**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.

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

DOB DATE,

Balance DECIMAL(10, 2),

LastModified TIMESTAMP

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR(20),

Balance DECIMAL(10, 2),

LastModified TIMESTAMP,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate TIMESTAMP,

Amount DECIMAL(10, 2),

TransactionType VARCHAR(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount DECIMAL(10, 2),

InterestRate DECIMAL(5, 2),

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Position VARCHAR(50),

Salary DECIMAL(10, 2),

Department VARCHAR(50),

HireDate DATE

);

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

VALUES (1, 'John Doe', '1985-05-15', 1000, NOW());

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

VALUES (2, 'Jane Smith', '1990-07-20', 1500, NOW());

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, NOW());

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, NOW());

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

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

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

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

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, NOW(), DATE\_ADD(NOW(), INTERVAL 5 YEAR));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', '2015-06-15');

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', '2017-03-20');

DELIMITER //

CREATE PROCEDURE SafeTransferFunds(

IN fromAccountID INT,

IN toAccountID INT,

IN transferAmount DECIMAL(10, 2)

)

BEGIN

DECLARE insufficientFunds BOOL DEFAULT FALSE;

DECLARE accountNotFound BOOL DEFAULT FALSE;

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

-- Rollback transaction and log the error

ROLLBACK;

SELECT 'An error occurred during the transfer. Transaction rolled back.' AS ErrorMessage;

END;

START TRANSACTION;

-- Check if both accounts exist

IF (SELECT COUNT(\*) FROM Accounts WHERE AccountID = fromAccountID) = 0 THEN

SET accountNotFound = TRUE;

END IF;

IF (SELECT COUNT(\*) FROM Accounts WHERE AccountID = toAccountID) = 0 THEN

SET accountNotFound = TRUE;

END IF;

IF accountNotFound THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'One or both accounts do not exist';

END IF;

-- Check if the fromAccount has sufficient balance

IF (SELECT Balance FROM Accounts WHERE AccountID = fromAccountID) < transferAmount THEN

SET insufficientFunds = TRUE;

END IF;

IF insufficientFunds THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds in the from account';

END IF;

-- Perform the transfer

UPDATE Accounts SET Balance = Balance - transferAmount WHERE AccountID = fromAccountID;

UPDATE Accounts SET Balance = Balance + transferAmount WHERE AccountID = toAccountID;

COMMIT;

SELECT 'Transfer completed successfully' AS SuccessMessage;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE UpdateSalary(

IN empID INT,

IN salaryIncrease DECIMAL(10, 2)

)

BEGIN

DECLARE employeeNotFound BOOL DEFAULT FALSE;

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

-- Log error and rollback

ROLLBACK;

SELECT 'An error occurred while updating the salary. Transaction rolled back.' AS ErrorMessage;

END;

START TRANSACTION;

-- Check if employee exists

IF (SELECT COUNT(\*) FROM Employees WHERE EmployeeID = empID) = 0 THEN

SET employeeNotFound = TRUE;

END IF;

IF employeeNotFound THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Employee ID does not exist';

END IF;

-- Update the salary

UPDATE Employees SET Salary = Salary + salaryIncrease WHERE EmployeeID = empID;

COMMIT;

SELECT 'Salary updated successfully' AS SuccessMessage;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE AddNewCustomer(

IN newCustomerID INT,

IN newName VARCHAR(100),

IN newDOB DATE,

IN newBalance DECIMAL(10, 2)

)

BEGIN

DECLARE customerExists BOOL DEFAULT FALSE;

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

-- Log error and rollback

ROLLBACK;

SELECT 'An error occurred while adding the new customer. Transaction rolled back.' AS ErrorMessage;

END;

START TRANSACTION;

-- Check if customer already exists

IF (SELECT COUNT(\*) FROM Customers WHERE CustomerID = newCustomerID) > 0 THEN

SET customerExists = TRUE;

END IF;

IF customerExists THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Customer with this ID already exists';

END IF;

-- Insert new customer

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

VALUES (newCustomerID, newName, newDOB, newBalance, CURRENT\_TIMESTAMP);

COMMIT;

SELECT 'New customer added successfully' AS SuccessMessage;

END //

DELIMITER ;

SELECT COUNT(\*) FROM Employees WHERE EmployeeID = 1;

UPDATE Employees SET Salary = Salary + 1000 WHERE EmployeeID = 1;

SELECT \* FROM Employees WHERE EmployeeID = 1;

SELECT COUNT(\*) FROM Customers WHERE CustomerID = 3;

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

VALUES (3, 'New Customer', '2000-01-01', 5000, CURRENT\_TIMESTAMP);

SELECT \* FROM Customers WHERE CustomerID = 3;

-- Call UpdateSalary procedure

CALL UpdateSalary(1, 1000);

-- Call AddNewCustomer procedure

CALL AddNewCustomer(3, 'New Customer', '2000-01-01', 5000);

-- Check updated salary

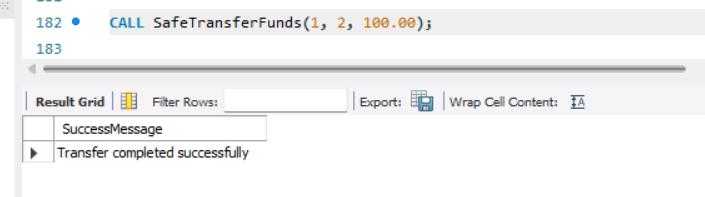
SELECT \* FROM Employees WHERE EmployeeID = 1;

-- Check new customer insertion

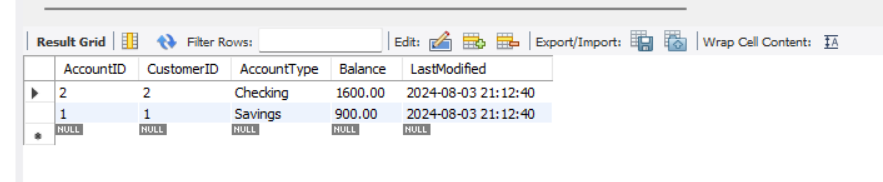
SELECT \* FROM Customers WHERE CustomerID = 3;

CALL UpdateSalary(1, 1000);

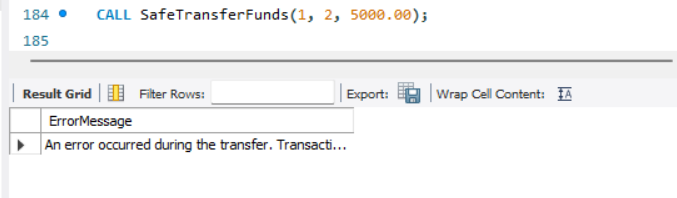
CALL AddNewCustomer(3, 'New Customer', '2000-01-01', 5000);

CALL SafeTransferFunds(1, 2, 100.00);

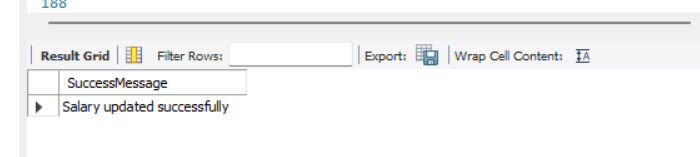
SELECT \* FROM Accounts;



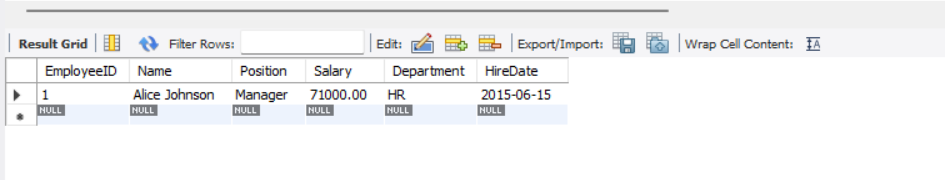
CALL SafeTransferFunds(1, 2, 5000.00);



CALL UpdateSalary(1, 1000.00);



SELECT \* FROM Employees WHERE EmployeeID = 1;



CALL UpdateSalary(999, 1000.00);

