Experiment 3: Stored Procedures, Functions & Triggers

Prerequisites

Knowledge of SQL queries and basic database concepts

Lab Objective

To understand and implement database programming concepts using stored procedures, user-defined functions, and triggers to enforce business logic and automate database operations.

Learning Outcomes

By completing this experiment, students will be able to:

- Create and execute stored procedures.
- Develop user-defined functions.
- Use input and output parameters.
- Implement triggers for automatic database actions.
- Enforce business rules at the database level.
- Understand the importance of database-level automation.

Introduction

Modern relational database systems provide procedural extensions that allow developers to implement logic directly within the database server. Stored procedures, functions, and triggers centralize business rules, improve performance, enhance security, and reduce redundancy in application code.

This experiment introduces students to database programming techniques widely used in enterprise systems such as banking, ERP, and academic management systems.

Key Concepts Covered

- **Stored Procedures:** Precompiled SQL statements used to perform operations.
- **User-Defined Functions:** Reusable routines that return computed values.
- **Parameters:** IN, OUT, and INOUT parameters for passing values.
- **Triggers:** Automatic programs executed during INSERT, UPDATE, or DELETE events.
- **Business Rule Enforcement:** Ensuring data validity and integrity at database level.

Experiment Steps

---

1. Create Stored Procedures

Develop procedures to perform operations such as inserting or retrieving records.

Example – Stored Procedure

```
CREATE PROCEDURE GetStudentByID(IN sid INT)

BEGIN

   SELECT * FROM Students

   WHERE student_id = sid;

END;
```

Execute Procedure

```
CALL GetStudentByID(1);
```

Students should observe how parameters are passed and how procedures execute predefined logic.

---

2. Create User-Defined Functions

Create functions that return calculated values and use them within queries.

Example – Function to Calculate Bonus Marks

```
CREATE FUNCTION AddBonus(marks INT)

RETURNS INT

DETERMINISTIC

BEGIN

    RETURN marks + 5;

END;
```

Use Function in Query

```
SELECT student_id, AddBonus(marks) AS Updated_Marks

FROM Enrollments;
```

---

3. Implement Triggers

Triggers execute automatically in response to specific database events.

Example – BEFORE INSERT Trigger

```
CREATE TRIGGER CheckAge

BEFORE INSERT ON Students
```

```
FOR EACH ROW
BEGIN
   IF NEW.age < 17 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Age must be 17 or above';
   END IF;
END;
```

Example – AFTER UPDATE Trigger (Logging)

```
CREATE TRIGGER LogUpdate
AFTER UPDATE ON Students
FOR EACH ROW
BEGIN
   INSERT INTO AuditLog(student_id, action_date)
   VALUES (NEW.student_id, NOW());
END;
```

Students should test the automatic execution of triggers during insert or update operations.

---

4. Test Business Logic Implementation

- Insert records to verify trigger constraints.
- Update records and observe automatic logging behavior.
- Compare manual execution versus automated database-level enforcement.

How It Works

- Stored procedures encapsulate logic for reuse and efficiency.
- Functions allow reusable computation inside SQL statements.
- Triggers execute automatically when database events occur.
- Database programming ensures centralized control and consistent enforcement of business rules.

Submission Instructions

Students must submit:

- Screenshot of stored procedure creation and execution.
- Output demonstrating function usage in a SELECT query.
- Screenshot of trigger creation.
- Example showing automatic trigger execution.

Conclusion

This experiment introduces procedural programming within relational databases. Mastery of stored procedures, functions, and triggers is essential for building secure, efficient, and enterprise-grade database applications.