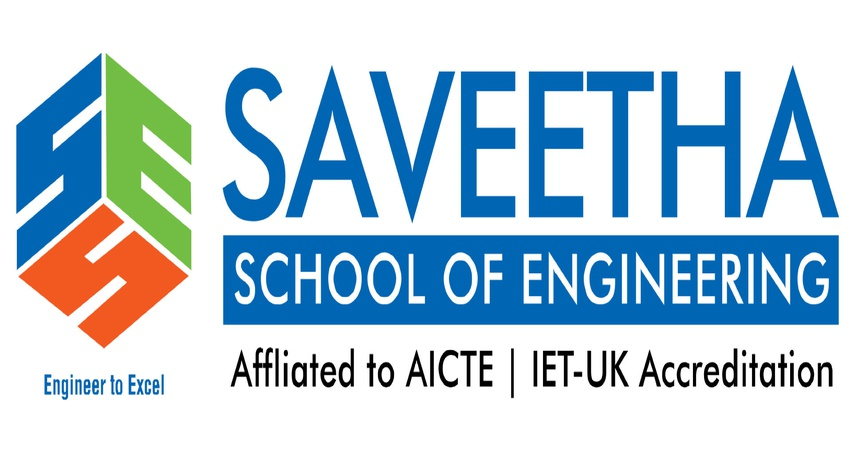
****

**Assignment - 3**

**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**

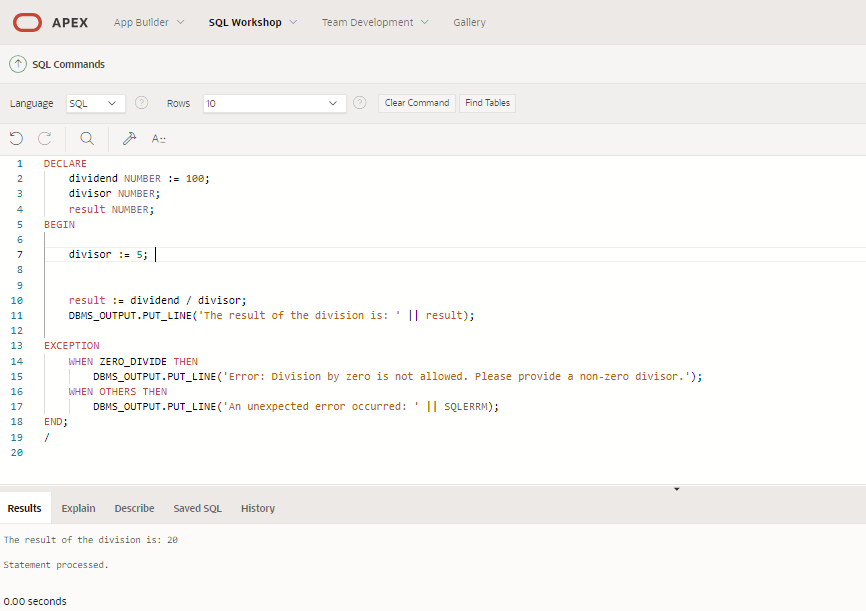
**Question 1: Handling Division Operation**

**Task:**

**Write a PL/SQL block to perform a division operation where the divisor is obtained from user input. Handle the ZERO\_DIVIDE exception gracefully with an appropriate error message.**

QUERY:

IMPLEMENTATION:

****

**Explanation of Error Handling:**

1. **ZERO\_DIVIDE Exception Handling**:
   * The ZERO\_DIVIDE exception is explicitly handled using the WHEN ZERO\_DIVIDE THEN clause. This exception occurs when the divisor is zero. The program catches this exception and outputs a user-friendly error message, informing the user that division by zero is not allowed.
2. **OTHERS Exception Handling**:
   * The WHEN OTHERS THEN clause is a catch-all for any other exceptions that might occur. This ensures that the program can handle unexpected errors gracefully. The SQLERRM function is used to display the error message associated with the exception.
3. **User Input Handling**:
   * In a real application, user input would typically be handled through a front-end interface. In this example, &divisor is used as a placeholder to represent user input. Ensure that your application captures and validates this input appropriately.

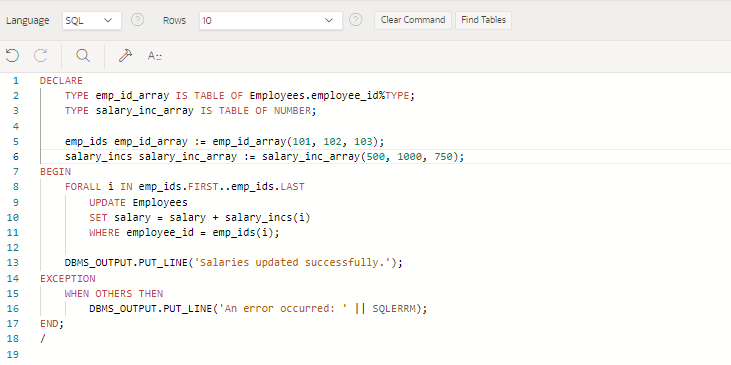
This PL/SQL block demonstrates how to handle exceptions in a structured manner, providing clear feedback to the user and ensuring the program can continue running or terminate gracefully in case of errors.

**Question 2: Updating Rows with FORALL**

**Task:**

**Use the FORALL statement to update multiple rows in the Employees table based on arrays of employee IDs and salary increments.**

**QUERY:**

****

**Explanation:  
Bulk Processing with FORALL:**

* + **T**he FORALL statement in PL/SQL is used for bulk DML operations (INSERT, UPDATE, DELETE) on collections.
  + It allows you to execute a DML statement for each element in a collection with a single context switch between the PL/SQL and SQL engines.
  + This is particularly useful when dealing with large data sets, as it reduces the number of context switches and thereby improves performance.

1. **Improved Performance:**
   * Reduced Context Switching: In a typical loop, each DML statement execution requires a context switch between the PL/SQL and SQL engines. This can be costly in terms of performance, especially when dealing with a large number of records. FORALL minimizes these context switches by executing the DML operation in bulk.
   * Efficient Array Processing: FORALL processes entire arrays in one go, rather than row-by-row, which is more efficient. This is particularly beneficial when the operation involves a large number of rows, as it can significantly reduce the execution time.
   * Optimized Network Usage: When the PL/SQL engine and the SQL engine are on different systems, FORALL can reduce network traffic by sending fewer messages between the client and the database server.

Overall, using FORALL for bulk updates can lead to substantial performance improvements in applications that need to perform large-scale data modifications.

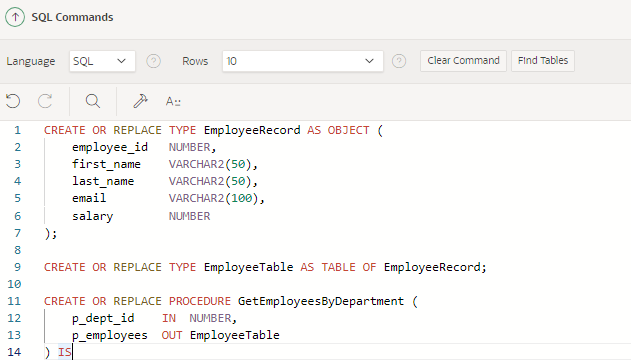
**Question 3: Implementing Nested Table Procedure**

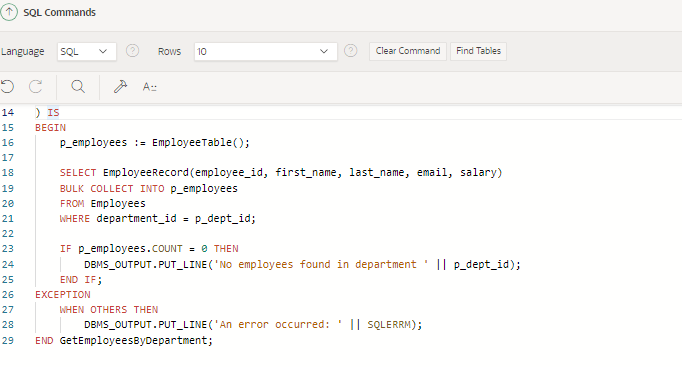
**Task:**

**Implement a PL/SQL procedure that accepts a department ID as input, retrieves employees belonging to the department, stores them in a nested table type, and returns this collection as an output parameter.**

QUERY:

Table Creation:



Implementation:  


**Explanation:**

1. **Nested Table Type Definition**:
   * The EmployeeRecord object type is defined to store individual employee details, including employee\_id, first\_name, last\_name, email, and salary.
   * The EmployeeTable type is a nested table that stores a collection of EmployeeRecord objects. This nested table type allows for storing and handling multiple employee records as a single collection.
2. **Procedure GetEmployeesByDepartment**:
   * **Input Parameter (p\_dept\_id)**: This parameter accepts the department ID to filter the employees.
   * **Output Parameter (p\_employees)**: This parameter is used to return the collection of employees belonging to the specified department.
   * **Fetching Employees**: The procedure uses a SELECT statement with BULK COLLECT INTO to fetch all employee records that match the given department ID. The results are stored in the p\_employees nested table.
   * **Handling No Employees Found**: If no employees are found for the specified department, a message is printed using DBMS\_OUTPUT.PUT\_LINE.

### **Utilization and Returning of Nested Tables**

1. **Nested Tables as Output**:
   * Nested tables are a PL/SQL collection type that can be used to store multiple records or elements. They are similar to arrays but can be sparse and allow for easy manipulation, such as adding or deleting elements.
   * In this procedure, the nested table p\_employees is populated with employee records and returned as an output parameter. This allows the calling environment to receive and process a collection of employee data in a structured format.
2. **Advantages of Nested Tables**:
   * Nested tables provide a flexible way to handle sets of data in PL/SQL. They can be used in bulk operations, such as BULK COLLECT, and can be passed between procedures and functions.
   * They are also useful for returning multiple rows of data from a procedure, as demonstrated in this example, where all employees from a specific department are returned as a single collection.

This procedure and the use of nested tables provide an efficient way to handle and return sets of related data in PL/SQL applications.

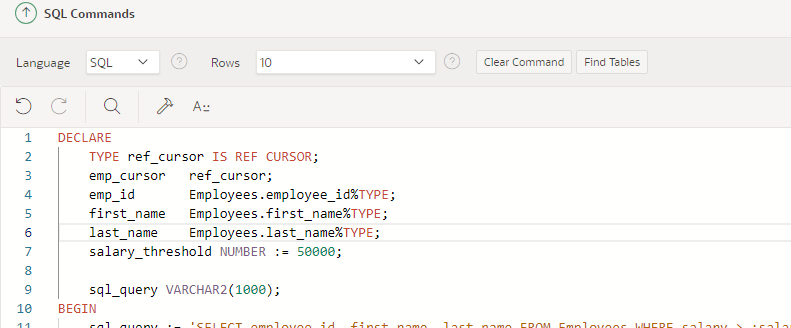
**Question 4: Using Cursor Variables and Dynamic SQL**

**Task:**

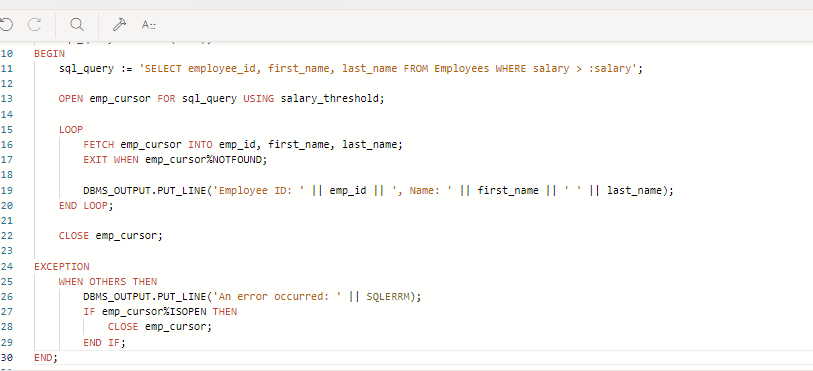
**Write a PL/SQL block demonstrating the use of cursor variables (REF CURSOR) and dynamic SQL. Declare a cursor variable for querying EmployeeID, FirstName, and LastName based on a specified salary threshold.**

QUERY:

TABLE CREATION:



IMPLEMENTATION:

**  
EXPLANATION:**

1. **Dynamic SQL**:
   * Dynamic SQL refers to SQL statements that are constructed and executed at runtime rather than being hard-coded. This allows for more flexibility, such as executing different queries based on user input or other runtime conditions.
   * In this example, the dynamic SQL query is constructed as a string in the sql\_query variable, which selects employee\_id, first\_name, and last\_name from the Employees table where the salary is above a certain threshold.
2. **Cursor Variables (REF CURSOR)**:
   * A cursor variable, defined using the REF CURSOR type, is a pointer to a result set that can be dynamically opened and closed. It allows for more flexible query execution, especially when the exact query is not known until runtime.
   * In this block, emp\_cursor is declared as a cursor variable of type ref\_cursor. The cursor is opened using the dynamic SQL query stored in sql\_query with the salary\_threshold passed as a bind variable using the USING clause.
3. **Using the Cursor**:
   * Once the cursor is open, the program fetches the results into local variables (emp\_id, first\_name, last\_name) using a loop. The FETCH statement retrieves one row at a time, and EXIT WHEN emp\_cursor%NOTFOUND exits the loop when there are no more rows to fetch.
   * After processing the result set, the cursor is closed using the CLOSE statement to release the associated resources.
4. **Handling Exceptions**:
   * The EXCEPTION block ensures that any errors during the execution are caught and handled appropriately. If an error occurs and the cursor is still open, it is closed within the exception handler to prevent resource leakage.

Dynamic SQL, combined with cursor variables, provides powerful capabilities in PL/SQL, enabling flexible and dynamic querying of data. This is particularly useful in scenarios where the exact query structure depends on user input or other runtime conditions.