# Green University of Bangladesh
# Department of Computer Science and Engineering (CSE)

**Faculty of Sciences and Engineering**

**Semester: (Fall, Year:2024), B.Sc. in CSE (Day)**

**Assignment: 02**

**Course Title:** Database

**Course Code: CSE 209 Section: 231_D1**

## Student Details

| | Name | ID |
|---|---|---|
| **1.** | Promod Chandra Das | 231002005 |

**Submission Date : 20.12.2024**

**Course Teacher's Name : Dr. Faiz Al Faisal**

**Lab Report Status**

**Marks: ……** **Signature:.....**

**Comments:.....** **Date:.....**

## 1. OBJECTIVES :

1. **Understand Database Automation**
   - Learn how to create **triggers** that respond to specific database events (e.g., INSERT, UPDATE, DELETE).
   - Understand how **functions** can encapsulate reusable logic that triggers can invoke.
2. **Enhance Data Integrity**
   - Automatically enforce rules and constraints (e.g., keeping logs, preventing invalid operations).
   - Ensure that critical business processes are executed without manual intervention.
3. **Practice Advanced SQL Concepts**
   - Gain proficiency in procedural SQL by writing functions.
   - Understand how triggers and functions work together to automate backend processes.
4. **Implement Real-World Use Cases**
   - Simulate scenarios where triggers and functions are used for:
     - Auditing changes in data (e.g., stock updates, salary changes).
     - Ensuring consistency across tables (e.g., cascading updates, validations).
     - Automating notifications or alerts.
5. **Demonstrate Practical Skills**
   - Showcase the ability to design a database with advanced features.
   - Highlight skills useful in real-world database administration and application development.

- Step 1: Creating Employees and Orders tables

SQL Query:

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   EmployeeName VARCHAR(100),
   Position VARCHAR(50),
   CYrSalary DECIMAL(10, 2),
   PYrSalary DECIMAL(10, 2)
);

CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
```

```
    EmployeeID INT,
    OrderDate DATE,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
```

**Explanation:**

- **Employees table contains details about employees, including their current and previous year salaries.**
- **Orders table tracks orders and associates them with employees using a foreign key.**

---

**Step 2: Inserting random data into Employees and Orders tables**

**SQL Query:**

**-- Inserting random employees**

```
INSERT INTO Employees (EmployeeID, EmployeeName, Position, CYrSalary, PYrSalary)
VALUES
(1, 'Alice', 'Manager', 50000.00, 50000.00),
(2, 'Bob', 'Sales', 40000.00, 40000.00),
(3, 'Charlie', 'Developer', 45000.00, 45000.00);
```

**-- Inserting random orders**

```
INSERT INTO Orders (OrderID, EmployeeID, OrderDate)
VALUES
(1, 1, '2023-01-15'),
(2, 1, '2023-01-20'),
(3, 2, '2023-02-15'),
(4, 2, '2023-03-10'),
(5, 2, '2023-03-15'),
(6, 3, '2023-04-01'),
(7, 3, '2023-04-05'),
(8, 3, '2023-05-01'),
(9, 3, '2023-05-20'),
(10, 3, '2023-06-01');
```

**Explanation:**

- **Three employees are added with their salaries initialized in both CYrSalary and PYrSalary.**
- **Orders are associated with employees to track their order counts.**

---

**Step 3: Writing a procedure to calculate salary increment**

**SQL Query:**

```
CREATE PROCEDURE IncrementSalary(
   IN EmployeeID INT,
   IN Year INT,
   IN IncrementRate DECIMAL(10, 2)
)
BEGIN
   DECLARE OrderCount INT;
   DECLARE IncrementAmount DECIMAL(10, 2);
```

**-- Calculate the total number of orders for the employee in the given year**

```
   SELECT COUNT(*) INTO OrderCount
   FROM Orders
   WHERE EmployeeID = EmployeeID AND YEAR(OrderDate) = Year;
```

**-- Calculate the salary increment (2% for every 10 orders)**

```
   SET IncrementAmount = FLOOR(OrderCount / 10) * IncrementRate;
```

**-- Update the current year salary**

```
   UPDATE Employees
   SET CYrSalary = PYrSalary + (PYrSalary * IncrementAmount / 100)
   WHERE EmployeeID = EmployeeID;
END $$
```

**DELIMITER ;**

**Explanation:**

- **The procedure IncrementSalary calculates the salary increment based on the number of orders created by an employee in a specific year.**
- **For every 10 orders, a 2% increment is applied.**

---

**Step 4: Running the procedure and checking the updated salary**

**SQL Query:**

**-- Example: Increment salary for employee ID 3 for the year 2023**

```
CALL IncrementSalary(3, 2023, 2.00);
```

**-- Check the updated salary**

```
SELECT * FROM Employees;
```

**Explanation:**

- **The procedure is called for EmployeeID 3 for the year 2023.**
- **After running the procedure, CYrSalary will reflect the updated salary.**

---

**Expected Output after running the procedure**

**The Employees table will show the updated CYrSalary for the employee. For example:**

| EmployeeID | EmployeeName | Position | CYrSalary | PYrSalary |
|---|---|---|---|---|
| 1 | Alice | Manager | 50000.00 | 50000.00 |
| 2 | Bob | Sales | 40000.00 | 40000.00 |
| 3 | Charlie | Developer | 45900.00 | 45000.00 |

**Explanation:**

- **For EmployeeID 3, there are 5 orders in 2023, which gives one increment (5 orders / 10 ≈ 0.5 → rounded to 0 increments).**
- **The salary increases accordingly based on the calculation logic.**

➢ **DISCUSSION :**

Triggers and functions in databases enable automation, enforce business rules, and maintain data integrity. Triggers act on events like INSERT or UPDATE, while functions encapsulate reusable logic. Together, they streamline workflows, such as logging changes or validating data. Proper design ensures efficiency, but overuse can lead to performance issues and complexity.