**Filename: BankManagementExercises.docx**

**Exercise 1: Control Structures**

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

DECLARE

CURSOR cust\_cursor IS

SELECT l.LoanID, c.DOB

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID;

BEGIN

FOR rec IN cust\_cursor LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

COMMIT;

END;

/

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

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID, Balance FROM Customers;

BEGIN

FOR rec IN vip\_cursor LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET Balance = rec.Balance, LastModified = SYSDATE

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

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

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate <= SYSDATE + 30;

BEGIN

FOR rec IN loan\_cursor LOOP

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

' for Customer ' || rec.CustomerID ||

' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1: Handle exceptions during fund transfers between accounts.**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

ex\_insufficient\_funds EXCEPTION;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts

WHERE AccountID = p\_from\_account FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE ex\_insufficient\_funds;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

EXCEPTION

WHEN ex\_insufficient\_funds THEN

ROLLBACK;

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

VALUES (SEQ\_TRANS\_ID.NEXTVAL, p\_from\_account, SYSDATE, 0, 'Failed');

WHEN OTHERS THEN

ROLLBACK;

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

VALUES (SEQ\_TRANS\_ID.NEXTVAL, p\_from\_account, SYSDATE, 0, 'Error');

END;

/

Scenario 2: Manage errors when updating employee salaries.

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID not found.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

VALUES (SEQ\_TRANS\_ID.NEXTVAL, NULL, SYSDATE, 0, 'Salary Update Error');

END;

/

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

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_cust\_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\_cust\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

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

VALUES (SEQ\_TRANS\_ID.NEXTVAL, NULL, SYSDATE, 0, 'Duplicate Customer');

WHEN OTHERS THEN

ROLLBACK;

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1: Process monthly interest for all savings accounts.**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

/

Scenario 2: Implement bonus scheme for employees.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept IN VARCHAR2,

p\_bonus\_pct IN NUMBER

)

IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_dept;

COMMIT;

END;

/

**Scenario 3: Transfer funds between customer accounts.**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts

WHERE AccountID = p\_from\_account FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient Balance');

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

END;

/

**Exercise 4: Functions**

**Scenario 1: Calculate age of customers.**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER

IS

BEGIN

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

END;

/

Scenario 2: Compute monthly installment.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amt IN NUMBER,

p\_rate IN NUMBER,

p\_years IN NUMBER

) RETURN NUMBER

IS

v\_r NUMBER := p\_rate / 12 / 100;

v\_n NUMBER := p\_years \* 12;

BEGIN

RETURN (p\_loan\_amt \* v\_r) / (1 - POWER(1 + v\_r, -v\_n));

END;

/

**Scenario 3: Check if sufficient balance.**

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;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

**Exercise 5: Triggers**

**Scenario 1: Update LastModified on customer update.**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

Scenario 2: Audit log for transactions.

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, ActionDate, Description)

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

END;

/

**Scenario 3: Enforce deposit/withdrawal rules.**

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(-20003, 'Withdrawal exceeds balance');

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

RAISE\_APPLICATION\_ERROR(-20004, 'Invalid deposit amount');

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1: Generate monthly statements.**

DECLARE

CURSOR trans\_cursor IS

SELECT t.AccountID, t.Amount, t.TransactionDate

FROM Transactions t

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

BEGIN

FOR rec IN trans\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Account: ' || rec.AccountID || ' Amount: ' || rec.Amount ||

' Date: ' || TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Scenario 2: Apply annual fee.**

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR rec IN acc\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 3: Update loan interest rates.**

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR rec IN loan\_cursor LOOP

UPDATE Loans

SET InterestRate = rec.InterestRate + 0.5

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1: CustomerManagement package.**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES(p\_id, p\_name, p\_dob, p\_balance, SYSDATE);

END;

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Customers SET Name = p\_name, LastModified = SYSDATE WHERE CustomerID = p\_id;

END;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_id;

RETURN v\_balance;

END;

END;

/

Scenario 2: EmployeeManagement package.

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2) IS

BEGIN

INSERT INTO Employees VALUES(p\_id, p\_name, p\_position, p\_salary, p\_dept, SYSDATE);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Employees SET Name = p\_name WHERE EmployeeID = p\_id;

END;

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;

RETURN v\_salary \* 12;

END;

END;

/

**Scenario 3: AccountOperations package.**

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES(p\_acc\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_acc\_id;

END;

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;

RETURN v\_total;

END;

END;

/