Phase 7: Advanced Database Programming and Auditing

a) Problem Statement Development

Problem Statement: The project aims to streamline the collection, analysis, and reporting of climate-related data while ensuring data integrity, accuracy, and security. The complexity of handling large datasets requires advanced database programming techniques to ensure real-time updates, automation, and accountability.

Justification for Techniques:

- **Triggers:** To enforce business rules like preventing invalid data or ensuring data integrity.
- **Cursors:** For processing large datasets row by row, e.g., calculating averages or generating climate reports.
- **Functions:** For modularizing logic like calculating averages or performing specific climate-related calculations.
- **Packages:** For grouping related functions and procedures for better organization and performance.
- **Auditing:** To track changes to sensitive data, providing accountability and security.

---

b) Trigger Implementation

Simple Trigger Example:

```
CREATE OR REPLACE TRIGGER validate_temp_insert
BEFORE INSERT ON climate_data
FOR EACH ROW
BEGIN
  IF :NEW.temperature < -100 OR :NEW.temperature > 60 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Temperature value out of range');
  END IF;
END;
```

Compound Trigger Example:

```
CREATE OR REPLACE TRIGGER climate_data_trigger
FOR INSERT OR UPDATE ON climate_data
COMPOUND TRIGGER
  AFTER INSERT OR UPDATE ON climate_data
IS
BEGIN
  INSERT INTO change_log (action, table_name, record_id)
  VALUES ('Insert/Update', 'climate_data', :NEW.id);
END AFTER;
END climate_data_trigger;
```

---

c) Cursor Usage

Cursor Example:

```
DECLARE
  CURSOR climate_cursor IS
    SELECT id, temperature, humidity FROM climate_data WHERE region = 'North';
  v_temperature climate_data.temperature%TYPE;
  v_humidity climate_data.humidity%TYPE;
BEGIN
  FOR record IN climate_cursor LOOP
    v_temperature := record.temperature;
    v_humidity := record.humidity;
    DBMS_OUTPUT.PUT_LINE('Temperature: ' || v_temperature || ', Humidity: ' ||
v_humidity);
  END LOOP;
END;
```

---

d) Attributes and Functions

Attributes Example:

```
DECLARE
   v_temp climate_data.temperature%TYPE;
   v_humidity climate_data.humidity%TYPE;
BEGIN
   SELECT temperature, humidity INTO v_temp, v_humidity FROM climate_data WHERE id = 1;
   DBMS_OUTPUT.PUT_LINE('Temperature: ' || v_temp || ', Humidity: ' || v_humidity);
END;
```

Function Example:

```
CREATE OR REPLACE FUNCTION get_avg_temp(region IN VARCHAR2)
RETURN NUMBER IS
   avg_temp NUMBER;
BEGIN
   SELECT AVG(temperature) INTO avg_temp
   FROM climate_data
   WHERE region = region;
   RETURN avg_temp;
END;
```

---

e) Package Development

Package Example:

```
CREATE OR REPLACE PACKAGE climate_pkg AS
   FUNCTION get_avg_temp(region IN VARCHAR2) RETURN NUMBER;
   PROCEDURE log_data_change(action IN VARCHAR2, data_id IN NUMBER);
END climate_pkg;


CREATE OR REPLACE PACKAGE BODY climate_pkg AS
```

```
  FUNCTION get_avg_temp(region IN VARCHAR2) RETURN NUMBER IS
    avg_temp NUMBER;
  BEGIN
    SELECT AVG(temperature) INTO avg_temp
    FROM climate_data
    WHERE region = region;
    RETURN avg_temp;
  END;

  PROCEDURE log_data_change(action IN VARCHAR2, data_id IN NUMBER) IS
  BEGIN
    INSERT INTO change_log (action, data_id)
    VALUES (action, data_id);
  END;
END climate_pkg;
```

---

f) Auditing with Restrictions and Tracking

Auditing Example:

```
CREATE OR REPLACE TRIGGER audit_climate_data_changes
AFTER UPDATE OR DELETE ON climate_data
FOR EACH ROW
BEGIN
  INSERT INTO audit_log (action, old_temperature, new_temperature, action_time)
  VALUES ('UPDATE', :OLD.temperature, :NEW.temperature, SYSDATE);
END;
```

Restrictions Example:

```
CREATE OR REPLACE TRIGGER restrict_temp_update
BEFORE UPDATE ON climate_data
```

```
FOR EACH ROW
BEGIN
  IF NOT (USER = 'admin') THEN
    RAISE_APPLICATION_ERROR(-20002, 'Only admin can update temperature data.');
  END IF;
END;
```

---

g) Scope and Limitations

Scope:

- **Triggers:**
  - **Before Insert Trigger:** Ensures that only valid temperature values are entered into the database. This trigger is designed to prevent inserting unrealistic temperature values (below -100°C or above 60°C).
  - **Compound Trigger:** Used to capture both INSERT and UPDATE events to log the changes into an audit log. This helps track changes to critical climate data for accountability and auditing purposes.
- **Cursors:**
  - **Explicit Cursors:** Cursors are used to process rows one by one, which is particularly beneficial when generating reports for specific regions, such as calculating the average temperature or reporting the temperature and humidity of specific records.
- **Attributes and Functions:**
  - **Attributes (e.g., %TYPE):** Used to declare variables that are of the same type as the columns in the table. This enhances reusability and ensures the code remains synchronized with the underlying table structure.
  - **Functions:** Functions are designed to modularize logic such as calculating the average temperature for a given region, promoting reusability and simplicity in the codebase.
- **Packages:**
  - **Climate Data Package:** A package is created to group related functions and procedures. This makes the codebase easier to manage, improves performance, and ensures that the logic for handling climate data is modular and reusable across the application.

- **Auditing and Restrictions:**
    - **Auditing:** A trigger is used to log updates and deletions on sensitive climate data into an audit table. This provides traceability for each change, which is crucial for security and data integrity.
    - **Restrictions:** A trigger is used to restrict updates to climate data, only allowing certain users (e.g., admin) to update temperature values. This ensures that critical data cannot be altered by unauthorized users.

Limitations:

- **Performance Impact:**
    - While triggers provide valuable business rule enforcement and auditing, they may introduce performance overhead, especially when dealing with large volumes of data. For example, the compound trigger might slow down transactional operations if there are many concurrent updates or inserts.
    - The use of explicit cursors in scenarios where bulk processing could be more efficient might also impact performance.
- **Complexity in Debugging:**
    - Triggers and compound triggers, especially when dealing with multiple actions (like INSERT, UPDATE), can be difficult to debug and maintain, particularly when the triggers interact with each other.
- **Limited User Control:**
    - Auditing and restriction mechanisms might limit the flexibility of users in performing certain actions. For example, restricting data updates to admin users could hinder other authorized roles that might need to perform such updates.
- **Scalability:**
    - As the volume of climate data grows, the current design might not scale effectively without additional performance optimizations (e.g., bulk operations or partitioning for large datasets).
- **Security:**
    - While auditing ensures tracking of changes, it also requires proper management of audit logs and potential extra resources for monitoring access and ensuring compliance with security policies.

---

h) Documentation and Demonstration

Documentation:

- **Design:**
    - The database design includes several critical tables for storing climate data, including temperature and humidity readings, and audit logs for tracking changes to sensitive data.
    - Triggers are employed to ensure that only valid data is inserted and to capture updates and deletions for auditing purposes.
    - Explicit cursors are utilized in cases where row-by-row processing is needed for generating reports or performing specific calculations.
    - A package (climate_pkg) is created to group the functions and procedures related to climate data handling, improving code organization and reusability.
- **Implementation:**
    - **Triggers:**
        - **validate_temp_insert:** Prevents inserting temperature values outside the valid range.
        - **climate_data_trigger:** Logs changes to climate data whenever a record is inserted or updated.
        - **audit_climate_data_changes:** Audits updates or deletions on climate data and logs them in the audit_log table.
        - **restrict_temp_update:** Restricts temperature updates to admin users only.
    - **Cursors:**
        - An explicit cursor is implemented to fetch climate data from the climate_data table for a specific region (e.g., "North"). Each row is processed individually, and output is displayed using DBMS_OUTPUT.PUT_LINE.
    - **Functions:**
        - get_avg_temp: A function that calculates the average temperature for a given region. This function is used to provide insights into regional climate trends.
    - **Packages:**
        - **climate_pkg**: A PL/SQL package is created to group related functions and procedures. This package includes get_avg_temp and log_data_change, providing an efficient way to handle climate data operations and ensure modularity.
    - **Auditing and Restrictions:**
        - The audit_climate_data_changes trigger ensures that all updates or deletions are tracked, and the changes are logged for security and accountability.

        - The restrict_temp_update trigger ensures that only admin users can update the temperature, maintaining data security.

- **Testing and Demonstration:**
    - During testing, the system will demonstrate how business rules and restrictions are enforced in real-time. For example:
        - Attempting to insert an invalid temperature (e.g., below -100°C) will trigger an error.
        - Updating a temperature value will only be allowed by an admin user.
        - The audit log will capture all changes to climate data, demonstrating accountability.
    - The demonstration will also show how the system can generate reports, such as the average temperature for a given region, using the explicit cursor and function designed for this purpose.