**#Schema that have to be created**

CREATE TABLE CUSTOMERS (

CUSTOMERID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

DOB DATE,

BALANCE NUMBER,

LASTMODIFIED DATE

);

CREATE TABLE ACCOUNTS (

ACCOUNTID NUMBER PRIMARY KEY,

CUSTOMERID NUMBER,

ACCOUNTTYPE VARCHAR2(20),

BALANCE NUMBER,

LASTMODIFIED DATE,

FOREIGN KEY ( CUSTOMERID )

REFERENCES CUSTOMERS ( CUSTOMERID )

);

CREATE TABLE TRANSACTIONS (

TRANSACTIONID NUMBER PRIMARY KEY,

ACCOUNTID NUMBER,

TRANSACTIONDATE DATE,

AMOUNT NUMBER,

TRANSACTIONTYPE VARCHAR2(10),

FOREIGN KEY ( ACCOUNTID )

REFERENCES ACCOUNTS ( ACCOUNTID )

);

CREATE TABLE LOANS (

LOANID NUMBER PRIMARY KEY,

CUSTOMERID NUMBER,

LOANAMOUNT NUMBER,

INTERESTRATE NUMBER,

STARTDATE DATE,

ENDDATE DATE,

FOREIGN KEY ( CUSTOMERID )

REFERENCES CUSTOMERS ( CUSTOMERID )

);

CREATE TABLE EMPLOYEES (

EMPLOYEEID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

POSITION VARCHAR2(50),

SALARY NUMBER,

DEPARTMENT VARCHAR2(50),

HIREDATE DATE

);

-- INSERT INTO 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);

-- INSERT INTO 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);

-- INSTER INTO TRANSACTIONS

INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

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

INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

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

-- INSERT INTO LOANS

INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

-- INSERT INTO EMPLOYEES

INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- QUESTIONS AND SOLUTIONS

/\*

**Exercise 1: Control Structures**

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

**Question: 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.**

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

**Question: 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.**

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

**Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder**

**message for each customer.**

**\*/**

-- SCENARIO 1

SELECT \* FROM CUSTOMERS;

SELECT \* FROM LOANS;

SET SERVEROUTPUT ON;

DECLARE

CURSOR CUSTOMER\_CURSOR IS

SELECT CUSTOMERID, EXTRACT (YEAR FROM SYSDATE) – EXTRACT (YEAR FROM DOB) AS AGE

FROM CUSTOMERS;

VAR\_CUSTOMER\_ID CUSTOMERS.CUSTOMERID%TYPE;

VAR\_AGE NUMBER;

BEGIN

FOR CUSTOMER\_RECORD IN CUSTOMER\_CURSOR LOOP

VAR\_CUSTOMER\_ID: = CUSTOMER\_RECORD.CUSTOMERID;

VAR\_AGE: = CUSTOMER\_RECORD.AGE;

IF VAR\_AGE > 60 THEN

UPDATE LOANS

SET INTERESTRATE = INTERESTRATE - 1

WHERE CUSTOMERID = VAR\_CUSTOMER\_ID;

ELSE

DBMS\_OUTPUT.PUT\_LINE ('CUSTOMER WITH CUSTOMER ID: ' || VAR\_CUSTOMER\_ID || ' IS OF AGE: ' || VAR\_AGE);

DBMS\_OUTPUT.PUT\_LINE ('NO CHANGE IN LOAN');

END IF;

END LOOP;

COMMIT;

END;

/

SELECT \* FROM LOANS;

-- SCENARIO 2

DESC CUSTOMERS;

ALTER TABLE CUSTOMERS ADD ISVIP CHAR(10) CONSTRAINT CHK1 CHECK(ISVIP IN ('TRUE','FALSE')) ;

SELECT \* FROM CUSTOMERS;

SET SERVEROUTPUT ON;

DECLARE

CURSOR CUSTOMER\_CURSOR IS

SELECT CUSTOMERID, BALANCE

FROM CUSTOMERS;

VAR\_CUSTOMER\_ID CUSTOMERS.CUSTOMERID%TYPE;

VAR\_BALANCE CUSTOMERS.BALANCE%TYPE;

BEGIN

FOR CUSTOMER\_RECORD IN CUSTOMER\_CURSOR LOOP

VAR\_CUSTOMER\_ID := CUSTOMER\_RECORD.CUSTOMERID;

VAR\_BALANCE := CUSTOMER\_RECORD.BALANCE;

IF VAR\_BALANCE > 10000 THEN

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ID : ' || VAR\_CUSTOMER\_ID || ' HAS BALANCE GREATER THAN 10000');

UPDATE CUSTOMERS

SET ISVIP = 'TRUE'

WHERE CUSTOMERID = VAR\_CUSTOMER\_ID;

ELSE

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ID : ' || VAR\_CUSTOMER\_ID || ' HAS BALANCE LESSER THAN 10000');

UPDATE CUSTOMERS

SET ISVIP = 'FALSE'

WHERE CUSTOMERID = VAR\_CUSTOMER\_ID;

END IF;

END LOOP;

COMMIT;

END;

/

SELECT \* FROM CUSTOMERS;

-- SCENARIO 3

SET SERVEROUTPUT ON;

DECLARE

CURSOR CUR\_LOANS IS

SELECT L.LOANID, L.CUSTOMERID, C.NAME, L.ENDDATE

FROM LOANS L

JOIN CUSTOMERS C ON L.CUSTOMERID = C.CUSTOMERID

WHERE L.ENDDATE BETWEEN SYSDATE AND SYSDATE + 30;

V\_LOAN\_ID LOANS.LOANID%TYPE;

V\_CUSTOMER\_ID LOANS.CUSTOMERID%TYPE;

V\_CUSTOMER\_NAME CUSTOMERS.NAME%TYPE;

V\_END\_DATE LOANS.ENDDATE%TYPE;

V\_FOUND BOOLEAN := FALSE;

BEGIN

OPEN CUR\_LOANS;

LOOP

FETCH CUR\_LOANS INTO V\_LOAN\_ID, V\_CUSTOMER\_ID, V\_CUSTOMER\_NAME, V\_END\_DATE;

EXIT WHEN CUR\_LOANS%NOTFOUND;

V\_FOUND := TRUE;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || V\_LOAN\_ID || ' for customer ' || V\_CUSTOMER\_NAME || ' (ID: ' || V\_CUSTOMER\_ID || ') is due on ' || TO\_CHAR(V\_END\_DATE, 'YYYY-MM-DD'));

END LOOP;

CLOSE CUR\_LOANS;

IF NOT V\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No loans are due within the next 30 days.');

END IF;

END;

/

/\*

**Exercise 2: Error Handling**

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

**? Question: 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.**

**Scenario 2: Manage errors when updating employee salaries.**

**? Question: 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.**

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

**? Question: 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.**

\*/

-- SCENARIO 1

SELECT \* FROM ACCOUNTS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE SAFETRANSFERFUNDS(

P\_FROM\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE,

P\_TO\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE,

P\_AMOUNT IN NUMBER

) AS

V\_FROM\_BALANCE ACCOUNTS.BALANCE%TYPE;

V\_TO\_BALANCE ACCOUNTS.BALANCE%TYPE;

BEGIN

SELECT BALANCE INTO V\_FROM\_BALANCE

FROM ACCOUNTS

WHERE ACCOUNTID = P\_FROM\_ACCOUNT\_ID

FOR UPDATE;

SELECT BALANCE INTO V\_TO\_BALANCE

FROM ACCOUNTS

WHERE ACCOUNTID = P\_TO\_ACCOUNT\_ID

FOR UPDATE;

-- Check for sufficient funds

IF V\_FROM\_BALANCE < P\_AMOUNT THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

-- Perform the transfer

UPDATE ACCOUNTS

SET BALANCE = BALANCE - P\_AMOUNT,

LASTMODIFIED = SYSDATE

WHERE ACCOUNTID = P\_FROM\_ACCOUNT\_ID;

UPDATE ACCOUNTS

SET BALANCE = BALANCE + P\_AMOUNT,

LASTMODIFIED = SYSDATE

WHERE ACCOUNTID = P\_TO\_ACCOUNT\_ID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END SAFETRANSFERFUNDS;

/

EXEC SAFETRANSFERFUNDS(2,1,500);

SELECT \* FROM ACCOUNTS;

-- SCENARIO 2

SELECT \* FROM EMPLOYEES;

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE UPDATESALARY(

P\_EMPLOYEE\_ID IN EMPLOYEES.EMPLOYEEID%TYPE,

P\_PERCENTAGE IN NUMBER

) AS

V\_OLD\_SALARY EMPLOYEES.SALARY%TYPE;

BEGIN

-- Fetch the current salary

SELECT SALARY INTO V\_OLD\_SALARY

FROM EMPLOYEES

WHERE EMPLOYEEID = P\_EMPLOYEE\_ID;

-- Update the salary

UPDATE EMPLOYEES

SET SALARY = SALARY \* (1 + P\_PERCENTAGE / 100),

HIREDATE = SYSDATE

WHERE EMPLOYEEID = P\_EMPLOYEE\_ID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || P\_EMPLOYEE\_ID || ' does not exist.');

WHEN OTHERS THEN

ROLLBACK;

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

END UPDATESALARY;

/

EXEC UPDATESALARY(1,5);

EXEC UPDATESALARY(2,3);

SELECT \* FROM EMPLOYEES;

-- SCENARIO 3

SELECT \* FROM CUSTOMERS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE ADDNEWCUSTOMER(

P\_CUSTOMER\_ID IN CUSTOMERS.CUSTOMERID%TYPE,

P\_NAME IN CUSTOMERS.NAME%TYPE,

P\_DOB IN CUSTOMERS.DOB%TYPE,

P\_BALANCE IN CUSTOMERS.BALANCE%TYPE

) AS

BEGIN

-- Attempt to insert the new customer

DBMS\_OUTPUT.PUT\_LINE('INSERTING...');

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER\_ID : ' || P\_CUSTOMER\_ID);

DBMS\_OUTPUT.PUT\_LINE('NAME : ' || P\_NAME);

DBMS\_OUTPUT.PUT\_LINE('DOB : ' || P\_DOB);

DBMS\_OUTPUT.PUT\_LINE('BALANCE : ' || P\_BALANCE);

INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

VALUES (P\_CUSTOMER\_ID, P\_NAME, TO\_DATE(P\_DOB,'YYYY-MM-DD'), P\_BALANCE, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || P\_CUSTOMER\_ID || ' already exists.');

WHEN OTHERS THEN

ROLLBACK;

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

END ADDNEWCUSTOMER;

/

EXEC ADDNEWCUSTOMER(3,'ARKA PRATIM GHOSH','21-10-2002',50000);

SELECT \* FROM CUSTOMERS;

/\*

**Exercise 3: Stored Procedures**

**Scenario 1: The bank needs to process monthly interest for all savings accounts.**

**? Question: 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.**

**Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.**

**? Question: Write a stored procedure UpdateEmployeeBonus that updates the salary of employees**

**in a given department by adding a bonus percentage passed as a parameter.**

**Scenario 3: Customers should be able to transfer funds between their accounts.**

**? Question: 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.**

\*/

-- SCENARIO 1

SELECT \* FROM ACCOUNTS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE PROCESSMONTHLYINTEREST AS

BEGIN

UPDATE ACCOUNTS

SET BALANCE = BALANCE \* 1.01,

LASTMODIFIED = SYSDATE

WHERE ACCOUNTTYPE = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END PROCESSMONTHLYINTEREST;

/

EXEC PROCESSMONTHLYINTEREST();

SELECT \* FROM ACCOUNTS;

-- SCENARIO 2

SELECT \* FROM EMPLOYEES;

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE UPDATEEMPLOYEEBONUS(

P\_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE,

P\_BONUS\_PERCENTAGE IN NUMBER

) AS

BEGIN

UPDATE EMPLOYEES

SET SALARY = SALARY \* (1 + P\_BONUS\_PERCENTAGE / 100),

HIREDATE = SYSDATE

WHERE DEPARTMENT = P\_DEPARTMENT;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to employees in the ' || P\_DEPARTMENT || ' department.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UPDATEEMPLOYEEBONUS;

/

EXEC UPDATEEMPLOYEEBONUS('IT',5);

EXEC UPDATEEMPLOYEEBONUS('HR',3);

SELECT \* FROM EMPLOYEES;

-- SCENARIO 3

SELECT \* FROM ACCOUNTS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE TRANSFERFUNDS(

P\_FROM\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE,

P\_TO\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE,

P\_AMOUNT IN NUMBER

) AS

V\_FROM\_BALANCE ACCOUNTS.BALANCE%TYPE;

BEGIN

SELECT BALANCE INTO V\_FROM\_BALANCE

FROM ACCOUNTS

WHERE ACCOUNTID = P\_FROM\_ACCOUNT\_ID

FOR UPDATE;

-- Check for sufficient funds

IF V\_FROM\_BALANCE < P\_AMOUNT THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

-- Perform the transfer

UPDATE ACCOUNTS

SET BALANCE = BALANCE - P\_AMOUNT,

LASTMODIFIED = SYSDATE

WHERE ACCOUNTID = P\_FROM\_ACCOUNT\_ID;

UPDATE ACCOUNTS

SET BALANCE = BALANCE + P\_AMOUNT,

LASTMODIFIED = SYSDATE

WHERE ACCOUNTID = P\_TO\_ACCOUNT\_ID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || P\_AMOUNT || ' from account ' || P\_FROM\_ACCOUNT\_ID || ' to account ' || P\_TO\_ACCOUNT\_ID || ' completed successfully.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END TRANSFERFUNDS;

/

EXEC TRANSFERFUNDS(1,2,100);

SELECT \* FROM ACCOUNTS;

/\*

**Exercise 4: Functions**

**Scenario 1: Calculate the age of customers for eligibility checks.**

**? Question: Write a function CalculateAge that takes a customer's date of birth as input and**

**returns their age in years.**

**Scenario 2: The bank needs to compute the monthly installment for a loan.**

**? Question: Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate,**

**and loan duration in years as input and returns the monthly installment amount.**

**Scenario 3: Check if a customer has sufficient balance before making a transaction.**

**? Question: 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.**

\*/

-- SCENARIO 1

DELETE FROM CUSTOMERS WHERE CUSTOMERID = 3;

SELECT \* FROM CUSTOMERS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE FUNCTION CALCULATEAGE(

P\_DOB IN DATE

) RETURN NUMBER IS

V\_AGE NUMBER;

BEGIN

V\_AGE := TRUNC((SYSDATE - P\_DOB) / 365);

RETURN V\_AGE;

END CALCULATEAGE;

/

SET SERVEROUTPUT ON;

DECLARE

CURSOR CURSOR\_CUST IS SELECT CUSTOMERID, DOB FROM CUSTOMERS;

V\_CUSTOMERID CUSTOMERS.CUSTOMERID%TYPE;

V\_DOB CUSTOMERS.DOB%TYPE;

V\_AGE NUMBER;

BEGIN

OPEN CURSOR\_CUST;

LOOP

FETCH CURSOR\_CUST INTO V\_CUSTOMERID, V\_DOB;

EXIT WHEN CURSOR\_CUST%NOTFOUND;

V\_AGE := CALCULATEAGE(V\_DOB);

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ID : ' || V\_CUSTOMERID || ' AGE : ' || V\_AGE);

END LOOP;

CLOSE CURSOR\_CUST;

END;

/

-- SCENARIO 2

SELECT \* FROM LOANS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE FUNCTION CALCULATEMONTHLYINSTALLMENT(

P\_LOAN\_AMOUNT IN NUMBER,

P\_INTEREST\_RATE IN NUMBER,

P\_LOAN\_DURATION\_YEARS IN NUMBER

) RETURN NUMBER IS

V\_MONTHLY\_RATE NUMBER;

V\_NUM\_PAYMENTS NUMBER;

V\_MONTHLY\_INSTALLMENT NUMBER;

BEGIN

V\_MONTHLY\_RATE := P\_INTEREST\_RATE / 12 / 100;

V\_NUM\_PAYMENTS := P\_LOAN\_DURATION\_YEARS \* 12;

IF V\_MONTHLY\_RATE = 0 THEN

V\_MONTHLY\_INSTALLMENT := P\_LOAN\_AMOUNT / V\_NUM\_PAYMENTS;

ELSE

V\_MONTHLY\_INSTALLMENT := P\_LOAN\_AMOUNT \* V\_MONTHLY\_RATE / (1 - POWER(1 + V\_MONTHLY\_RATE, -V\_NUM\_PAYMENTS));

END IF;

RETURN V\_MONTHLY\_INSTALLMENT;

END CALCULATEMONTHLYINSTALLMENT;

/

SET SERVEROUTPUT ON;

DECLARE

CURSOR LOAN\_CUR IS SELECT \* FROM LOANS;

V\_DATA LOANS%ROWTYPE;

V\_DURATION NUMBER;

V\_MONTHLYINSTALLMENT NUMBER;

BEGIN

OPEN LOAN\_CUR;

LOOP

FETCH LOAN\_CUR INTO V\_DATA;

EXIT WHEN LOAN\_CUR%NOTFOUND;

V\_DURATION := TRUNC((V\_DATA.ENDDATE - V\_DATA.STARTDATE)/365);

V\_MONTHLYINSTALLMENT := TRUNC(CALCULATEMONTHLYINSTALLMENT(V\_DATA.LOANAMOUNT, V\_DATA.INTERESTRATE, V\_DURATION),2);

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ID : ' || V\_DATA.CUSTOMERID || ' MONTHLY INSTALLAMENT : ' || V\_MONTHLYINSTALLMENT);

END LOOP;

CLOSE LOAN\_CUR;

END;

/

-- SCENARIO 3

SELECT \* FROM ACCOUNTS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE FUNCTION HASSUFFICIENTBALANCE(

P\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE,

P\_AMOUNT IN NUMBER

) RETURN BOOLEAN IS

V\_BALANCE ACCOUNTS.BALANCE%TYPE;

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;

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Error checking balance: ' || SQLERRM);

END HASSUFFICIENTBALANCE;

/

SET SERVEROUTPUT ON;

DECLARE

V\_ACCOUNTID ACCOUNTS.ACCOUNTID%TYPE := &ACCOUNTID;

V\_AMOUNT NUMBER := &AMOUNT;

V\_HAS BOOLEAN;

BEGIN

V\_HAS := HASSUFFICIENTBALANCE(V\_ACCOUNTID, V\_AMOUNT);

IF V\_HAS = TRUE THEN DBMS\_OUTPUT.PUT\_LINE(V\_ACCOUNTID || ' HAS SUFFICIENT AMOUNT');

ELSE DBMS\_OUTPUT.PUT\_LINE(V\_ACCOUNTID || ' DOES NOT HAVE SUFFICIENT AMOUNT');

END IF;

END;

/

/\*

**Exercise 5: Triggers**

**Scenario 1: Automatically update the last modified date when a customer's record is updated.**

**? Question: Write a trigger UpdateCustomerLastModified that updates the LastModified column of the Customers**

**table to the current date whenever a customer's record is updated.**

**Scenario 2: Maintain an audit log for all transactions.**

**? Question: Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a**

**transaction is inserted into the Transactions table.**

**Scenario 3: Enforce business rules on deposits and withdrawals.**

**? Question: Write a trigger CheckTransactionRules that ensures withdrawals do not exceed the balance and deposits**

**are positive before inserting a record into the Transactions table.**

\*/

-- SCENARIO 1

SELECT \* FROM CUSTOMERS;

SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER UPDATECUSTOMERLASTMODIFIED

BEFORE UPDATE ON CUSTOMERS

FOR EACH ROW

BEGIN

:NEW.LASTMODIFIED := SYSDATE;

DBMS\_OUTPUT.PUT\_LINE('LAST MODIFIED UPDATED');

END UPDATECUSTOMERLASTMODIFIED;

/

UPDATE CUSTOMERS SET NAME = 'JOHN DOE' WHERE CUSTOMERID = 1;

-- SCENARIO 2

CREATE TABLE AUDITLOG (

LOGID NUMBER PRIMARY KEY,

TRANSACTIONID NUMBER,

ACCOUNTID NUMBER,

TRANSACTIONDATE DATE,

AMOUNT NUMBER,

TRANSACTIONTYPE VARCHAR2(10),

LOGTIMESTAMP DATE DEFAULT SYSDATE

);

SELECT \* FROM TRANSACTIONS;

CREATE SEQUENCE AUDITLOG\_SEQ

START WITH 1

INCREMENT BY 1;

SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER LOGTRANSACTIONS

AFTER INSERT ON TRANSACTIONS

FOR EACH ROW

BEGIN

INSERT INTO AUDITLOG (LOGID, TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

VALUES (AUDITLOG\_SEQ.NEXTVAL, :NEW.TRANSACTIONID, :NEW.ACCOUNTID, SYSDATE, :NEW.AMOUNT, :NEW.TRANSACTIONTYPE);

DBMS\_OUTPUT.PUT\_LINE('INSERT SUCCESSFUL');

END LOGTRANSACTIONS;

/

INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

VALUES (6, 2, SYSDATE, 600, 'Deposit');

SELECT \* FROM AUDITLOG;

SELECT \* FROM TRANSACTIONS;

-- SCENARIO 3

SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER CHECKTRANSACTIONRULES

BEFORE INSERT ON TRANSACTIONS

FOR EACH ROW

DECLARE

V\_BALANCE ACCOUNTS.BALANCE%TYPE;

BEGIN

SELECT BALANCE INTO V\_BALANCE

FROM ACCOUNTS

WHERE ACCOUNTID = :NEW.ACCOUNTID

FOR UPDATE;

IF :NEW.TRANSACTIONTYPE = 'Withdrawal' THEN

IF :NEW.AMOUNT > V\_BALANCE THEN

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

END IF;

ELSIF :NEW.TRANSACTIONTYPE = 'Deposit' THEN

IF :NEW.AMOUNT <= 0 THEN

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

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid transaction type.');

END IF;

END CHECKTRANSACTIONRULES;

/

SELECT \* FROM ACCOUNTS;

SELECT \* FROM CUSTOMERS;

INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)

VALUES (4, 1, 'Recurring', 3500, SYSDATE);

/\*

**Exercise 6: Cursors**

**Scenario 1: Generate monthly statements for all customers.**

**? Question: 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.**

**Scenario 2: Apply annual fee to all accounts.**

**? Question: Write a PL/SQL block using an explicit cursor ApplyAnnualFee that deducts an annual maintenance**

**fee from the balance of all accounts.**

**Scenario 3: Update the interest rate for all loans based on a new policy.**

**? Question: Write a PL/SQL block using an explicit cursor UpdateLoanInterestRates that fetches all loans and**

**updates their interest rates based on the new policy.**

\*/

-- SCENARIO 1

SET SERVEROUTPUT ON;

DECLARE

CURSOR CUR\_MONTHLY\_TRANSACTIONS IS

SELECT C.CUSTOMERID, C.NAME, T.TRANSACTIONDATE, T.AMOUNT, T.TRANSACTIONTYPE

FROM CUSTOMERS C

JOIN ACCOUNTS A ON C.CUSTOMERID = A.CUSTOMERID

JOIN TRANSACTIONS T ON A.ACCOUNTID = T.ACCOUNTID

WHERE TRUNC(T.TRANSACTIONDATE, 'MM') = TRUNC(SYSDATE, 'MM')

ORDER BY C.CUSTOMERID, T.TRANSACTIONDATE;

V\_CUSTOMER\_ID CUSTOMERS.CUSTOMERID%TYPE;

V\_CUSTOMER\_NAME CUSTOMERS.NAME%TYPE;

V\_TRANSACTION\_DATE TRANSACTIONS.TRANSACTIONDATE%TYPE;

V\_AMOUNT TRANSACTIONS.AMOUNT%TYPE;

V\_TRANSACTION\_TYPE TRANSACTIONS.TRANSACTIONTYPE%TYPE;

BEGIN

OPEN CUR\_MONTHLY\_TRANSACTIONS;

LOOP

FETCH CUR\_MONTHLY\_TRANSACTIONS INTO V\_CUSTOMER\_ID, V\_CUSTOMER\_NAME, V\_TRANSACTION\_DATE, V\_AMOUNT, V\_TRANSACTION\_TYPE;

EXIT WHEN CUR\_MONTHLY\_TRANSACTIONS%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || V\_CUSTOMER\_ID || ', Name: ' || V\_CUSTOMER\_NAME);

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || TO\_CHAR(V\_TRANSACTION\_DATE, 'YYYY-MM-DD') || ', Amount: ' || V\_AMOUNT || ', Type: ' || V\_TRANSACTION\_TYPE);

END LOOP;

CLOSE CUR\_MONTHLY\_TRANSACTIONS;

END;

/

-- SCENARIO 2

SET SERVEROUTPUT ON;

DECLARE

CURSOR CUR\_ACCOUNTS IS

SELECT ACCOUNTID, BALANCE

FROM ACCOUNTS;

V\_ACCOUNT\_ID ACCOUNTS.ACCOUNTID%TYPE;

V\_BALANCE ACCOUNTS.BALANCE%TYPE;

V\_ANNUAL\_FEE CONSTANT NUMBER := 50; -- Annual fee amount

BEGIN

OPEN CUR\_ACCOUNTS;

LOOP

FETCH CUR\_ACCOUNTS INTO V\_ACCOUNT\_ID, V\_BALANCE;

EXIT WHEN CUR\_ACCOUNTS%NOTFOUND;

UPDATE ACCOUNTS

SET BALANCE = BALANCE - V\_ANNUAL\_FEE,

LASTMODIFIED = SYSDATE

WHERE ACCOUNTID = V\_ACCOUNT\_ID;

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || V\_ANNUAL\_FEE || ' deducted from Account ID: ' || V\_ACCOUNT\_ID);

END LOOP;

CLOSE CUR\_ACCOUNTS;

COMMIT;

END;

/

-- SCENARIO 3

SET SERVEROUTPUT ON;

DECLARE

CURSOR CUR\_LOANS IS

SELECT LOANID, INTERESTRATE

FROM LOANS;

V\_LOAN\_ID LOANS.LOANID%TYPE;

V\_INTEREST\_RATE LOANS.INTERESTRATE%TYPE;

V\_NEW\_INTEREST\_RATE NUMBER;

V\_NEW\_POLICY NUMBER := 2;

FUNCTION CALCULATENEWINTERESTRATE(OLD\_RATE NUMBER) RETURN NUMBER IS

BEGIN

RETURN OLD\_RATE \* (1 + (V\_NEW\_POLICY / 100));

END CALCULATENEWINTERESTRATE;

BEGIN

OPEN CUR\_LOANS;

LOOP

FETCH CUR\_LOANS INTO V\_LOAN\_ID, V\_INTEREST\_RATE;

EXIT WHEN CUR\_LOANS%NOTFOUND;

V\_NEW\_INTEREST\_RATE := CALCULATENEWINTERESTRATE(V\_INTEREST\_RATE);

UPDATE LOANS

SET INTERESTRATE = V\_NEW\_INTEREST\_RATE

WHERE LOANID = V\_LOAN\_ID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || V\_LOAN\_ID || ' interest rate updated to ' || V\_NEW\_INTEREST\_RATE);

END LOOP;

CLOSE CUR\_LOANS;

COMMIT;

END;

/

/\*

**Exercise 7: Packages**

**Scenario 1: Group all customer-related procedures and functions into a package.**

**? Question: Create a package CustomerManagement with procedures for adding a new customer,**

**updating customer details, and a function to get customer balance.**

**Scenario 2: Create a package to manage employee data.**

**? Question: Write a package EmployeeManagement with procedures to hire new employees, update employee details,**

**and a function to calculate annual salary.**

**Scenario 3: Group all account-related operations into a package.**

**? Question: 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.**

\*/

-- SCENARIO 1

SET SERVEROUTPUT ON;

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddNewCustomer(

p\_customer\_id IN CUSTOMERS.CUSTOMERID%TYPE,

p\_name IN CUSTOMERS.NAME%TYPE,

p\_dob IN CUSTOMERS.DOB%TYPE,

p\_balance IN CUSTOMERS.BALANCE%TYPE

);

PROCEDURE UpdateCustomerDetails(

p\_customer\_id IN CUSTOMERS.CUSTOMERID%TYPE,

p\_name IN CUSTOMERS.NAME%TYPE,

p\_dob IN CUSTOMERS.DOB%TYPE,

p\_balance IN CUSTOMERS.BALANCE%TYPE

);

FUNCTION GetCustomerBalance(

p\_customer\_id IN CUSTOMERS.CUSTOMERID%TYPE

) RETURN CUSTOMERS.BALANCE%TYPE;

END CustomerManagement;

/

SET SERVEROUTPUT ON;

CREATE OR REPLACE PACKAGE BODY CUSTOMERMANAGEMENT IS

PROCEDURE ADDNEWCUSTOMER(

P\_CUSTOMER\_ID IN CUSTOMERS.CUSTOMERID%TYPE,

P\_NAME IN CUSTOMERS.NAME%TYPE,

P\_DOB IN CUSTOMERS.DOB%TYPE,

P\_BALANCE IN CUSTOMERS.BALANCE%TYPE

) IS

BEGIN

INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

VALUES (P\_CUSTOMER\_ID, P\_NAME, P\_DOB, P\_BALANCE, SYSDATE);

END ADDNEWCUSTOMER;

PROCEDURE UPDATECUSTOMERDETAILS(

P\_CUSTOMER\_ID IN CUSTOMERS.CUSTOMERID%TYPE,

P\_NAME IN CUSTOMERS.NAME%TYPE,

P\_DOB IN CUSTOMERS.DOB%TYPE,

P\_BALANCE IN CUSTOMERS.BALANCE%TYPE

) IS

BEGIN

UPDATE CUSTOMERS

SET NAME = P\_NAME,

DOB = P\_DOB,

BALANCE = P\_BALANCE,

LASTMODIFIED = SYSDATE

WHERE CUSTOMERID = P\_CUSTOMER\_ID;

END UPDATECUSTOMERDETAILS;

FUNCTION GETCUSTOMERBALANCE(

P\_CUSTOMER\_ID IN CUSTOMERS.CUSTOMERID%TYPE

) RETURN CUSTOMERS.BALANCE%TYPE IS

V\_BALANCE CUSTOMERS.BALANCE%TYPE;

BEGIN

SELECT BALANCE INTO V\_BALANCE

FROM CUSTOMERS

WHERE CUSTOMERID = P\_CUSTOMER\_ID;

RETURN V\_BALANCE;

END GETCUSTOMERBALANCE;

END CUSTOMERMANAGEMENT;

/

-- SCENARIO 2

SET SERVEROUTPUT ON;

CREATE OR REPLACE PACKAGE EMPLOYEEMANAGEMENT IS

PROCEDURE HIREEMPLOYEE(

P\_EMPLOYEE\_ID IN EMPLOYEES.EMPLOYEEID%TYPE,

P\_NAME IN EMPLOYEES.NAME%TYPE,

P\_POSITION IN EMPLOYEES.POSITION%TYPE,

P\_SALARY IN EMPLOYEES.SALARY%TYPE,

P\_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE,

P\_HIRE\_DATE IN EMPLOYEES.HIREDATE%TYPE

);

PROCEDURE UPDATEEMPLOYEEDETAILS(

P\_EMPLOYEE\_ID IN EMPLOYEES.EMPLOYEEID%TYPE,

P\_NAME IN EMPLOYEES.NAME%TYPE,

P\_POSITION IN EMPLOYEES.POSITION%TYPE,

P\_SALARY IN EMPLOYEES.SALARY%TYPE,

P\_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE

);

FUNCTION CALCULATEANNUALSALARY(

P\_EMPLOYEE\_ID IN EMPLOYEES.EMPLOYEEID%TYPE

) RETURN NUMBER;

END EMPLOYEEMANAGEMENT;

/

SET SERVEROUTPUT ON;

CREATE OR REPLACE PACKAGE BODY EMPLOYEEMANAGEMENT IS

PROCEDURE HIREEMPLOYEE(

P\_EMPLOYEE\_ID IN EMPLOYEES.EMPLOYEEID%TYPE,

P\_NAME IN EMPLOYEES.NAME%TYPE,

P\_POSITION IN EMPLOYEES.POSITION%TYPE,

P\_SALARY IN EMPLOYEES.SALARY%TYPE,

P\_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE,

P\_HIRE\_DATE IN EMPLOYEES.HIREDATE%TYPE

) IS

BEGIN

INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

VALUES (P\_EMPLOYEE\_ID, P\_NAME, P\_POSITION, P\_SALARY, P\_DEPARTMENT, P\_HIRE\_DATE);

END HIREEMPLOYEE;

PROCEDURE UPDATEEMPLOYEEDETAILS(

P\_EMPLOYEE\_ID IN EMPLOYEES.EMPLOYEEID%TYPE,

P\_NAME IN EMPLOYEES.NAME%TYPE,

P\_POSITION IN EMPLOYEES.POSITION%TYPE,

P\_SALARY IN EMPLOYEES.SALARY%TYPE,

P\_DEPARTMENT IN EMPLOYEES.DEPARTMENT%TYPE

) IS

BEGIN

UPDATE EMPLOYEES

SET NAME = P\_NAME,

POSITION = P\_POSITION,

SALARY = P\_SALARY,

DEPARTMENT = P\_DEPARTMENT

WHERE EMPLOYEEID = P\_EMPLOYEE\_ID;

END UPDATEEMPLOYEEDETAILS;

FUNCTION CALCULATEANNUALSALARY(

P\_EMPLOYEE\_ID IN EMPLOYEES.EMPLOYEEID%TYPE

) RETURN NUMBER IS

V\_SALARY EMPLOYEES.SALARY%TYPE;

BEGIN

SELECT SALARY INTO V\_SALARY

FROM EMPLOYEES

WHERE EMPLOYEEID = P\_EMPLOYEE\_ID;

RETURN V\_SALARY \* 12; -- Assuming salary is monthly

END CALCULATEANNUALSALARY;

END EMPLOYEEMANAGEMENT;

/

-- SCENARIO 3

SET SERVEROUTPUT ON;

CREATE OR REPLACE PACKAGE ACCOUNTOPERATIONS IS

PROCEDURE OPENNEWACCOUNT(

P\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE,

P\_CUSTOMER\_ID IN ACCOUNTS.CUSTOMERID%TYPE,

P\_ACCOUNT\_TYPE IN ACCOUNTS.ACCOUNTTYPE%TYPE,

P\_BALANCE IN ACCOUNTS.BALANCE%TYPE

);

PROCEDURE CLOSEACCOUNT(

P\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE

);

FUNCTION GETTOTALBALANCE(

P\_CUSTOMER\_ID IN ACCOUNTS.CUSTOMERID%TYPE

) RETURN NUMBER;

END ACCOUNTOPERATIONS;

/

SET SERVEROUTPUT ON;

CREATE OR REPLACE PACKAGE BODY ACCOUNTOPERATIONS IS

PROCEDURE OPENNEWACCOUNT(

P\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE,

P\_CUSTOMER\_ID IN ACCOUNTS.CUSTOMERID%TYPE,

P\_ACCOUNT\_TYPE IN ACCOUNTS.ACCOUNTTYPE%TYPE,

P\_BALANCE IN ACCOUNTS.BALANCE%TYPE

) IS

BEGIN

INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)

VALUES (P\_ACCOUNT\_ID, P\_CUSTOMER\_ID, P\_ACCOUNT\_TYPE, P\_BALANCE, SYSDATE);

END OPENNEWACCOUNT;

PROCEDURE CLOSEACCOUNT(

P\_ACCOUNT\_ID IN ACCOUNTS.ACCOUNTID%TYPE

) IS

BEGIN

DELETE FROM ACCOUNTS

WHERE ACCOUNTID = P\_ACCOUNT\_ID;

END CLOSEACCOUNT;

FUNCTION GETTOTALBALANCE (

P\_CUSTOMER\_ID IN ACCOUNTS.CUSTOMERID%TYPE

) 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;

END GETTOTALBALANCE;

END ACCOUNTOPERATIONS;

/