**PL/SQL**

**EXCERRCISE 1**

**SCENARIO 1**

BEGIN

FOR rec IN (SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l USING (CustomerID)) LOOP

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID ||

' age >60, loan '||rec.LoanID||

' new rate: '||

TO\_CHAR(rec.InterestRate -1));

END IF;

END LOOP;

COMMIT;

END;

**OUTPUT:**

Customer 3 age >60, loan 5 new rate: 4

**SCENARIO 2**

BEGIN

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

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer '||rec.CustomerID||' marked VIP');

END IF;

END LOOP;

COMMIT;

END;

**OUTPUT**

Customer 4 marked VIP

**SCENARIO 3**

BEGIN

FOR rec IN (SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c USING (CustomerID)

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE+30) LOOP

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

' for '||rec.Name||

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

END LOOP;

END;

**OUTPUT**

Reminder: Loan 1 for John Doe is due on 2025-07-10

**Exercise 2: Error Handling**

**SCENARIO 1**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_acct NUMBER,

p\_to\_acct NUMBER,

p\_amount NUMBER

) IS

l\_balance NUMBER;

BEGIN

SELECT Balance INTO l\_balance FROM Accounts WHERE AccountID = p\_from\_acct FOR UPDATE;

IF l\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in account '||p\_from\_acct);

END IF;

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred '||p\_amount||' from '||p\_from\_acct||' to '||p\_to\_acct);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

**OUTPUT**

Transfer failed: ORA-20001: Insufficient funds in account 1

**SCENARIO 2**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_empid NUMBER,

p\_pct\_increase NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_pct\_increase/100)

WHERE EmployeeID = p\_empid;

IF SQL%ROWCOUNT = 0 THEN

RAISE NO\_DATA\_FOUND;

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for employee '||p\_empid);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee '||p\_empid||' not found');

WHEN OTHERS THEN

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

END UpdateSalary;

**OUTPUT**

Error: Employee 99 not found

SCENARIO 3

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) IS

BEGIN

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added: '||p\_id);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END AddNewCustomer;

**OUTPUT**

Error: Customer ID 1 already exists

**Exercise 3: Stored Procedures**

**Scenario 1**

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;

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

' new balance: '||

TO\_CHAR(rec.Balance \* 1.01,'FM9999990.00'));

END LOOP;

COMMIT;

END ProcessMonthlyInterest;

**OUTPUT**

1 employees in IT updated with 5% bonus

**SCENARIO 2**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_acct NUMBER,

p\_to\_acct NUMBER,

p\_amount NUMBER

) IS

l\_bal NUMBER;

BEGIN

SELECT Balance INTO l\_bal FROM Accounts WHERE AccountID = p\_from\_acct FOR UPDATE;

IF l\_bal < p\_amount THEN

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

END IF;

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred '||p\_amount);

END TransferFunds;

**OUTPUT**

Transferred 100

**Exercise 4: Functions**

**Scenario 1**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS

BEGIN

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

END CalculateAge;

**OUTPUT**

40

**Scenario 2**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amt NUMBER,

p\_rate\_pct NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

v\_r NUMBER := p\_rate\_pct/1200;

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

BEGIN

RETURN ROUND((p\_loan\_amt \* v\_r)/(1 - POWER(1+v\_r, -v\_n)),2);

END CalculateMonthlyInstallment;

**OUTPUT**

1060.66

**Scenario 3**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_acct\_id NUMBER,

p\_amt NUMBER

) RETURN BOOLEAN IS

l\_bal NUMBER;

BEGIN

SELECT Balance INTO l\_bal FROM Accounts WHERE AccountID = p\_acct\_id;

RETURN l\_bal >= p\_amt;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END HasSufficientBalance;

**Exercise 5**

SCENARIO 1

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**OUTPUT**

LASTMODIFIED

2025-06-29 17:22:00

**SCENARIO 2**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AuditID, TableName, RecordID, Action, ActionDate)

VALUES (

AuditLog\_SEQ.NEXTVAL, 'Transactions', :NEW.TransactionID,

'INSERT', SYSDATE

);

END;

**OUTPUT**

**AUDITID | TABLENAME | RECORDID | ACTION | ACTIONDATE**

**1 | Transactions | 3 | INSERT | 2025-06-29 17:25:00**

**Scenario 3**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

l\_bal NUMBER;

BEGIN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20010,'Amount must be positive');

END IF;

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO l\_bal FROM Accounts WHERE AccountID = :NEW.AccountID;

IF l\_bal < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20011,'Withdrawal exceeds balance');

END IF;

UPDATE Accounts SET Balance = Balance - :NEW.Amount WHERE AccountID = :NEW.AccountID;

ELSIF :NEW.TransactionType = 'Deposit' THE

UPDATE Accounts SET Balance = Balance + :NEW.Amount WHERE AccountID = :NEW.AccountID;

END IF;

END;

**OUTPUT**

ORA-20011: Withdrawal exceeds balance

**Exercise 6: Cursors**

**Scenario 1**

DECLARE

CURSOR c\_txn IS

SELECT t.AccountID, t.TransactionDate, t.Amount, t.TransactionType, a.CustomerID

FROM Transactions t JOIN Accounts a USING (AccountID)

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

BEGIN

FOR rec IN c\_txn LOOP

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

' Customer '||rec.CustomerID||

' '||rec.TransactionType||

' '||rec.Amount||

' on '||TO\_CHAR(rec.TransactionDate,'YYYY-MM-DD'));

END LOOP;

END;

**OUTPUT**

Account 1 Customer 1 Deposit 200 on 2025-06-28

Account 2 Customer 2 Withdrawal 300 on 2025-06-28

**SCENARIO 2**

DECLARE

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

BEGIN

FOR rec IN c\_acc LOOP

UPDATE Accounts

SET Balance = Balance - 100 -- flat annual fee

WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Deducted fee from '||rec.AccountID);

END LOOP;

COMMIT;

END;

**OUTPUT**

Deducted fee from 1

Deducted fee from 2

**Scenario 3**

DECLARE

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

BEGIN

FOR rec IN c\_loan LOOP

UPDATE Loans

SET InterestRate = InterestRate + 0.5 -- new policy

WHERE LoanID = rec.LoanID;

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

' new rate: '||

TO\_CHAR(rec.InterestRate + 0.5));

END LOOP;

COMMIT;

END;

**OUTPUT**

Loan 1 new rate: 5.5

**Exercise 7: Packages**

**Scenario 1**

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

PROCEDURE UpdateDetails(p\_id NUMBER, p\_name VARCHAR2, p\_balance NUMBER);

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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);

EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN

NULL;

END;

PROCEDURE UpdateDetails(p\_id NUMBER, p\_name VARCHAR2, p\_balance NUMBER) IS

BEGIN

UPDATE Customers SET Name=p\_name, Balance=p\_balance, LastModified=SYSDATE

WHERE CustomerID=p\_id;

END;

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS

l\_bal NUMBER;

BEGIN

SELECT Balance INTO l\_bal FROM Customers WHERE CustomerID=p\_id;

RETURN l\_bal;

EXCEPTION WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END CustomerManagement;

**OUTPUT**

Customer added: 3

**Scenario 2**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

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

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

BEGIN

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

EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN

NULL;

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_id;

END;

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

l\_sal NUMBER;

BEGIN

SELECT Salary INTO l\_sal FROM Employees WHERE EmployeeID = p\_id;

RETURN l\_sal \* 12;

EXCEPTION WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END EmployeeManagement;

SCENARIO 3

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_initbal NUMBER);

PROCEDURE CloseAccount(p\_accid NUMBER);

FUNCTION TotalBalance(p\_custid NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_initbal NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES(p\_accid, p\_custid, p\_type, p\_initbal, SYSDATE);

EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN

NULL;

END;

PROCEDURE CloseAccount(p\_accid NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID=p\_accid;

END;

FUNCTION TotalBalance(p\_custid NUMBER) RETURN NUMBER IS

l\_tot NUMBER;

BEGIN

SELECT SUM(Balance) INTO l\_tot FROM Accounts WHERE CustomerID=p\_custid;

RETURN NVL(l\_tot, 0);

END;

END AccountOperations;

**OUTPUT**

account opened