Exercise 1: Control Structures

Scenario 1: Apply a discount to loan interest rates for customers above 60 years olds

```
BEGIN

FOR rec IN (SELECT CustomerID, InterestRate FROM Loans L JOIN Customers C ON L.CustomerID = C.CustomerID WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM C.DOB) > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

END LOOP;

END;
```

Scenario 2: Promote customers to VIP status based on their balance

```
BEGIN

FOR rec IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END LOOP;

END;
```

Scenario 3: Send reminders to customers whose loans are due within the next 30 days

```
BEGIN

FOR rec IN (SELECT C.CustomerID, C.Name, L.EndDate FROM Loans L JOIN Customers C ON L.CustomerID = C.CustomerID WHERE L.EndDate <= SYSDATE + 30) LOOP

DBMS_OUTPUT.PUT_LINE('Reminder: Loan for customer ' || rec.Name || ' is due on ' || TO_CHAR(rec.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;
```

Exercise 2: Error Handling

Scenario 1: SafeTransferFunds stored procedure

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(p_from_account_id IN NUMBER, p_to_account_id IN NUMBER, p_amount IN NUMBER) IS

insufficient_funds EXCEPTION;

v_balance NUMBER;

BEGIN

SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_from_account_id FOR UPDATE;


IF v_balance < p_amount THEN

RAISE insufficient_funds;

END IF;


UPDATE Accounts SET Balance = Balance - p_amount WHERE AccountID = p_from_account_id;

UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to_account_id;


COMMIT;


EXCEPTION

WHEN insufficient_funds THEN

DBMS_OUTPUT.PUT_LINE('Error: Insufficient funds.');

ROLLBACK;

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);

ROLLBACK;

END;
```

Scenario 2: UpdateSalary stored procedure

```
CREATE OR REPLACE PROCEDURE UpdateSalary(p_employee_id IN NUMBER, p_percentage IN NUMBER) IS
```

```
no_employee_found EXCEPTION;
PRAGMA EXCEPTION_INIT(no_employee_found, -01403);
BEGIN
UPDATE Employees SET Salary = Salary * (1 + p_percentage / 100) WHERE EmployeeID = p_employee_id;

IF SQL%NOTFOUND THEN
RAISE no_employee_found;
END IF;

COMMIT;

EXCEPTION
WHEN no_employee_found THEN
DBMS_OUTPUT.PUT_LINE('Error: Employee not found.');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
```

Scenario 3: AddNewCustomer stored procedure

```
CREATE OR REPLACE PROCEDURE AddNewCustomer(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob IN DATE, p_balance IN NUMBER) IS
customer_exists EXCEPTION;
PRAGMA EXCEPTION_INIT(customer_exists, -00001);
BEGIN
INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (p_customer_id, p_name, p_dob, p_balance, SYSDATE);
COMMIT;

EXCEPTION
WHEN customer_exists THEN
DBMS_OUTPUT.PUT_LINE('Error: Customer already exists.');
```

```
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
```

Exercise 3: Stored Procedures

Scenario 1: ProcessMonthlyInterest stored procedure

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP
UPDATE Accounts SET Balance = Balance * 1.01 WHERE AccountID = rec.AccountID;
END LOOP;

COMMIT;
END;
```

Scenario 2: UpdateEmployeeBonus stored procedure

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p_department IN VARCHAR2,
p_bonus_percentage IN NUMBER) IS
BEGIN
UPDATE Employees SET Salary = Salary * (1 + p_bonus_percentage / 100) WHERE Department =
p_department;
COMMIT;
END;
```

Scenario 3: TransferFunds stored procedure

```
CREATE OR REPLACE PROCEDURE TransferFunds(p_from_account_id IN NUMBER, p_to_account_id IN
NUMBER, p_amount IN NUMBER) IS
insufficient_funds EXCEPTION;
v_balance NUMBER;
BEGIN
```

```
SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_from_account_id FOR UPDATE;

IF v_balance < p_amount THEN
RAISE insufficient_funds;
END IF;

UPDATE Accounts SET Balance = Balance - p_amount WHERE AccountID = p_from_account_id;
UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to_account_id;

COMMIT;

EXCEPTION
WHEN insufficient_funds THEN
DBMS_OUTPUT.PUT_LINE('Error: Insufficient funds.');
ROLLBACK;
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
ROLLBACK;
END;
```

Exercise 4: Functions

Scenario 1: CalculateAge function

```
CREATE OR REPLACE FUNCTION CalculateAge(p_dob IN DATE) RETURN NUMBER IS
v_age NUMBER;
BEGIN
v_age := FLOOR((SYSDATE - p_dob) / 365.25);
RETURN v_age;
END;
```

Scenario 2: CalculateMonthlyInstallment function

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p_loan_amount IN NUMBER, p_interest_rate IN NUMBER, p_duration_years IN NUMBER) RETURN NUMBER IS
v_monthly_installment NUMBER;
BEGIN
v_monthly_installment := p_loan_amount * (p_interest_rate / 1200) / (1 - POWER(1 + (p_interest_rate / 1200), -p_duration_years * 12));
RETURN v_monthly_installment;
END;
```

Scenario 3: HasSufficientBalance function

```
CREATE OR REPLACE FUNCTION HasSufficientBalance(p_account_id IN NUMBER, p_amount IN NUMBER) RETURN BOOLEAN IS
v_balance NUMBER;
BEGIN
SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_account_id;

IF v_balance >= p_amount THEN
RETURN TRUE;
ELSE
RETURN FALSE;
END IF;
END;
```

Exercise 5: Triggers

Scenario 1: UpdateCustomerLastModified trigger

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
```

```
:NEW.LastModified := SYSDATE;

END;
```

Scenario 2: LogTransaction trigger

```
CREATE OR REPLACE 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;
```

Scenario 3: CheckTransactionRules trigger

```
CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = :NEW.AccountID FOR UPDATE;

IF v_balance < :NEW.Amount THEN

RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE_APPLICATION_ERROR(-20002, 'Deposit amount must be positive.');
```

END IF;

END IF;

END;

Exercise 6: Cursors

Scenario 1: GenerateMonthlyStatements cursor

BEGIN

FOR rec IN (SELECT C.CustomerID, C.Name, T.TransactionDate, T.Amount, T.TransactionType FROM Transactions T JOIN Accounts A ON T.AccountID = A.AccountID JOIN Customers C ON A.CustomerID = C.CustomerID WHERE T.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST_DAY(SYSDATE)) LOOP

DBMS_OUTPUT.PUT_LINE('Customer ' || rec.Name || ' had a ' || rec.TransactionType || ' of ' || rec.Amount || ' on ' || TO_CHAR(rec.TransactionDate, 'YYYY-MM-DD'));

END LOOP;

END;

Scenario 2: ApplyAnnualFee cursor

BEGIN

FOR rec IN (SELECT AccountID FROM Accounts) LOOP

UPDATE Accounts SET Balance = Balance - 50 WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

```

Scenario 3: UpdateLoanInterestRates cursor

BEGIN

FOR rec IN (SELECT LoanID FROM Loans) LOOP

UPDATE Loans SET InterestRate = InterestRate + 1 WHERE LoanID = rec.LoanID;

END LOOP;
```

```
COMMIT;

END;
```

Exercise 7: Packages

Scenario 1: CustomerManagement package

```
CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob IN DATE, p_balance IN NUMBER);

PROCEDURE UpdateCustomerDetails(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_balance IN NUMBER);

FUNCTION GetCustomerBalance(p_customer_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;


CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob IN DATE, p_balance IN NUMBER) IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (p_customer_id, p_name, p_dob, p_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);

END AddNewCustomer;


PROCEDURE UpdateCustomerDetails(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_balance IN NUMBER) IS

BEGIN
```

```
UPDATE Customers SET Name = p_name, Balance = p_balance, LastModified = SYSDATE WHERE CustomerID = p_customer_id;
COMMIT;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p_customer_id IN NUMBER) RETURN NUMBER IS
v_balance NUMBER;
BEGIN
SELECT Balance INTO v_balance FROM Customers WHERE CustomerID = p_customer_id;
RETURN v_balance;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
RETURN NULL;
END GetCustomerBalance;
```

Scenario 2: EmployeeManagement package

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
PROCEDURE HireEmployee(p_employee_id IN NUMBER, p_name IN VARCHAR2, p_position IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2, p_hiredate IN DATE);
PROCEDURE UpdateEmployeeDetails(p_employee_id IN NUMBER, p_name IN VARCHAR2, p_position IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2);
FUNCTION CalculateAnnualSalary(p_employee_id IN NUMBER) RETURN NUMBER;
END EmployeeManagement;


CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
```

```
PROCEDURE HireEmployee(p_employee_id IN NUMBER, p_name IN VARCHAR2, p_position IN
VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2, p_hiredate IN DATE) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES
(p_employee_id, p_name, p_position, p_salary, p_department, p_hiredate);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);

END HireEmployee;


PROCEDURE UpdateEmployeeDetails(p_employee_id IN NUMBER, p_name IN VARCHAR2, p_position
IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2) IS

BEGIN

UPDATE Employees SET Name = p_name, Position = p_position, Salary = p_salary, Department =
p_department WHERE EmployeeID = p_employee_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);

END UpdateEmployeeDetails;


FUNCTION CalculateAnnualSalary(p_employee_id IN NUMBER) RETURN NUMBER IS

v_salary NUMBER;

BEGIN

SELECT Salary INTO v_salary FROM Employees WHERE EmployeeID = p_employee_id;

RETURN v_salary * 12;

EXCEPTION

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);

RETURN NULL;

END CalculateAnnualSalary;
END EmployeeManagement;
```

Scenario 3: AccountOperations package

```
CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p_account_id IN NUMBER, p_customer_id IN NUMBER, p_account_type
IN VARCHAR2, p_balance IN NUMBER);

PROCEDURE CloseAccount(p_account_id IN NUMBER);

FUNCTION GetTotalBalance(p_customer_id IN NUMBER) RETURN NUMBER;

END AccountOperations;


CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p_account_id IN NUMBER, p_customer_id IN NUMBER, p_account_type
IN VARCHAR2, p_balance IN NUMBER) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES
(p_account_id, p_customer_id, p_account_type, p_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);

END OpenAccount;


PROCEDURE CloseAccount(p_account_id IN NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p_account_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN
```

```
DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END CloseAccount;

FUNCTION GetTotalBalance(p_customer_id IN NUMBER) RETURN NUMBER IS
v_total_balance NUMBER;
BEGIN
SELECT SUM(Balance) INTO v_total_balance FROM Accounts WHERE CustomerID = p_customer_id;
RETURN v_total_balance;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
RETURN NULL;
END GetTotalBalance;
END AccountOperations;
```

These PL/SQL blocks, procedures, functions, and packages cover the scenarios provided, addressing control structures, error handling, stored procedures, functions, triggers, cursors, and packages. They also follow the structure and requirements specified in the schema provided.