**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.

**DELIMITER //**

**CREATE PROCEDURE ApplyDiscountToLoans()**

**BEGIN**

**DECLARE done INT DEFAULT 0;**

**DECLARE cust\_id INT;**

**DECLARE age INT;**

**DECLARE cur CURSOR FOR SELECT CustomerID, TIMESTAMPDIFF(YEAR, DOB, CURDATE()) AS Age FROM Customers;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;**

**OPEN cur;**

**read\_loop: LOOP**

**FETCH cur INTO cust\_id, age;**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**IF age > 60 THEN**

**UPDATE Loans**

**SET InterestRate = InterestRate - 1**

**WHERE CustomerID = cust\_id;**

**END IF;**

**END LOOP;**

**CLOSE cur;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE PromoteVIPs()**

**BEGIN**

**DECLARE done INT DEFAULT 0;**

**DECLARE cust\_id INT;**

**DECLARE balance DECIMAL(10, 2);**

**DECLARE cur CURSOR FOR SELECT CustomerID, Balance FROM Customers;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;**

**OPEN cur;**

**read\_loop: LOOP**

**FETCH cur INTO cust\_id, balance;**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**IF balance > 10000 THEN**

**UPDATE Customers**

**SET IsVIP = TRUE**

**WHERE CustomerID = cust\_id;**

**END IF;**

**END LOOP;**

**CLOSE cur;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE SendLoanReminders()**

**BEGIN**

**DECLARE done INT DEFAULT 0;**

**DECLARE cust\_id INT;**

**DECLARE due\_date DATE;**

**DECLARE cur CURSOR FOR SELECT CustomerID, EndDate FROM Loans WHERE EndDate BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY);**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;**

**OPEN cur;**

**read\_loop: LOOP**

**FETCH cur INTO cust\_id, due\_date;**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**SELECT CONCAT('Reminder: Customer ID ', cust\_id, ', Your loan is due on ', due\_date) AS ReminderMessage;**

**END LOOP;**

**CLOSE cur;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE SafeTransferFunds(from\_account INT, to\_account INT, amount DECIMAL(10, 2))**

**BEGIN**

**DECLARE insufficient\_funds CONDITION FOR SQLSTATE '45000';**

**DECLARE EXIT HANDLER FOR insufficient\_funds**

**BEGIN**

**ROLLBACK;**

**INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType)**

**VALUES (NULL, from\_account, NOW(), amount, 'Failed Transfer - Insufficient Funds');**

**END;**

**START TRANSACTION;**

**IF (SELECT Balance FROM Accounts WHERE AccountID = from\_account) < amount THEN**

**SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds';**

**END IF;**

**UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from\_account;**

**UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to\_account;**

**INSERT INTO Transactions (AccountID, TransactionDate, Amount, TransactionType) VALUES**

**(from\_account, NOW(), amount, 'Transfer Out'),**

**(to\_account, NOW(), amount, 'Transfer In');**

**COMMIT;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE UpdateSalary(emp\_id INT, percentage DECIMAL(5, 2))**

**BEGIN**

**DECLARE employee\_not\_found CONDITION FOR SQLSTATE '02000';**

**DECLARE EXIT HANDLER FOR employee\_not\_found**

**BEGIN**

**INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType)**

**VALUES (NULL, NULL, NOW(), NULL, CONCAT('Failed Salary Update - Employee ID ', emp\_id, ' not found'));**

**END;**

**UPDATE Employees**

**SET Salary = Salary \* (1 + percentage / 100)**

**WHERE EmployeeID = emp\_id;**

**IF ROW\_COUNT() = 0 THEN**

**SIGNAL SQLSTATE '02000' SET MESSAGE\_TEXT = 'Employee not found';**

**END IF;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE AddNewCustomer(cust\_id INT, cust\_name VARCHAR(100), dob DATE, balance DECIMAL(10, 2))**

**BEGIN**

**DECLARE duplicate\_customer CONDITION FOR SQLSTATE '23000';**

**DECLARE EXIT HANDLER FOR duplicate\_customer**

**BEGIN**

**INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType)**

**VALUES (NULL, NULL, NOW(), NULL, CONCAT('Failed Customer Insertion - Customer ID ', cust\_id, ' already exists'));**

**END;**

**INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)**

**VALUES (cust\_id, cust\_name, dob, balance, NOW());**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE ProcessMonthlyInterest()**

**BEGIN**

**UPDATE Accounts**

**SET Balance = Balance \* 1.01**

**WHERE AccountType = 'Savings';**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE UpdateEmployeeBonus(dept VARCHAR(50), bonus\_percentage DECIMAL(5, 2))**

**BEGIN**

**UPDATE Employees**

**SET Salary = Salary \* (1 + bonus\_percentage / 100)**

**WHERE Department = dept;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE TransferFunds(from\_account INT, to\_account INT, amount DECIMAL(10, 2))**

**BEGIN**

**DECLARE insufficient\_funds CONDITION FOR SQLSTATE '45000';**

**DECLARE EXIT HANDLER FOR insufficient\_funds**

**BEGIN**

**ROLLBACK;**

**INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType)**

**VALUES (NULL, from\_account, NOW(), amount, 'Failed Transfer - Insufficient Funds');**

**END;**

**START TRANSACTION;**

**IF (SELECT Balance FROM Accounts WHERE AccountID = from\_account) < amount THEN**

**SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds';**

**END IF;**

**UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from\_account;**

**UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to\_account;**

**INSERT INTO Transactions (AccountID, TransactionDate, Amount, TransactionType) VALUES**

**(from\_account, NOW(), amount, 'Transfer Out'),**

**(to\_account, NOW(), amount, 'Transfer In');**

**COMMIT;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE FUNCTION CalculateAge(dob DATE) RETURNS INT**

**BEGIN**

**RETURN TIMESTAMPDIFF(YEAR, dob, CURDATE());**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE FUNCTION CalculateMonthlyInstallment(loan\_amount DECIMAL(10, 2), interest\_rate DECIMAL(5, 2), duration\_years INT) RETURNS DECIMAL(10, 2)**

**BEGIN**

**DECLARE monthly\_rate DECIMAL(5, 2);**

**DECLARE num\_payments INT;**

**DECLARE monthly\_payment DECIMAL(10, 2);**

**SET monthly\_rate = interest\_rate / 1200;**

**SET num\_payments = duration\_years \* 12;**

**SET monthly\_payment = (loan\_amount \* monthly\_rate) / (1 - POWER(1 + monthly\_rate, -num\_payments));**

**RETURN monthly\_payment;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE FUNCTION HasSufficientBalance(account\_id INT, amount DECIMAL(10, 2)) RETURNS BOOLEAN**

**BEGIN**

**DECLARE balance DECIMAL(10, 2);**

**SELECT Balance INTO balance FROM Accounts WHERE AccountID = account\_id;**

**RETURN balance >= amount;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE TRIGGER UpdateCustomerLastModified**

**BEFORE UPDATE ON Customers**

**FOR EACH ROW**

**BEGIN**

**SET NEW.LastModified = NOW();**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE TRIGGER LogTransaction**

**AFTER INSERT ON Transactions**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType)**

**VALUES (NEW.TransactionID, NEW.AccountID, NEW.TransactionDate, NEW.Amount, NEW.TransactionType);**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE TRIGGER CheckTransactionRules**

**BEFORE INSERT ON Transactions**

**FOR EACH ROW**

**BEGIN**

**IF NEW.TransactionType = 'Withdrawal' AND NEW.Amount > (SELECT Balance FROM Accounts WHERE AccountID = NEW.AccountID) THEN**

**SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds for withdrawal';**

**END IF;**

**IF NEW.TransactionType = 'Deposit' AND NEW.Amount <= 0 THEN**

**SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Deposit amount must be positive';**

**END IF;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE GenerateMonthlyStatements()**

**BEGIN**

**DECLARE done INT DEFAULT 0;**

**DECLARE cust\_id INT;**

**DECLARE transaction\_id INT;**

**DECLARE trans\_date DATE;**

**DECLARE amount DECIMAL(10, 2);**

**DECLARE trans\_type VARCHAR(10);**

**DECLARE cur CURSOR FOR SELECT CustomerID FROM Customers;**

**DECLARE trans\_cur CURSOR FOR SELECT TransactionID, TransactionDate, Amount, TransactionType FROM Transactions WHERE TransactionDate BETWEEN DATE\_SUB(CURDATE(), INTERVAL 1 MONTH) AND CURDATE();**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;**

**OPEN cur;**

**customer\_loop: LOOP**

**FETCH cur INTO cust\_id;**

**IF done THEN**

**LEAVE customer\_loop;**

**END IF;**

**OPEN trans\_cur;**

**transaction\_loop: LOOP**

**FETCH trans\_cur INTO transaction\_id, trans\_date, amount, trans\_type;**

**IF done THEN**

**LEAVE transaction\_loop;**

**END IF;**

**SELECT CONCAT('Customer ID ', cust\_id, ', Transaction ID ', transaction\_id, ', Date ', trans\_date, ', Amount ', amount, ', Type ', trans\_type) AS Statement;**

**END LOOP;**

**CLOSE trans\_cur;**

**END LOOP;**

**CLOSE cur;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE ApplyAnnualFee()**

**BEGIN**

**DECLARE done INT DEFAULT 0;**

**DECLARE acc\_id INT;**

**DECLARE cur CURSOR FOR SELECT AccountID FROM Accounts;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;**

**OPEN cur;**

**read\_loop: LOOP**

**FETCH cur INTO acc\_id;**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**UPDATE Accounts**

**SET Balance = Balance - 50**

**WHERE AccountID = acc\_id;**

**INSERT INTO Transactions (AccountID, TransactionDate, Amount, TransactionType)**

**VALUES (acc\_id, NOW(), -50, 'Annual Fee');**

**END LOOP;**

**CLOSE cur;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE UpdateLoanInterestRates(new\_rate DECIMAL(5, 2))**

**BEGIN**

**DECLARE done INT DEFAULT 0;**

**DECLARE loan\_id INT;**

**DECLARE cur CURSOR FOR SELECT LoanID FROM Loans;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;**

**OPEN cur;**

**read\_loop: LOOP**

**FETCH cur INTO loan\_id;**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**UPDATE Loans**

**SET InterestRate = new\_rate**

**WHERE LoanID = loan\_id;**

**END LOOP;**

**CLOSE cur;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE AddNewCustomer(cust\_id INT, cust\_name VARCHAR(100), dob DATE, balance DECIMAL(10, 2))**

**BEGIN**

**DECLARE duplicate\_customer CONDITION FOR SQLSTATE '23000';**

**DECLARE EXIT HANDLER FOR duplicate\_customer**

**BEGIN**

**INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType)**

**VALUES (NULL, NULL, NOW(), NULL, CONCAT('Failed Customer Insertion - Customer ID ', cust\_id, ' already exists'));**

**END;**

**INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)**

**VALUES (cust\_id, cust\_name, dob, balance, NOW());**

**END //**

**CREATE PROCEDURE UpdateCustomerDetails(cust\_id INT, cust\_name VARCHAR(100), dob DATE, balance DECIMAL(10, 2))**

**BEGIN**

**UPDATE Customers**

**SET Name = cust\_name, DOB = dob, Balance = balance, LastModified = NOW()**

**WHERE CustomerID = cust\_id;**

**END //**

**CREATE FUNCTION GetCustomerBalance(cust\_id INT) RETURNS DECIMAL(10, 2)**

**BEGIN**

**DECLARE balance DECIMAL(10, 2);**

**SELECT Balance INTO balance FROM Customers WHERE CustomerID = cust\_id;**

**RETURN balance;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE HireNewEmployee(emp\_id INT, emp\_name VARCHAR(100), position VARCHAR(50), salary DECIMAL(10, 2), dept VARCHAR(50), hire\_date DATE)**

**BEGIN**

**INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)**

**VALUES (emp\_id, emp\_name, position, salary, dept, hire\_date);**

**END //**

**CREATE PROCEDURE UpdateEmployeeDetails(emp\_id INT, emp\_name VARCHAR(100), position VARCHAR(50), salary DECIMAL(10, 2), dept VARCHAR(50))**

**BEGIN**

**UPDATE Employees**

**SET Name = emp\_name, Position = position, Salary = salary, Department = dept**

**WHERE EmployeeID = emp\_id;**

**END //**

**CREATE FUNCTION CalculateAnnualSalary(emp\_id INT) RETURNS DECIMAL(10, 2)**

**BEGIN**

**DECLARE salary DECIMAL(10, 2);**

**SELECT Salary INTO salary FROM Employees WHERE EmployeeID = emp\_id;**

**RETURN salary \* 12;**

**END //**

**DELIMITER ;**

**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.

**DELIMITER //**

**CREATE PROCEDURE OpenNewAccount(acc\_id INT, cust\_id INT, acc\_type VARCHAR(20), balance DECIMAL(10, 2))**

**BEGIN**

**INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)**

**VALUES (acc\_id, cust\_id, acc\_type, balance, NOW());**

**END //**

**CREATE PROCEDURE CloseAccount(acc\_id INT)**

**BEGIN**

**DELETE FROM Accounts WHERE AccountID = acc\_id;**

**END //**

**CREATE FUNCTION GetTotalBalance(cust\_id INT) RETURNS DECIMAL(10, 2)**

**BEGIN**

**DECLARE total\_balance DECIMAL(10, 2);**

**SELECT SUM(Balance) INTO total\_balance FROM Accounts WHERE CustomerID = cust\_id;**

**RETURN total\_balance;**

**END //**

**DELIMITER ;**