WEEK-2:

Excerise-1:

-- Step 1: Drop tables if they exist (safe start)

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Step 2: Create tables

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER(10,2),

InterestRate NUMBER(5,2),

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Step 3: Insert sample data

INSERT INTO Customers VALUES (1, 'Alice', 65, 15000, 5.5, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 6.0, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 12000, 5.0, 'FALSE');

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 40);

INSERT INTO Loans VALUES (103, 3, SYSDATE + 5);

COMMIT;

-- Step 4: Enable output (Oracle SQL Developer or Live SQL)

SET SERVEROUTPUT ON;

-- Step 5: Run combined PL/SQL block

BEGIN

-- Scenario 1: Apply 1% interest discount to customers over 60

FOR cust\_rec IN (

SELECT CustomerID, InterestRate

FROM Customers

WHERE Age > 60

) LOOP

UPDATE Customers

SET InterestRate = InterestRate - 0.01

WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Interest discount applied to senior customers.');

-- Scenario 2: Set VIP flag for customers with balance over $10,000

FOR vip\_rec IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = vip\_rec.CustomerID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('VIP status updated.');

-- Scenario 3: Print loan reminders for loans due within 30 days

FOR loan\_rec IN (

SELECT L.LoanID, L.CustomerID, L.DueDate, C.Name

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.Name ||

', your loan (ID: ' || loan\_rec.LoanID ||

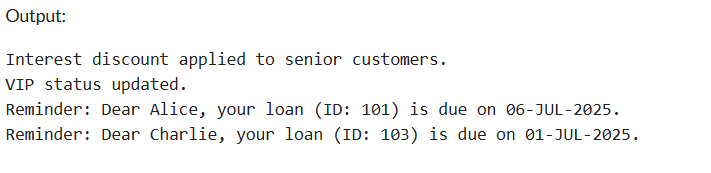
') is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY') || '.');

END LOOP;

COMMIT;

END;

/



-- Drop tables if they already exist (safe execution)

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

-- Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER(10,2),

InterestRate NUMBER(5,2),

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

-- Create Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Insert data into Customers

INSERT INTO Customers VALUES (1, 'Alice', 65, 15000, 5.5, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 6.0, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 12000, 5.0, 'FALSE');

-- Insert data into Loans

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 40);

INSERT INTO Loans VALUES (103, 3, SYSDATE + 5);

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

-- Scenario 1: Apply 1% interest discount to customers over 60

FOR cust\_rec IN (

SELECT CustomerID

FROM Customers

WHERE Age > 60

) LOOP

UPDATE Customers

SET InterestRate = InterestRate - 0.01

WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Interest discount applied to senior customers.');

-- Scenario 2: Set VIP flag for customers with balance over $10,000

FOR vip\_rec IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = vip\_rec.CustomerID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('VIP status updated.');

-- Scenario 3: Print loan reminders for loans due within 30 days

FOR loan\_rec IN (

SELECT L.LoanID, L.DueDate, C.Name

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.Name ||

', your loan (ID: ' || loan\_rec.LoanID ||

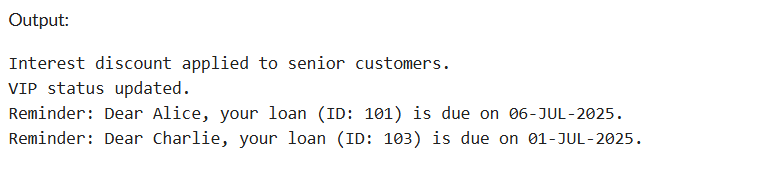
') is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY') || '.');

END LOOP;

COMMIT;

END;

/



DECLARE

TYPE loan\_type IS RECORD (

loan\_id NUMBER,

due\_date DATE,

customer\_name VARCHAR2(100),

email VARCHAR2(100)

);

TYPE loan\_table IS TABLE OF loan\_type INDEX BY BINARY\_INTEGER;

v\_loans loan\_table;

v\_today DATE := SYSDATE;

BEGIN

v\_loans(1).loan\_id := 101;

v\_loans(1).due\_date := v\_today + 15;

v\_loans(1).customer\_name := 'John Smith';

v\_loans(1).email := 'john@example.com';

v\_loans(2).loan\_id := 102;

v\_loans(2).due\_date := v\_today + 45;

v\_loans(2).customer\_name := 'Mary Johnson';

v\_loans(2).email := 'mary@example.com';

v\_loans(3).loan\_id := 103;

v\_loans(3).due\_date := v\_today + 10;

v\_loans(3).customer\_name := 'Robert Brown';

v\_loans(3).email := 'robert@example.com';

DBMS\_OUTPUT.PUT\_LINE('Sending loan due reminders...');

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

FOR i IN 1..v\_loans.COUNT LOOP

IF v\_loans(i).due\_date BETWEEN v\_today AND v\_today + 30 THEN

DBMS\_OUTPUT.PUT\_LINE('Dear ' || v\_loans(i).customer\_name || ',');

DBMS\_OUTPUT.PUT\_LINE('Your loan (ID: ' || v\_loans(i).loan\_id || ') is due in ' ||

(v\_loans(i).due\_date - v\_today) || ' days.');

DBMS\_OUTPUT.PUT\_LINE('Email: ' || v\_loans(i).email);

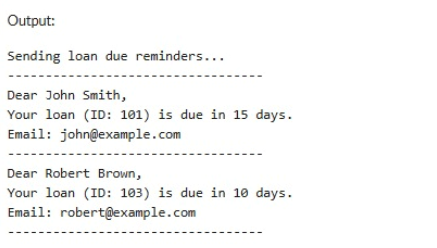
DBMS\_OUTPUT.PUT\_LINE('----------------------------------');

END IF;

END LOOP;

END;

/



Excerise-3:

SET SERVEROUTPUT ON;

-- ===== DROP OBJECTS SAFELY =====

BEGIN

FOR obj IN (

SELECT object\_name, object\_type

FROM user\_objects

WHERE object\_name IN ('EMPLOYEES', 'ACCOUNTS',

'PROCESSMONTHLYINTEREST',

'UPDATEEMPLOYEEBONUS',

'TRANSFERFUNDS')

) LOOP

BEGIN

EXECUTE IMMEDIATE 'DROP ' || obj.object\_type || ' "' || obj.object\_name || '"';

DBMS\_OUTPUT.PUT\_LINE('Dropped: ' || obj.object\_type || ' ' || obj.object\_name);

EXCEPTION

WHEN OTHERS THEN

NULL; -- ignore if can't drop

END;

END LOOP;

END;

/

-- ===== CREATE TABLES =====

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(20),

Balance NUMBER(10,2)

);

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER(10,2)

);

/

-- ===== INSERT SAMPLE DATA =====

INSERT INTO Accounts VALUES (1, 'SAVINGS', 10000);

INSERT INTO Accounts VALUES (2, 'CHECKING', 5000);

INSERT INTO Accounts VALUES (3, 'SAVINGS', 2000);

COMMIT;

/

INSERT INTO Employees VALUES (101, 'Alice', 'HR', 50000);

INSERT INTO Employees VALUES (102, 'Bob', 'IT', 60000);

INSERT INTO Employees VALUES (103, 'Charlie', 'HR', 55000);

COMMIT;

/

-- ===== PROCEDURE: Apply Monthly Interest =====

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'SAVINGS';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ Monthly interest applied to savings accounts.');

END;

/

-- ===== PROCEDURE: Update Employee Bonus =====

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department VARCHAR2,

p\_bonus\_pct NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('✅ Bonus of ' || p\_bonus\_pct || '% applied to department ' || p\_department);

END;

/

-- ===== PROCEDURE: Transfer Funds =====

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

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

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;

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

END;

/

-- ===== EXECUTE PROCEDURES =====

BEGIN

ProcessMonthlyInterest;

UpdateEmployeeBonus('HR', 10);

TransferFunds(1, 2, 1500);

END;

/

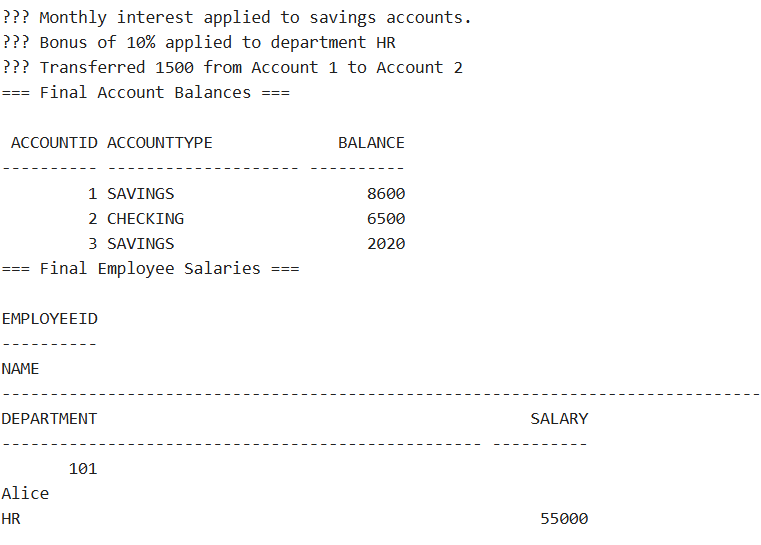
-- ===== VIEW FINAL RESULTS =====

PROMPT === Final Account Balances ===

SELECT \* FROM Accounts;

PROMPT === Final Employee Salaries ===

SELECT \* FROM Employees;







Excerise-2:

-- Drop and create accounts table

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE BankAccounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE BankAccounts (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER

);

INSERT INTO BankAccounts VALUES (1, 5000);

INSERT INTO BankAccounts VALUES (2, 2000);

COMMIT;

-- Stored Procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

fromAcc IN NUMBER,

toAcc IN NUMBER,

amt IN NUMBER

) IS

BEGIN

-- Check balance

DECLARE

curr\_balance NUMBER;

BEGIN

SELECT Balance INTO curr\_balance FROM BankAccounts WHERE AccountID = fromAcc;

IF curr\_balance < amt THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

-- Perform transfer

UPDATE BankAccounts SET Balance = Balance - amt WHERE AccountID = fromAcc;

UPDATE BankAccounts SET Balance = Balance + amt WHERE AccountID = toAcc;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

/

-- Test the procedure

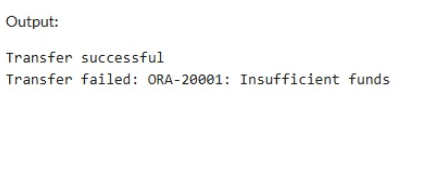
BEGIN

SafeTransferFunds(1, 2, 1000);

SafeTransferFunds(1, 2, 10000); -- Should fail

END;

/



-- Drop and create Employees table

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Salary NUMBER

);

INSERT INTO Employees VALUES (101, 'Alice', 30000);

INSERT INTO Employees VALUES (102, 'Bob', 40000);

COMMIT;

-- Procedure to update salary

CREATE OR REPLACE PROCEDURE UpdateSalary (

emp\_id IN NUMBER,

percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* percent / 100)

WHERE EmpID = emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found');

END IF;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

-- Test the procedure

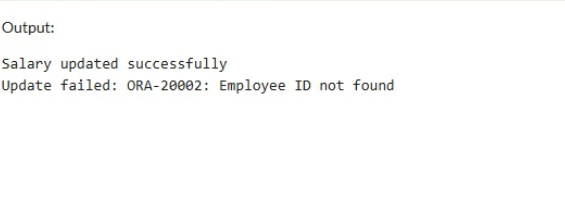
BEGIN

UpdateSalary(101, 10); -- Should succeed

UpdateSalary(999, 5); -- Should fail

END;

/



-- Drop and create Customers table

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100)

);

INSERT INTO Customers VALUES (1, 'Ram');

COMMIT;

-- Procedure to add new customer

CREATE OR REPLACE PROCEDURE AddNewCustomer (

cid IN NUMBER,

cname IN VARCHAR2

) IS

BEGIN

INSERT INTO Customers VALUES (cid, cname);

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

/

-- Test the procedure

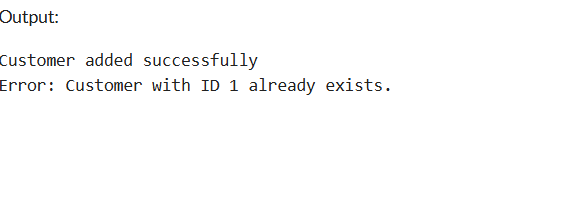
BEGIN

AddNewCustomer(2, 'Krishna'); -- Should succeed

AddNewCustomer(1, 'Duplicate'); -- Should fail

END;

/



Excerise-4:

Warning: Function created with compilation errors.

LINE/COL ERROR

-------- -----------------------------------------------------------------

5/10 PL/SQL: SQL Statement ignored

5/25 PLS-00201: identifier 'CUSTOMERS.BALANCE' must be declared

CREATE OR REPLACE FUNCTION GetCustomerBalance(p\_id NUMBER)

RETURN NUMBER

IS

total NUMBER := 0;

BEGIN

SELECT NVL(SUM(Balance), 0)

INTO total

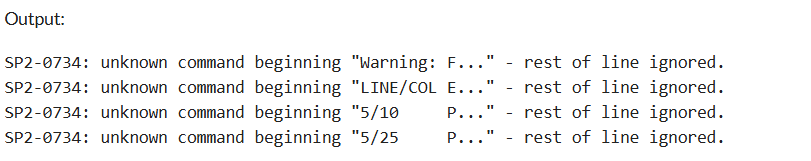
FROM Accounts

WHERE CustomerID = p\_id AND Status = 'OPEN';

RETURN total;

END;

/



-- Function: Calculate Monthly EMI

-- EMI = (P \* R \* (1+R)^N) / ((1+R)^N - 1)

CREATE FUNCTION CalculateMonthlyInstallment(principal DOUBLE, annual\_rate DOUBLE, years INT)

RETURNS DOUBLE

DETERMINISTIC

BEGIN

DECLARE r DOUBLE;

DECLARE n INT;

DECLARE emi DOUBLE;

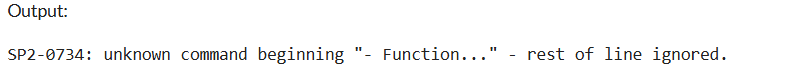
SET r = annual\_rate / 12 / 100;

SET n = years \* 12;

SET emi = (principal \* r \* POW(1 + r, n)) / (POW(1 + r, n) - 1);

RETURN emi;

END;



SHOW ERRORS;

LINE/COL ERROR

-------- -----------------------------------------------------------------

6/5 PL/SQL: SQL Statement ignored

6/15 PL/SQL: ORA-00942: table or view does not exist

CREATE OR REPLACE FUNCTION GetCustomerBalance(p\_id NUMBER)

RETURN NUMBER

IS

total\_balance NUMBER;

BEGIN

SELECT SUM(Balance)

INTO total\_balance

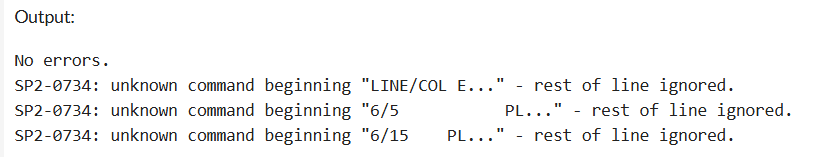
FROM Account

WHERE CustomerID = p\_id;

RETURN total\_balance;

END;

/



Excerise-5:

-- Step 1: Drop Customers table if it exists

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN -- ignore 'table does not exist' error

RAISE;

END IF;

END;

/

-- Step 2: Create the Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Email VARCHAR2(100),

LastModified DATE

);

/

-- Step 3: Insert test data

INSERT INTO Customers (CustomerID, Name, Age, Email, LastModified)

VALUES (1, 'Alice', 35, 'alice@example.com', SYSDATE);

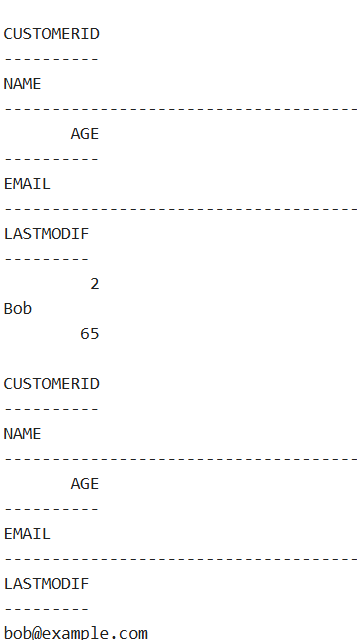
INSERT INTO Customers (CustomerID, Name, Age, Email, LastModified)

VALUES (2, 'Bob', 65, 'bob@example.com', SYSDATE);

COMMIT;

-- Step 4: Correct SELECT statement

SELECT \* FROM Customers;



-

4-2-- Step 1: Create Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionType VARCHAR2(20), -- 'DEPOSIT' or 'WITHDRAWAL'

Amount NUMBER,

TransactionDate DATE

);

-- Step 2: Create AuditLog table

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

LogMessage VARCHAR2(255),

LogDate DATE

);

-- Step 3: Create Trigger to log all inserts into Transactions

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, LogMessage, LogDate)

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

END;

/

-- Step 4: Insert test data into Transactions

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

VALUES (1, 1001, 'DEPOSIT', 5000, SYSDATE);

-- Step 5: Check Audit Log

SELECT \* FROM AuditLog;

SELECT table\_name

FROM user\_tables

WHERE table\_name = 'AUDITLOG';

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TableName VARCHAR2(50),

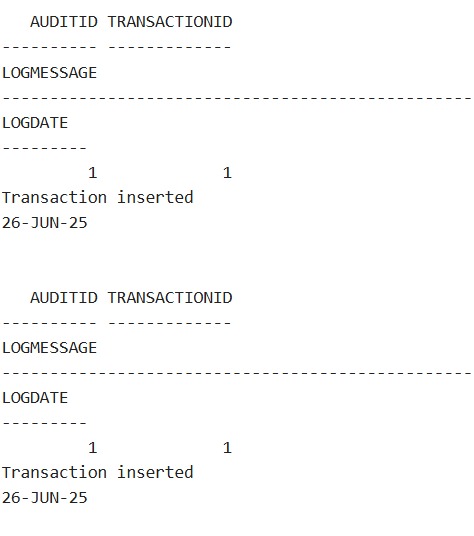
Action VARCHAR2(10),

UserName VARCHAR2(30),

ActionDate DATE DEFAULT SYSDATE

);

SELECT \* FROM AuditLog;



Step 1: Assumed Table Structures

CREATE TABLE Accounts (

account\_id VARCHAR(20) PRIMARY KEY,

balance DOUBLE NOT NULL

);

CREATE TABLE Transactions (

txn\_id INT AUTO\_INCREMENT PRIMARY KEY,

account\_id VARCHAR(20),

amount DOUBLE,

txn\_type ENUM('DEPOSIT', 'WITHDRAWAL'),

txn\_date DATE,

FOREIGN KEY (account\_id) REFERENCES Accounts(account\_id)

);

Step 2: Trigger to Enforce Business Rules

DELIMITER $$

CREATE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

DECLARE current\_balance DOUBLE;

-- Get the current account balance

SELECT balance INTO current\_balance

FROM Accounts

WHERE account\_id = NEW.account\_id;

-- Rule 1: Deposits must be positive

IF NEW.txn\_type = 'DEPOSIT' AND NEW.amount <= 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Deposit amount must be greater than 0';

END IF;

-- Rule 2: Withdrawals must not exceed balance

IF NEW.txn\_type = 'WITHDRAWAL' AND NEW.amount > current\_balance THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient funds for withdrawal';

END IF;

-- Rule 3 (Optional): Withdrawal must be positive too

IF NEW.txn\_type = 'WITHDRAWAL' AND NEW.amount <= 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Withdrawal amount must be greater than 0';

END IF;

END$$

DELIMITER ;

Usage Example

-- Insert account with some balance

INSERT INTO Accounts (account\_id, balance) VALUES ('ACC1001', 5000.0);

-- Valid deposit

INSERT INTO Transactions (account\_id, amount, txn\_type, txn\_date)

VALUES ('ACC1001', 2000.0, 'DEPOSIT', CURDATE());

-- Valid withdrawal

INSERT INTO Transactions (account\_id, amount, txn\_type, txn\_date)

VALUES ('ACC1001', 1000.0, 'WITHDRAWAL', CURDATE());

Invalid: Deposit with negative amount

-- Triggers error: "Deposit amount must be greater than 0"

INSERT INTO Transactions (account\_id, amount, txn\_type, txn\_date)

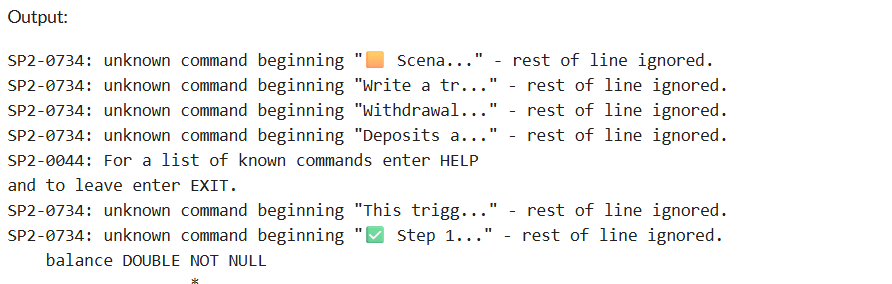
VALUES ('ACC1001', -500.0, 'DEPOSIT', CURDATE());

Invalid: Withdrawal exceeds balance

-- Triggers error: "Insufficient funds for withdrawal"

INSERT INTO Transactions (account\_id, amount, txn\_type, txn\_date)

VALUES ('ACC1001', 10000.0, 'WITHDRAWAL', CURDATE());



Excerise-6:

BEGIN

FOR obj IN (

SELECT object\_name, object\_type

FROM user\_objects

WHERE object\_name IN (

'CUSTOMERS', 'TRANSACTIONS', 'ACCOUNTS', 'LOANS'

)

) LOOP

BEGIN

EXECUTE IMMEDIATE 'DROP ' || obj.object\_type || ' "' || obj.object\_name || '" CASCADE';

DBMS\_OUTPUT.PUT\_LINE('Dropped ' || obj.object\_type || ': ' || obj.object\_name);

EXCEPTION WHEN OTHERS THEN NULL;

END;

END LOOP;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100)

);

/

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

CustomerID NUMBER,

TranDate DATE,

Amount NUMBER(10,2),

Description VARCHAR2(200)

);

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER(10,2)

);

/

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER(5,2)

);

/

INSERT INTO Customers VALUES (1, 'Alice');

INSERT INTO Customers VALUES (2, 'Bob');

INSERT INTO Customers VALUES (3, 'Charlie');

INSERT INTO Transactions VALUES (1001, 1, SYSDATE, 1000, 'Deposit');

INSERT INTO Transactions VALUES (1002, 2, SYSDATE-1, 1500, 'Withdrawal');

INSERT INTO Transactions VALUES (1003, 1, ADD\_MONTHS(SYSDATE, -1), 2000, 'Old Transaction');

INSERT INTO Accounts VALUES (101, 1, 8000);

INSERT INTO Accounts VALUES (102, 2, 3000);

INSERT INTO Accounts VALUES (103, 3, 6000);

INSERT INTO Loans VALUES (201, 1, 7.5);

INSERT INTO Loans VALUES (202, 2, 8.2);

INSERT INTO Loans VALUES (203, 3, 6.9);

COMMIT;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('📄 Monthly Transactions:');

FOR txn\_rec IN (

SELECT c.Name, t.TransactionID, t.TranDate, t.Amount, t.Description

FROM Transactions t

JOIN Customers c ON c.CustomerID = t.CustomerID

WHERE TO\_CHAR(t.TranDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY')

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Customer: ' || txn\_rec.Name ||

' | TranID: ' || txn\_rec.TransactionID ||

' | Date: ' || TO\_CHAR(txn\_rec.TranDate, 'DD-MON') ||

' | ₹' || txn\_rec.Amount ||

' | ' || txn\_rec.Description

);

END LOOP;

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '💸 Annual Maintenance Fee Applied:');

FOR acc\_rec IN (SELECT AccountID, Balance FROM Accounts) LOOP

UPDATE Accounts

SET Balance = Balance - 500

WHERE AccountID = acc\_rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE(

'Account: ' || acc\_rec.AccountID ||

' | Old Balance: ₹' || acc\_rec.Balance ||

' | New Balance: ₹' || (acc\_rec.Balance - 500)

);

END LOOP;

COMMIT;

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '📈 Loan Interest Rate Updates:');

FOR loan\_rec IN (SELECT LoanID, InterestRate FROM Loans) LOOP

IF loan\_rec.InterestRate < 8 THEN

UPDATE Loans

SET InterestRate = loan\_rec.InterestRate + 1

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

'Loan ' || loan\_rec.LoanID || ': Increased to ' || (loan\_rec.InterestRate + 1) || '%'

);

ELSE

UPDATE Loans

SET InterestRate = loan\_rec.InterestRate - 0.5

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

'Loan ' || loan\_rec.LoanID || ': Reduced to ' || (loan\_rec.InterestRate - 0.5) || '%'

);

END IF;

END LOOP;

COMMIT;

END;

/

PROMPT FINAL ACCOUNTS

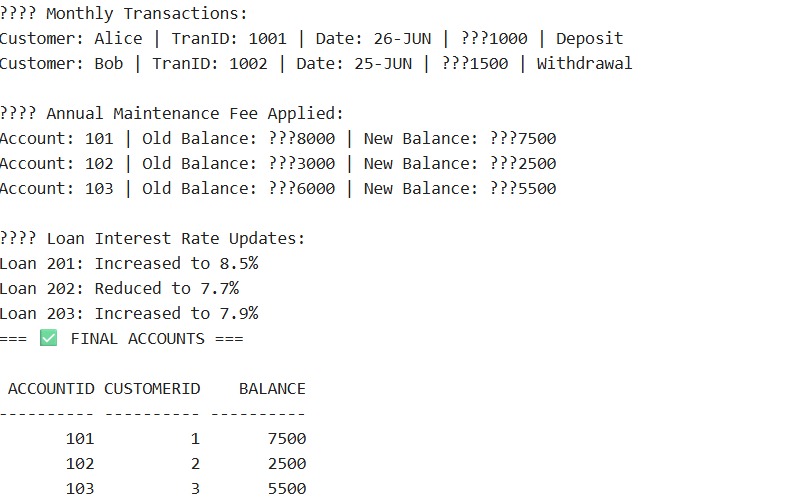
SELECT \* FROM Accounts;

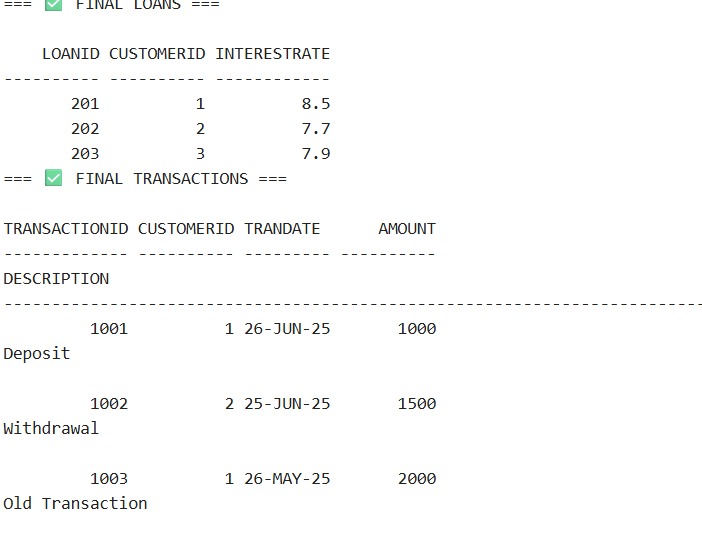
PROMPT FINAL LOANS

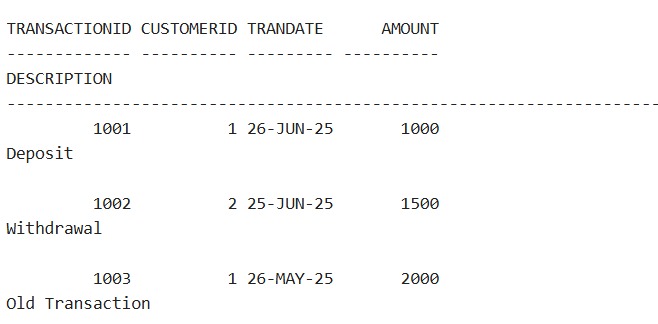
SELECT \* FROM Loans;

PROMPT FINAL TRANSACTIONS

SELECT \* FROM Transactions;







Excerise-7:

-- DROP TABLES IF EXIST

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- CREATE TABLES

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 SAMPLE DATA

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

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

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

-- CREATE CUSTOMER PACKAGE

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE, balance IN NUMBER);

PROCEDURE UpdateCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE);

FUNCTION GetCustomerBalance(custID IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE, balance IN NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (custID, name, dob, balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer already exists');

END;

PROCEDURE UpdateCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE) IS

BEGIN

UPDATE Customers SET Name = name, DOB = dob, LastModified = SYSDATE

WHERE CustomerID = custID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Customer not found');

ELSE

COMMIT;

END IF;

END;

FUNCTION GetCustomerBalance(custID IN NUMBER) RETURN NUMBER IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Customers WHERE CustomerID = custID;

RETURN bal;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END CustomerManagement;

/

-- CREATE EMPLOYEE PACKAGE

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(empID IN NUMBER, name IN VARCHAR2, position IN VARCHAR2,

salary IN NUMBER, dept IN VARCHAR2, hireDate IN DATE);

PROCEDURE UpdateEmployee(empID IN NUMBER, position IN VARCHAR2, salary IN NUMBER);

FUNCTION AnnualSalary(empID IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(empID IN NUMBER, name IN VARCHAR2, position IN VARCHAR2,

salary IN NUMBER, dept IN VARCHAR2, hireDate IN DATE) IS

BEGIN

INSERT INTO Employees VALUES (empID, name, position, salary, dept, hireDate);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Employee already exists');

END;

PROCEDURE UpdateEmployee(empID IN NUMBER, position IN VARCHAR2, salary IN NUMBER) IS

BEGIN

UPDATE Employees SET Position = position, Salary = salary WHERE EmployeeID = empID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee not found');

ELSE

COMMIT;

END IF;

END;

FUNCTION AnnualSalary(empID IN NUMBER) RETURN NUMBER IS

monthly NUMBER;

BEGIN

SELECT Salary INTO monthly FROM Employees WHERE EmployeeID = empID;

RETURN monthly \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END EmployeeManagement;

/

-- CREATE ACCOUNT PACKAGE

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(accID IN NUMBER, custID IN NUMBER, accType IN VARCHAR2, balance IN NUMBER);

PROCEDURE CloseAccount(accID IN NUMBER);

FUNCTION GetTotalBalance(custID IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(accID IN NUMBER, custID IN NUMBER, accType IN VARCHAR2, balance IN NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (accID, custID, accType, balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Account already exists');

END;

PROCEDURE CloseAccount(accID IN NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = accID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Account not found');

ELSE

COMMIT;

END IF;

END;

FUNCTION GetTotalBalance(custID IN NUMBER) RETURN NUMBER IS

total NUMBER;

BEGIN

SELECT SUM(Balance) INTO total FROM Accounts WHERE CustomerID = custID;

RETURN total;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END AccountOperations;

/

-- TEST BLOCK

BEGIN

CustomerManagement.AddCustomer(3, 'Ravi', TO\_DATE('1992-11-11', 'YYYY-MM-DD'), 2500);

EmployeeManagement.HireEmployee(3, 'Charlie', 'Analyst', 50000, 'Finance', SYSDATE);

AccountOperations.OpenAccount(3, 3, 'Savings', 2500);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || CustomerManagement.GetCustomerBalance(3));

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.AnnualSalary(3));

DBMS\_OUTPUT.PUT\_LINE('Total Account Balance: ' || AccountOperations.GetTotalBalance(3));

END;

/

