**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.

-- Create or replace the UpdateCustomerLastModified trigger

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

-- Set the LastModified column to the current date and time

:NEW.LastModified := SYSDATE;

END;

/

-- Update a customer's record

UPDATE Customers

SET Name = 'John Doe Jr.'

WHERE CustomerID = 1;

-- Check the LastModified column to verify the trigger's effect

SELECT CustomerID, Name, LastModified

FROM Customers

WHERE CustomerID = 1;

**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.

-- Create the AuditLog table

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE

);

-- Create or replace the LogTransaction trigger

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

-- Insert a record into the AuditLog table

INSERT INTO AuditLog (

TransactionID,

TransactionDate,

Amount,

TransactionType,

LogDate

)

VALUES (

:NEW.TransactionID,

:NEW.TransactionDate,

:NEW.Amount,

:NEW.TransactionType,

SYSDATE -- Current date and time

);

END;

/

-- Insert a new transaction

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

VALUES (1, 1, SYSDATE, 1000, 'Deposit');

-- Check the AuditLog table to verify the log entry

SELECT \* FROM AuditLog

WHERE TransactionID = 1;

**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.

-- Create or replace the CheckTransactionRules trigger

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_CurrentBalance NUMBER;

BEGIN

-- Fetch the current balance of the account

SELECT Balance INTO v\_CurrentBalance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

-- Check if the transaction is a withdrawal

IF :NEW.TransactionType = 'Withdrawal' THEN

-- Ensure withdrawals do not exceed the available balance

IF :NEW.Amount > v\_CurrentBalance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal amount exceeds available balance.');

END IF;

-- Check if the transaction is a deposit

ELSIF :NEW.TransactionType = 'Deposit' THEN

-- Ensure deposits are positive

IF :NEW.Amount <= 0 THEN

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

END IF;

ELSE

-- Handle unknown transaction types

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

END IF;

END;

/

-- Attempt to insert a withdrawal that exceeds the balance

BEGIN

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

VALUES (1, 1, SYSDATE, 2000, 'Withdrawal');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

-- Attempt to insert a deposit with a negative amount

BEGIN

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

VALUES (2, 1, SYSDATE, -100, 'Deposit');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

-- Attempt to insert a valid deposit

BEGIN

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

VALUES (3, 1, SYSDATE, 500, 'Deposit');

END;

/

-- Check the Transactions table to verify successful insertion

SELECT \* FROM Transactions;