**PLSQL ALL EXERCISES (BY SHALINI G SUPERSET ID - 6387392)**

**Exercise 1: Control Structures**

**Scenario 1: Discount for Senior Citizens**

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

**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.**

**Answer:**

DECLARE

v\_age NUMBER;

BEGIN

FOR cust IN (SELECT CustomerID, DOB FROM Customers) LOOP

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - (InterestRate \* 0.01)

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 2: Promote Customers to VIP**

**Question:**  
A customer can be promoted to VIP status based on their balance.

**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.**

**Answer:**

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5));

UPDATE Customers SET IsVIP = 'FALSE';

BEGIN

FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3: Loan Payment Reminders**

**Question:**  
The bank wants to send reminders to customers whose loans are due within the next 30 days.  
**Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.**

**Answer:**

BEGIN

FOR loan IN (SELECT LoanID, CustomerID, EndDate FROM Loans

WHERE EndDate <= SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || loan.LoanID ||

' for Customer ' || loan.CustomerID ||

' is due on ' || loan.EndDate);

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1: Safe Fund Transfer**

**Question:**  
Handle exceptions during fund transfers between accounts.  
**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.**

**Answer:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_fromAccountID IN NUMBER,

p\_toAccountID IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_fromAccountID;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_fromAccountID;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_toAccountID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

**Scenario 2: Handle Error in Salary Update**

**Question:**  
Manage errors when updating employee salaries.  
**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.**

**Answer:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employeeID IN NUMBER,

p\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_percentage / 100)

WHERE EmployeeID = p\_employeeID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID not found');

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

**Scenario 3: Add New Customer Safely**

**Question:**  
Ensure data integrity when adding a new customer.  
**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.**

**Answer:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customerID IN NUMBER,

p\_name IN VARCHAR2,

p\_DOB IN DATE,

p\_balance IN NUMBER

) AS

BEGIN

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

VALUES (p\_customerID, p\_name, p\_DOB, p\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customerID || ' already exists.');

ROLLBACK;

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1: Process Monthly Interest**

**Question:**  
The bank needs to process monthly interest for all savings accounts.  
**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.**

**Answer:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

/

**Scenario 2: Employee Bonus Update**

**Question:**  
The bank wants to implement a bonus scheme for employees based on their performance.  
**Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.**

**Answer:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percentage / 100)

WHERE Department = p\_department;

COMMIT;

END;

/

**Scenario 3: Transfer Funds**

**Question:**  
Customers should be able to transfer funds between their accounts.  
**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.**

**Answer:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_fromAccountID IN NUMBER,

p\_toAccountID IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_fromAccountID;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance');

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_fromAccountID;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_toAccountID;

COMMIT;

END;

/

**Exercise 4: Functions**

**Scenario 1: Calculate Age**

**Question:**  
Calculate the age of customers for eligibility checks.  
**Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.**

**Answer:**

CREATE OR REPLACE FUNCTION CalculateAge(

p\_DOB DATE

) RETURN NUMBER AS

BEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

END;

/

**Scenario 2: Calculate Monthly Installment**

**Question:**  
The bank needs to compute the monthly installment for a loan.  
**Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.**

**Answer:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loanAmount IN NUMBER,

p\_interestRate IN NUMBER,

p\_years IN NUMBER

) RETURN NUMBER AS

v\_monthlyInterest NUMBER;

v\_months NUMBER;

BEGIN

v\_monthlyInterest := p\_interestRate / (12 \* 100);

v\_months := p\_years \* 12;

RETURN (p\_loanAmount \* v\_monthlyInterest) / (1 - POWER(1 + v\_monthlyInterest, -v\_months));

END;

/

**Scenario 3: Check Sufficient Balance**

**Question:**  
Check if a customer has sufficient balance before making a transaction.  
**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.**

**Answer:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_accountID IN NUMBER,

p\_amount IN NUMBER

) RETURN BOOLEAN AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_accountID;

RETURN v\_balance >= p\_amount;

END;

/

**Exercise 5: Triggers**

**Scenario 1: Update Last Modified Date**

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

**Answer:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2: Maintain Audit Log for Transactions**

**Question:**  
Maintain an audit log for all transactions.  
**Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.**

**Answer:**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Action VARCHAR2(50)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, LogDate, Action)

VALUES (:NEW.TransactionID, SYSDATE, 'Transaction Inserted');

END;

/

**Scenario 3: Enforce Business Rules on Transactions**

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

**Answer:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for withdrawal');

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive');

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1: Generate Monthly Statements**

**Question:**  
Generate monthly statements for all customers.  
**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.**

**Answer:**

DECLARE

CURSOR c\_transactions IS

SELECT AccountID, Amount, TransactionDate FROM Transactions

WHERE EXTRACT(MONTH FROM TransactionDate) = EXTRACT(MONTH FROM SYSDATE);

BEGIN

FOR txn IN c\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || txn.AccountID ||

' Transaction Amount: ' || txn.Amount ||

' Date: ' || txn.TransactionDate);

END LOOP;

END;

/

**Scenario 2: Apply Annual Fee**

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

**Answer:**

DECLARE

CURSOR c\_accounts IS SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR acc IN c\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 3: Update Loan Interest Rates**

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

**Answer:**

DECLARE

CURSOR c\_loans IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR loan IN c\_loans LOOP

UPDATE Loans

SET InterestRate = loan.InterestRate + 0.5

WHERE LoanID = loan.LoanID;

END LOOP;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1: Customer Management Package**

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

**Answer:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(...);

PROCEDURE UpdateCustomer(...);

FUNCTION GetCustomerBalance(p\_customerID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

**Scenario 2: Employee Management Package**

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

**Answer:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(...);

PROCEDURE UpdateEmployee(...);

FUNCTION CalculateAnnualSalary(p\_employeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

**Scenario 3: Account Operations Package**

**Question:**  
Group all account-related operations into a package.  
**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.**

**Answer:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(...);

PROCEDURE CloseAccount(...);

FUNCTION GetTotalBalance(p\_customerID NUMBER) RETURN NUMBER;

END AccountOperations;

/