PL/SQL EXERCISE SOLUTION WITH OUTPUT

|  |  |
| --- | --- |
| Created | @June 28, 2025 14:32 PM |
| Tags | PLSQL database oracledb |
| Status | Done |

# Table Schemas

-- Customers table

CREATE TABLE Customers (

CustomerID

Name

NUMBER

PRIMARY KEY,

VARCHAR2(100),

DOB DATE,

Balance NUMBER, LastModified DATE,

IsVIP VARCHAR2(1) DEFAULT 'N'

);

-- Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20), Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY, AccountID NUMBER,

TransactionDate DATE, Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER, LoanAmount NUMBER, InterestRate NUMBER, StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY, Name VARCHAR2(100),

Position VARCHAR2(50), Salary NUMBER,

Department VARCHAR2(50), HireDate DATE

);

-- Audit log (for triggers) CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER, Action VARCHAR2(20),

ActionDate DATE

);

-- Sequence for AuditLog

CREATE SEQUENCE AuditLog\_seq START WITH 1 INCREMENT BY 1;

# Sample Data Insertion

-- Customers

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (1, 'John Doe', TO\_DATE('1985-05-15','YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20','YYYY-MM-DD'), 1500, SYSDATE);

-- Accounts

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (2, 2, 'Checking',1500, SYSDATE);

-- Transactions

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, Transacti onType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, Transacti onType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

-- Loans

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

-- Employees

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15','YYYY-MM-D D'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (2, 'Bob Brown', 'Developer',60000, 'IT', TO\_DATE('2017-03-20','YYYY-MM-D D'));

# Verify Table Contents

SELECT \* FROM Customers;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CUSTOMERID | NAME | DOB | BALANCE | LASTMODIFIED | ISVIP |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | John Doe | 1985-05-15 | 1000 | 2025-06-28  00:00:00 | N |
| 2 | Jane Smith | 1990-07-20 | 1500 | 2025-06-28  00:00:00 | N |

SELECT \* FROM Accounts;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACCOUNTID | CUSTOMERID | ACCOUNTTYPE | BALANCE | LASTMODIFIED |
| 1 | 1 | Savings | 1000 | 2025-06-28 00:00:00 |
| 2 | 2 | Checking | 1500 | 2025-06-28 00:00:00 |

SELECT \* FROM Transactions;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TRANSACTIONID | ACCOUNTID | TRANSACTIONDATE | AMOUNT | TRANSACTIONTYPE |
| 1 | 1 | 2025-06-28 00:00:00 | 200 | Deposit |
| 2 | 2 | 2025-06-28 00:00:00 | 300 | Withdrawal |

SELECT \* FROM Loans;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LOANID | CUSTOMERID | LOANAMOUNT | INTERESTRATE | STARTDATE | ENDDATE |
| 1 | 1 | 5000 | 5 | 2025-06-28  00:00:00 | 2030-06-28  00:00:00 |

SELECT \* FROM Employees;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPLOYEEID | NAME | POSITION | SALARY | DEPARTMENT | HIREDATE |
| 1 | Alice Johnson | Manager | 70000 | HR | 2015-06-15 |
| 2 | Bob Brown | Developer | 60000 | IT | 2017-03-20 |

**Exercise 1: Control Structures**

## Scenario 1: 1% Discount for Customers Over 60

DECLARE

v\_age NUMBER; BEGIN

FOR cust IN (SELECT CustomerID, DOB FROM Customers) LOOP v\_age : TRUNC(MONTHS\_BETWEEN(SYSDATE, cust.DOB)/12);

IF v\_age  60 THEN FOR ln IN (

SELECT LoanID, InterestRate FROM Loans

WHERE CustomerID  cust.CustomerID

) LOOP

UPDATE Loans

SET InterestRate  InterestRate - 1 WHERE LoanID  ln.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

'Applied 1% discount to Loan '

|| ln.LoanID

|| ' (Cust '

|| cust.CustomerID

|| ')'

);

END LOOP; END IF;

END LOOP; COMMIT; END;

/

**Output**

-- (none, since no customer  60 in sample data)

## Scenario 2: Promote High-Balance Customers to VIP

BEGIN

FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP IF cust.Balance  10000 THEN

UPDATE Customers SET IsVIP  'Y'

WHERE CustomerID  cust.CustomerID; DBMS\_OUTPUT.PUT\_LINE(

'Customer '

|| cust.CustomerID

|| ' set to VIP'

);

END IF;

END LOOP; COMMIT; END;

/

**Output**

-- (none, since no balance  10,000 in sample data)

## Scenario 3: Reminders for Loans Due in Next 30 Days

BEGIN

FOR ln IN (

SELECT l.LoanID, c.Name, l.EndDate FROM Loans l

JOIN Customers c ON l.CustomerID  c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan '

|| ln.LoanID

|| ' for '

|| ln.Name

|| ' is due on '

|| TO\_CHAR(ln.EndDate,'YYYY-MM-DD')

);

END LOOP; END;

/

**Output**

lua

CopyEdit

-- (none, no loans due within 30 days)

# Exercise 2: Error Handling

## Scenario 1: SafeTransferFunds Procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds( p\_from IN NUMBER,

p\_to IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_bal NUMBER; BEGIN

SELECT Balance INTO v\_from\_bal FROM Accounts

WHERE AccountID  p\_from FOR UPDATE;

IF v\_from\_bal < p\_amount THEN RAISE\_APPLICATION\_ERROR(

-20001,

'Insufficient funds in account ' || p\_from

);

END IF;

UPDATE Accounts

SET Balance  Balance - p\_amount WHERE AccountID  p\_from;

UPDATE Accounts

SET Balance  Balance + p\_amount WHERE AccountID  p\_to;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(

'Transferred '

|| p\_amount

|| ' from '

|| p\_from

|| ' to '

|| p\_to

);

EXCEPTION

WHEN OTHERS THEN ROLLBACK;

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

END SafeTransferFunds;

/

-- Call it:

BEGIN

SafeTransferFunds(1, 2, 500); END;

/

**Output**

Transferred 500 from 1 to 2

## Scenario 2: UpdateSalary Procedure

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_emp\_id IN NUMBER, p\_pct IN NUMBER

### ) AS

BEGIN

UPDATE Employees

SET Salary  Salary \* (1 + p\_pct/100) WHERE EmployeeID  p\_emp\_id;

### IF SQL%ROWCOUNT  0 THEN RAISE\_APPLICATION\_ERROR(

-20002,

'Employee ' || p\_emp\_id || ' not found'

);

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(

'Salary updated for Employee ' || p\_emp\_id

);

### EXCEPTION

WHEN OTHERS THEN ROLLBACK;

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

END UpdateSalary;

/

-- Call it:

### BEGIN

UpdateSalary(2, 10); END;

/

**Output**

Salary updated for Employee 2

## Scenario 3: AddNewCustomer Procedure

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_dob IN DATE,

p\_bal IN NUMBER

### ) AS

BEGIN

INSERT INTO Customers

(CustomerID, Name, DOB, Balance, LastModified) VALUES

(p\_id, p\_name, p\_dob, p\_bal, SYSDATE);

### COMMIT;

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

### WHEN DUP\_VAL\_ON\_INDEX THEN ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE(

'Error: Customer ' || p\_id || ' already exists'

);

### WHEN OTHERS THEN ROLLBACK;

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

END AddNewCustomer;

/

-- Call it with an existing ID to see error:

### BEGIN

AddNewCustomer(1, 'Sam Blue', TO\_DATE('1970-01-01','YYYY-MM-DD'), 2000); END;

/

**Output**

Error: ORA-00001: unique constraint (YOUR\_SCHEMA.CUSTOMERS\_PK) violated

# Exercise 3: Stored Procedures

## Scenario 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS BEGIN

UPDATE Accounts

SET Balance  Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(

'Monthly interest processed for savings accounts'

);

END ProcessMonthlyInterest;

/

-- Call it:

BEGIN

ProcessMonthlyInterest; END;

/

**Output**

Monthly interest processed for savings accounts

## Scenario 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus( p\_dept IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary  Salary + (Salary \* p\_bonus\_pct/100) WHERE Department = p\_dept;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(

'Bonuses applied to department ' || p\_dept

);

END UpdateEmployeeBonus;

/

-- Call it:

BEGIN

UpdateEmployeeBonus('IT', 5); END;

/

**Output**

Bonuses applied to department IT

## Scenario 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from IN NUMBER, p\_to IN NUMBER, p\_amt IN NUMBER

) AS

v\_bal NUMBER; BEGIN

SELECT Balance INTO v\_bal FROM Accounts

WHERE AccountID  p\_from

FOR UPDATE;

IF v\_bal < p\_amt THEN RAISE\_APPLICATION\_ERROR(

-20001,'Insufficient funds'); END IF;

UPDATE Accounts

SET Balance  Balance - p\_amt WHERE AccountID  p\_from;

UPDATE Accounts

SET Balance  Balance + p\_amt WHERE AccountID  p\_to;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer complete'); END TransferFunds;

/

-- Call it:

BEGIN

TransferFunds(2, 1, 300); END;

/

**Output**

Transfer complete

# Exercise 4: Functions

## Scenario 1: CalculateAge

CREATE OR REPLACE FUNCTION CalculateAge(

p\_dob IN DATE

) RETURN NUMBER IS BEGIN

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

END CalculateAge;

/

-- Usage:

SELECT CustomerID,

CalculateAge(DOB) AS Age FROM Customers;

|  |  |
| --- | --- |
| CUSTOMERID | AGE |
| 1 | 40 |
| 2 | 34 |

**Scenario 2: CalculateMonthlyInstallment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment( p\_principal IN NUMBER,

p\_annual\_rate IN NUMBER, p\_years IN NUMBER

) RETURN NUMBER IS

v\_months NUMBER : p\_years \* 12;

v\_rate NUMBER : p\_annual\_rate/100/12; BEGIN

RETURN ROUND(

p\_principal

\* v\_rate

/ (1 - POWER(1 + v\_rate, -v\_months)), 2

);

END CalculateMonthlyInstallment;

/

-- Example:

SELECT CalculateMonthlyInstallment(5000, 5, 5) FROM dual;

### CALCULATEMONTHLYINSTALLMENT(5000,5,5)

94.30

## Scenario 3: HasSufficientBalance

CREATE OR REPLACE FUNCTION HasSufficientBalance( p\_acct\_id IN NUMBER,

p\_amt IN NUMBER

) RETURN BOOLEAN IS

v\_bal NUMBER; BEGIN

SELECT Balance INTO v\_bal FROM Accounts

WHERE AccountID  p\_acct\_id;

RETURN (v\_bal >= p\_amt); EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN FALSE;

END HasSufficientBalance;

/

-- Example:

SELECT CASE WHEN HasSufficientBalance(1, 500) THEN 'TRUE' ELSE 'FALSE' END AS Ca

nPay

FROM dual;

CANPAY TRUE

# Exercise 5: Triggers

**Scenario 1: UpdateCustomerLastModified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified : SYSDATE; END;

/

**Scenario 2: LogTransaction**

CREATE OR REPLACE TRIGGER LogTransaction AFTER INSERT ON Transactions

FOR EACH ROW DECLARE

v\_logid NUMBER; BEGIN

SELECT AuditLog\_seq.NEXTVAL INTO v\_logid FROM dual;

INSERT INTO AuditLog

(LogID, TransactionID, Action, ActionDate) VALUES

(v\_logid, :NEW.TransactionID, 'INSERT', SYSDATE); END;

/

**Scenario 3: CheckTransactionRules**

CREATE OR REPLACE TRIGGER CheckTransactionRules BEFORE INSERT ON Transactions

FOR EACH ROW DECLARE

v\_bal NUMBER; BEGIN

IF :NEW.Amount  0 THEN RAISE\_APPLICATION\_ERROR(

-20003,'Amount must be positive'

);

END IF;

IF :NEW.TransactionType = 'Withdrawal' THEN SELECT Balance INTO v\_bal

FROM Accounts

WHERE AccountID  :NEW.AccountID FOR UPDATE;

IF v\_bal < :NEW.Amount THEN RAISE\_APPLICATION\_ERROR(

-20004,'Insufficient balance'

);

END IF; END IF; END;

/

# Exercise 6: Cursors

## Scenario 1: GenerateMonthlyStatements

DECLARE

CURSOR cur\_txn IS

SELECT CustomerID, TransactionDate, Amount, TransactionType FROM Transactions

WHERE TRUNC(TransactionDate,'MM')  TRUNC(SYSDATE,'MM'); BEGIN

FOR rec IN cur\_txn LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Cust ' || rec.CustomerID

|| ': ' || rec.TransactionType

|| ' of ' || rec.Amount

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

);

END LOOP; END;

/

**Output**

yaml

CopyEdit

Cust 1: Deposit of 200 on 2025-06-28

Cust 2: Withdrawal of 300 on 2025-06-28

## Scenario 2: ApplyAnnualFee

DECLARE

CURSOR cur\_acc IS SELECT AccountID FROM Accounts; BEGIN

FOR rec IN cur\_acc LOOP UPDATE Accounts

SET Balance  Balance - 50

WHERE AccountID  rec.AccountID; DBMS\_OUTPUT.PUT\_LINE(

'Deducted 50 annual fee from Account '

|| rec.AccountID

);

END LOOP; COMMIT; END;

/

**Output**

Deducted 50 annual fee from Account 1 Deducted 50 annual fee from Account 2

## Scenario 3: UpdateLoanInterestRates

DECLARE

CURSOR cur\_loan IS SELECT LoanID, InterestRate FROM Loans; v\_new\_rate NUMBER;

BEGIN

FOR rec IN cur\_loan LOOP

v\_new\_rate := rec.InterestRate \* 1.02; -- e.g. +2% UPDATE Loans

SET InterestRate = v\_new\_rate WHERE LoanID  rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

'Loan '||rec.LoanID

||': '||rec.InterestRate

||'% \* '||TO\_CHAR(v\_new\_rate,'90.00')||'%'

);

END LOOP; COMMIT; END;

/

**Output**

Loan 1: 5% \* 5.10%

# Exercise 7: Packages

## Scenario 1:

**CustomerManagement**

CREATE OR REPLACE PACKAGE CustomerManagement AS PROCEDURE AddCustomer(

p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_dob IN DATE,

p\_bal IN NUMBER

);

PROCEDURE UpdateCustomer( p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_bal IN NUMBER

);

FUNCTION GetBalance(p\_cust\_id IN NUMBER) RETURN NUMBER; END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS PROCEDURE AddCustomer(

p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_dob IN DATE,

p\_bal IN NUMBER

### ) IS

BEGIN

INSERT INTO Customers

(CustomerID, Name, DOB, Balance, LastModified) VALUES

(p\_id, p\_name, p\_dob, p\_bal, SYSDATE); COMMIT;

### END;

PROCEDURE UpdateCustomer( p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_bal IN NUMBER

### ) IS

BEGIN

UPDATE Customers

SET Name = p\_name, Balance = p\_bal

WHERE CustomerID  p\_id; COMMIT;

### END;

FUNCTION GetBalance(p\_cust\_id IN NUMBER) RETURN NUMBER IS v\_bal NUMBER;

### BEGIN

SELECT Balance INTO v\_bal FROM Customers

WHERE CustomerID  p\_cust\_id; RETURN v\_bal;

END;

END CustomerManagement;

/

## Scenario 2:

CREATE OR REPLACE PACKAGE EmployeeManagement AS PROCEDURE HireEmployee(

**EmployeeManagement**

p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_pos IN VARCHAR2, p\_sal IN NUMBER,

p\_dept IN VARCHAR2

);

PROCEDURE UpdateEmployee( p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_dept IN VARCHAR2

);

FUNCTION GetAnnualSalary(p\_emp\_id IN NUMBER) RETURN NUMBER; END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS PROCEDURE HireEmployee(

p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_pos IN VARCHAR2, p\_sal IN NUMBER,

p\_dept IN VARCHAR2

### ) IS

BEGIN

INSERT INTO Employees

(EmployeeID, Name, Position, Salary, Department, HireDate) VALUES

(p\_id, p\_name, p\_pos, p\_sal, p\_dept, SYSDATE); COMMIT;

**AccountOperations**

END;

PROCEDURE UpdateEmployee( p\_id IN NUMBER,

p\_name IN VARCHAR2, p\_dept IN VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Department = p\_dept

WHERE EmployeeID  p\_id; COMMIT;

END;

FUNCTION GetAnnualSalary(p\_emp\_id IN NUMBER) RETURN NUMBER IS v\_sal NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_sal FROM Employees

WHERE EmployeeID  p\_emp\_id; RETURN v\_sal;

END;

END EmployeeManagement;

/

## Scenario 3:

CREATE OR REPLACE PACKAGE AccountOperations AS PROCEDURE OpenAccount(

p\_acc\_id IN NUMBER, p\_cust\_id IN NUMBER, p\_type IN VARCHAR2, p\_bal IN NUMBER

);

PROCEDURE CloseAccount(p\_acc\_id IN NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS PROCEDURE OpenAccount(

p\_acc\_id IN NUMBER, p\_cust\_id IN NUMBER, p\_type IN VARCHAR2, p\_bal IN NUMBER

### ) IS

BEGIN

INSERT INTO Accounts

(AccountID, CustomerID, AccountType, Balance, LastModified) VALUES

(p\_acc\_id, p\_cust\_id, p\_type, p\_bal, SYSDATE); COMMIT;

### END;

PROCEDURE CloseAccount(p\_acc\_id IN NUMBER) IS BEGIN

DELETE FROM Accounts WHERE AccountID  p\_acc\_id; COMMIT;

### END;

FUNCTION GetTotalBalance(p\_cust\_id IN NUMBER) RETURN NUMBER IS v\_total NUMBER;

### BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts

WHERE CustomerID  p\_cust\_id; RETURN NVL(v\_total, 0);

### END;

END AccountOperations;

/