Exercise 1: Control Structures

Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

- **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE ApplyInterestDiscount()
    -> BEGIN
    ->     UPDATE Loans l
    ->     JOIN Customers c ON l.CustomerID = c.CustomerID
    ->     SET l.InterestRate = l.InterestRate - 1
    ->     WHERE c.Age > 60;
    -> END;
    -> //
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the procedure:
mysql> CALL ApplyInterestDiscount();
Query OK, 2 rows affected (0.02 sec)

mysql>
```

```
mysql> SELECT * FROM Loans;
+--------+------------+--------------+------------+
| LoanID | CustomerID | InterestRate | DueDate    |
+--------+------------+--------------+------------+
|    101 |          1 |         3.50 | 2025-07-09 |
|    102 |          2 |         6.00 | 2025-08-08 |
|    103 |          3 |         3.00 | 2025-07-04 |
|    104 |          4 |         4.50 | 2025-06-19 |
+--------+------------+--------------+------------+
4 rows in set (0.00 sec)

mysql>
```

Scenario 2: A customer can be promoted to VIP status based on their balance.

- **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE PromoteToVIP()
    -> BEGIN
    ->     UPDATE Customers
    ->     SET IsVIP = 'Y'
    ->     WHERE Balance > 10000;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the procedure:
mysql> CALL PromoteToVIP();
Query OK, 2 rows affected (0.02 sec)

mysql>
```

```
mysql> CALL PromoteToVIP();
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Customers;
+------------+---------------+------+----------+-------+
| CustomerID | Name          | Age  | Balance  | IsVIP |
+------------+---------------+------+----------+-------+
|          1 | John Smith    |   65 | 12000.00 | Y     |
|          2 | Jane Doe      |   45 |  8000.00 | N     |
|          3 | Alice Johnson |   70 |  5000.00 | N     |
|          4 | Bob Lee       |   35 | 15000.00 | Y     |
+------------+---------------+------+----------+-------+
4 rows in set (0.00 sec)

mysql>
```

Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.

- **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE LoanReminders()
    -> BEGIN
    ->     SELECT l.LoanID, c.Name, l.DueDate
    ->     FROM Loans l
    ->     JOIN Customers c ON l.CustomerID = c.CustomerID
    ->     WHERE l.DueDate BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql>
mysql> -- Call the procedure:
mysql> CALL LoanReminders();
+--------+---------------+------------+
| LoanID | Name          | DueDate    |
+--------+---------------+------------+
|    101 | John Smith    | 2025-07-09 |
|    103 | Alice Johnson | 2025-07-04 |
+--------+---------------+------------+
2 rows in set (0.00 sec)

Query OK, 0 rows affected (0.02 sec)

mysql>
```

Exercise 2: Error Handling

Scenario 1: Handle exceptions during fund transfers between accounts.

- **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE SafeTransferFunds(
    ->     IN fromAcc INT,
    ->     IN toAcc INT,
    ->     IN amount DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     DECLARE EXIT HANDLER FOR SQLEXCEPTION
    ->     BEGIN
    ->         ROLLBACK;
    ->         INSERT INTO ErrorLog(ErrorMessage) VALUES ('Transfer failed due to an error or insufficient funds.');
    ->     END;
    ->
    ->     START TRANSACTION;
    ->
    ->     -- Check if from account has enough funds
    ->     IF (SELECT Balance FROM Accounts WHERE AccountID = fromAcc) < amount THEN
    ->         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
    ->     END IF;
    ->
    ->     -- Deduct from sender
    ->     UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;
    ->
    ->     -- Add to receiver
    ->     UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAcc;
    ->
    ->     COMMIT;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL SafeTransferFunds(1, 2, 300.00); -- should succeed
Query OK, 0 rows affected (0.02 sec)

mysql> CALL SafeTransferFunds(2, 1, 1000.00); -- should fail and log error
Query OK, 1 row affected (0.02 sec)

mysql> select * from Accounts;
+-----------+---------------+---------+
| AccountID | AccountHolder | Balance |
+-----------+---------------+---------+
|         1 | Alice         |  700.00 |
|         2 | Bob           |  800.00 |
+-----------+---------------+---------+
2 rows in set (0.00 sec)
```

Scenario 2: Manage errors when updating employee salaries.

- **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE UpdateSalary(
    ->     IN empID INT,
    ->     IN percentage DECIMAL(5, 2)
    -> )
    -> BEGIN
    ->     DECLARE EXIT HANDLER FOR SQLEXCEPTION
    ->     BEGIN
    ->         INSERT INTO ErrorLog(ErrorMessage)
    ->         VALUES (CONCAT('Error updating salary for EmployeeID = ', empID));
    ->     END;
    ->
    ->     -- Try to update salary
    ->     UPDATE Employees
    ->     SET Salary = Salary + (Salary * percentage / 100)
    ->     WHERE EmployeeID = empID;
    ->
    ->     IF ROW_COUNT() = 0 THEN
    ->         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Employee not found';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL UpdateSalary(101, 10);  -- should succeed
Query OK, 1 row affected (0.01 sec)

mysql> CALL UpdateSalary(999, 5);   -- should fail and log error
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Employees;
+------------+------------+----------+
| EmployeeID | Name       | Salary   |
+------------+------------+----------+
|        101 | John Doe   | 66000.00 |
|        102 | Jane Smith | 75000.00 |
+------------+------------+----------+
2 rows in set (0.00 sec)

mysql>
```

Scenario 3: Ensure data integrity when adding a new customer.

- **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE AddNewCustomer(
    ->     IN custID INT,
    ->     IN custName VARCHAR(100),
    ->     IN custAge INT,
    ->     IN custBalance DECIMAL(10,2)
    -> )
    -> BEGIN
    ->     DECLARE EXIT HANDLER FOR SQLEXCEPTION
    ->     BEGIN
    ->         INSERT INTO ErrorLog(ErrorMessage)
    ->         VALUES (CONCAT('Customer insert failed: ID ', custID, ' already exists.'));
    ->     END;
    ->
    ->     INSERT INTO Customers(CustomerID, Name, Age, Balance)
    ->     VALUES (custID, custName, custAge, custBalance);
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL AddNewCustomer(3, 'Sophia', 28, 5000.00); -- should succeed
Query OK, 1 row affected (0.01 sec)

mysql> CALL AddNewCustomer(1, 'Emily', 30, 3000.00);  -- should fail and log error
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Customers;
+------------+--------+------+---------+
| CustomerID | Name   | Age  | Balance |
+------------+--------+------+---------+
|          1 | Emily  |   30 | 3000.00 |
|          2 | David  |   40 | 4000.00 |
|          3 | Sophia |   28 | 5000.00 |
+------------+--------+------+---------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM ErrorLog;
+-------+---------------------+-----------------------------------------------------+
| LogID | ErrorTime           | ErrorMessage                                        |
+-------+---------------------+-----------------------------------------------------+
|     1 | 2025-06-24 10:18:51 | Transfer failed due to an error or insufficient funds. |
|     2 | 2025-06-24 10:20:51 | Error updating salary for EmployeeID = 999          |
|     3 | 2025-06-24 10:22:05 | Customer insert failed: ID 1 already exists.        |
+-------+---------------------+-----------------------------------------------------+
3 rows in set (0.00 sec)
```

Exercise 3: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

- **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

```
mysql> CREATE TABLE SavingsAccounts (
    ->     AccountID INT PRIMARY KEY,
    ->     CustomerID INT,
    ->     Balance DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO SavingsAccounts VALUES (1, 101, 1000.00);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO SavingsAccounts VALUES (2, 102, 2000.00);
Query OK, 1 row affected (0.00 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE ProcessMonthlyInterest()
    -> BEGIN
    ->     UPDATE SavingsAccounts
    ->     SET Balance = Balance + (Balance * 0.01);
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL ProcessMonthlyInterest();
Query OK, 2 rows affected (0.02 sec)

mysql> SELECT * FROM SavingsAccounts;
+-----------+------------+---------+
| AccountID | CustomerID | Balance |
+-----------+------------+---------+
|         1 |        101 | 1010.00 |
|         2 |        102 | 2020.00 |
+-----------+------------+---------+
2 rows in set (0.00 sec)

mysql>
```

Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

- **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE UpdateEmployeeBonus(
    ->     IN deptName VARCHAR(50),
    ->     IN bonusPercent DECIMAL(5, 2)
    -> )
    -> BEGIN
    ->     UPDATE Employees
    ->     SET Salary = Salary + (Salary * bonusPercent / 100)
    ->     WHERE Department = deptName;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL UpdateEmployeeBonus('Finance', 10);
ERROR 1054 (42S22): Unknown column 'Department' in 'where clause'
mysql> SELECT * FROM Employees;
+------------+------------+----------+
| EmployeeID | Name       | Salary   |
+------------+------------+----------+
|        101 | John Doe   | 66000.00 |
|        102 | Jane Smith | 75000.00 |
+------------+------------+----------+
2 rows in set (0.02 sec)

mysql>
```

Scenario 3: Customers should be able to transfer funds between their accounts.

- **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE TransferFunds(
    ->     IN fromAcc INT,
    ->     IN toAcc INT,
    ->     IN amount DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     DECLARE fromBalance DECIMAL(10,2);
    ->
    ->     SELECT Balance INTO fromBalance FROM Accounts WHERE AccountID = fromAcc;
    ->
    ->     IF fromBalance IS NULL THEN
    ->         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Source account not found';
    ->     ELSEIF fromBalance < amount THEN
    ->         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
    ->     ELSE
    ->         START TRANSACTION;
    ->             UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;
    ->             UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAcc;
    ->         COMMIT;
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL TransferFunds(1, 2, 200.00); -- should succeed
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Accounts;
+-----------+---------------+---------+
| AccountID | AccountHolder | Balance |
+-----------+---------------+---------+
|         1 | Alice         |  500.00 |
|         2 | Bob           | 1000.00 |
+-----------+---------------+---------+
2 rows in set (0.00 sec)

mysql>
```

Exercise 4: Functions

Scenario 1: Calculate the age of customers for eligibility checks.

- **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION CalculateAge(dob DATE) RETURNS INT
    -> DETERMINISTIC
    -> BEGIN
    ->     RETURN TIMESTAMPDIFF(YEAR, dob, CURDATE());
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
```

```
mysql> SELECT Name, CalculateAge(DateOfBirth) AS Age FROM Customers;
+--------+------+
| Name   | Age  |
+--------+------+
| Emily  |   35 |
| David  | NULL |
| Sophia | NULL |
+--------+------+
3 rows in set (0.00 sec)

mysql>
```

Scenario 2: The bank needs to compute the monthly installment for a loan.

- **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION CalculateMonthlyInstallment(
    ->     loanAmount DECIMAL(10, 2),
    ->     annualRate DECIMAL(5, 2),
    ->     years INT
    -> ) RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE r DECIMAL(10,6);
    ->     DECLARE n INT;
    ->     DECLARE emi DECIMAL(10,2);
    ->
    ->     SET r = annualRate / 12 / 100;
    ->     SET n = years * 12;
    ->
    ->     SET emi = loanAmount * r * POW(1 + r, n) / (POW(1 + r, n) - 1);
    ->
    ->     RETURN emi;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT CalculateMonthlyInstallment(100000, 7.5, 5) AS EMI;
+---------+
| EMI     |
+---------+
| 2003.79 |
+---------+
1 row in set (0.01 sec)
```

Scenario 3: Check if a customer has sufficient balance before making a transaction.

- **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION HasSufficientBalance(accID INT, amt DECIMAL(10,2)) RETURNS BOOLEAN
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE bal DECIMAL(10,2);
    ->     SELECT Balance INTO bal FROM Accounts WHERE AccountID = accID;
    ->
    ->     IF bal IS NULL THEN
    ->         RETURN FALSE;
    ->     ELSEIF bal >= amt THEN
    ->         RETURN TRUE;
    ->     ELSE
    ->         RETURN FALSE;
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT HasSufficientBalance(1, 1000.00) AS IsSufficient;
+--------------+
| IsSufficient |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)

mysql> SELECT HasSufficientBalance(2, 1000.00) AS IsSufficient;
+--------------+
| IsSufficient |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)

mysql>
```

Exercise 5: Triggers

Scenario 1: Automatically update the last modified date when a customer's record is updated.

- **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER UpdateCustomerLastModified
    -> BEFORE UPDATE ON Customers
    -> FOR EACH ROW
    -> BEGIN
    ->     SET NEW.LastModified = NOW();
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
mysql> select * from customers;
+------------+--------+------+---------+-------------+---------------------+
| CustomerID | Name   | Age  | Balance | DateOfBirth | LastModified        |
+------------+--------+------+---------+-------------+---------------------+
|          1 | Emily  |   30 | 3000.00 | 1990-06-15  | 2025-06-24 10:35:11 |
|          2 | David  |   40 | 4000.00 | NULL        | 2025-06-24 10:35:11 |
|          3 | Sophia |   28 | 5000.00 | NULL        | 2025-06-24 10:35:11 |
+------------+--------+------+---------+-------------+---------------------+
3 rows in set (0.03 sec)

mysql> _
```

Scenario 2: Maintain an audit log for all transactions.

- **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER LogTransaction
    -> AFTER INSERT ON Transactions
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO AuditLog (TransactionID, Action)
    ->     VALUES (NEW.TransactionID, CONCAT('Inserted ', NEW.Type));
    -> END;
    -> //
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SELECT * FROM Transactions;
+---------------+-----------+------------+---------+---------------------+
| TransactionID | AccountID | Type       | Amount  | TransactionDate     |
+---------------+-----------+------------+---------+---------------------+
|             1 |         1 | deposit    |  200.00 | 2025-06-24 10:38:52 |
|             2 |         2 | withdrawal |  100.00 | 2025-06-24 10:38:56 |
|             3 |         1 | deposit    |  -50.00 | 2025-06-24 10:39:00 |
|             4 |         2 | withdrawal | 1000.00 | 2025-06-24 10:39:05 |
|             5 |         1 | deposit    |  300.00 | 2025-06-24 10:41:04 |
+---------------+-----------+------------+---------+---------------------+
5 rows in set (0.02 sec)

mysql> SELECT * FROM AuditLog;
+-------+---------------+---------------------+---------------------+
| LogID | TransactionID | Action              | LogTime             |
+-------+---------------+---------------------+---------------------+
|     1 |             1 | Inserted deposit    | 2025-06-24 10:38:52 |
|     2 |             2 | Inserted withdrawal | 2025-06-24 10:38:56 |
|     3 |             3 | Inserted deposit    | 2025-06-24 10:39:00 |
|     4 |             4 | Inserted withdrawal | 2025-06-24 10:39:05 |
|     5 |             5 | Inserted deposit    | 2025-06-24 10:41:04 |
+-------+---------------+---------------------+---------------------+
5 rows in set (0.00 sec)

mysql>
```

Scenario 3: Enforce business rules on deposits and withdrawals.

- **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER CheckTransactionRules
    -> BEFORE INSERT ON Transactions
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE accBalance DECIMAL(10,2);
    ->
    ->     -- Get the current balance
    ->     SELECT Balance INTO accBalance FROM Accounts WHERE AccountID = NEW.AccountID;
    ->
    ->     IF NEW.Type = 'withdrawal' AND NEW.Amount > accBalance THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Withdrawal exceeds current balance';
    ->     END IF;
    ->
    ->     IF NEW.Type = 'deposit' AND NEW.Amount <= 0 THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Deposit amount must be positive';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT * FROM Accounts;
+-----------+---------------+---------+
| AccountID | AccountHolder | Balance |
+-----------+---------------+---------+
|         1 | Alice         |  500.00 |
|         2 | Bob           | 1000.00 |
+-----------+---------------+---------+
2 rows in set (0.00 sec)

mysql>
```

Exercise 6: Cursors

Scenario 1: Generate monthly statements for all customers.

- **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

```
mysql> CREATE PROCEDURE GenerateMonthlyStatements()
    -> BEGIN
    ->     DECLARE done INT DEFAULT FALSE;
    ->     DECLARE cID INT;
    ->     DECLARE amt DECIMAL(10,2);
    ->     DECLARE tDate DATE;
    ->     DECLARE cur CURSOR FOR
    ->         SELECT CustomerID, Amount, TransactionDate
    ->         FROM Transactions
    ->         WHERE MONTH(TransactionDate) = MONTH(CURDATE())
    ->         AND YEAR(TransactionDate) = YEAR(CURDATE());
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    ->
    ->     OPEN cur;
    ->     read_loop: LOOP
    ->         FETCH cur INTO cID, amt, tDate;
    ->         IF done THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->         SELECT CONCAT('Customer ', cID, ' had a transaction of ', amt, ' on ', tDate) AS Statement;
    ->     END LOOP;
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CALL GenerateMonthlyStatements();
+-----------------------------------------------------+
| Statement                                           |
+-----------------------------------------------------+
| Customer 1 had a transaction of 200.00 on 2025-06-24 |
+-----------------------------------------------------+
1 row in set (0.00 sec)

+-----------------------------------------------------+
| Statement                                           |
+-----------------------------------------------------+
| Customer 1 had a transaction of -50.00 on 2025-06-24 |
+-----------------------------------------------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.05 sec)

mysql>
```

Scenario 2: Apply annual fee to all accounts.

- **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE ApplyAnnualFee()
    -> BEGIN
    ->     DECLARE done INT DEFAULT FALSE;
    ->     DECLARE accID INT;
    ->     DECLARE fee DECIMAL(10,2) DEFAULT 50.00;
    ->     DECLARE cur CURSOR FOR SELECT AccountID FROM Accounts;
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    ->
    ->     OPEN cur;
    ->     read_loop: LOOP
    ->         FETCH cur INTO accID;
    ->         IF done THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->         UPDATE Accounts SET Balance = Balance - fee WHERE AccountID = accID;
    ->     END LOOP;
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL ApplyAnnualFee();
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM Accounts;
+-----------+---------------+---------+
| AccountID | AccountHolder | Balance |
+-----------+---------------+---------+
|         1 | Alice         |  450.00 |
|         2 | Bob           |  950.00 |
|       101 | 1             | 1150.00 |
|       102 | 2             |  750.00 |
+-----------+---------------+---------+
4 rows in set (0.00 sec)

mysql> _
```

Scenario 3: Update the interest rate for all loans based on a new policy.

- **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE UpdateLoanInterestRates()
    -> BEGIN
    ->     DECLARE done INT DEFAULT FALSE;
    ->     DECLARE lID INT;
    ->     DECLARE cur CURSOR FOR SELECT LoanID FROM Loans;
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    ->
    ->     OPEN cur;
    ->     read_loop: LOOP
    ->         FETCH cur INTO lID;
    ->         IF done THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->         UPDATE Loans SET InterestRate = InterestRate + 1.0 WHERE LoanID = lID;
    ->     END LOOP;
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL UpdateLoanInterestRates();
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Loans;
+--------+------------+--------------+
| LoanID | CustomerID | InterestRate |
+--------+------------+--------------+
|      1 |          1 |         6.00 |
|      2 |          2 |         7.50 |
+--------+------------+--------------+
2 rows in set (0.00 sec)

mysql>
```

Exercise 7: Packages

Scenario 1: Group all customer-related procedures and functions into a package.

- **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

```
mysql> -- Add a new customer
mysql> DELIMITER //
mysql> CREATE PROCEDURE Customer_Add(
    ->     IN p_id INT,
    ->     IN p_name VARCHAR(100),
    ->     IN p_dob DATE
    -> )
    -> BEGIN
    ->     INSERT INTO Customers (CustomerID, Name, DateOfBirth)
    ->     VALUES (p_id, p_name, p_dob);
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Update customer details
mysql> CREATE PROCEDURE Customer_Update(
    ->     IN p_id INT,
    ->     IN p_name VARCHAR(100)
    -> )
    -> BEGIN
    ->     UPDATE Customers
    ->     SET Name = p_name
    ->     WHERE CustomerID = p_id;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Get customer balance
mysql> CREATE FUNCTION Customer_GetBalance(p_id INT) RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE total DECIMAL(10,2);
    ->     SELECT SUM(Amount) INTO total
    ->     FROM Transactions
    ->     WHERE CustomerID = p_id;
    ->     RETURN IFNULL(total, 0);
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DELIMITER ;
mysql> CALL Customer_Add(1, 'Alice', '1990-05-10');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> CALL Customer_Update(1, 'Alicia');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT Customer_GetBalance(1);
+------------------------+
| Customer_GetBalance(1) |
+------------------------+
|                 150.00 |
+------------------------+
1 row in set (0.00 sec)

mysql>
```

Scenario 2: Create a package to manage employee data.

- **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE Employee_Hire(
    ->     IN p_id INT,
    ->     IN p_name VARCHAR(100),
    ->     IN p_dept VARCHAR(50),
    ->     IN p_salary DECIMAL(10,2)
    -> )
    -> BEGIN
    ->     INSERT INTO Employees VALUES (p_id, p_name, p_dept, p_salary);
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Update employee
mysql> CREATE PROCEDURE Employee_Update(
    ->     IN p_id INT,
    ->     IN p_salary DECIMAL(10,2)
    -> )
    -> BEGIN
    ->     UPDATE Employees
    ->     SET Salary = p_salary
    ->     WHERE EmployeeID = p_id;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Get annual salary
mysql> CREATE FUNCTION Employee_GetAnnualSalary(p_id INT) RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE annual DECIMAL(10,2);
    ->     SELECT Salary * 12 INTO annual FROM Employees WHERE EmployeeID = p_id;
    ->     RETURN IFNULL(annual, 0);
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```

```
mysql> CALL Employee_Hire(1, 'John', 'HR', 5000);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT Employee_GetAnnualSalary(1);
+-----------------------------+
| Employee_GetAnnualSalary(1) |
+-----------------------------+
|                    60000.00 |
+-----------------------------+
1 row in set (0.00 sec)

mysql>
```

Scenario 3: Group all account-related operations into a package.

- **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE Account_Open(IN p_customer_id INT, IN p_initial_balance DECIMAL(10,2))
    -> BEGIN
    ->     INSERT INTO Accounts (CustomerID, Balance)
    ->     VALUES (p_customer_id, p_initial_balance);
    -> END;
    -> //
ERROR 1304 (42000): PROCEDURE Account_Open already exists
mysql> CREATE FUNCTION Account_GetTotalBalance(p_customer_id INT) RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE total DECIMAL(10,2);
    ->     SELECT IFNULL(SUM(Balance), 0.00) INTO total
    ->     FROM Accounts
    ->     WHERE CustomerID = p_customer_id;
    ->     RETURN total;
    -> END;
    -> //
```

```
mysql> CALL Account_Open(1, 2000.00);
Query OK, 1 row affected (0.00 sec)

mysql> CALL Account_Open(1, 1500.00);
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> SELECT Account_GetTotalBalance(1);
+----------------------------+
| Account_GetTotalBalance(1) |
+----------------------------+
|                    3500.00 |
+----------------------------+
1 row in set (0.00 sec)

mysql>
```