Group Project Phase 2

Robert Parker, Arianne Guedes, Jorge Gayer

-- Step 1. Implementing different functionalities using stored procedure and user defined function.

-- How many unpaid loans each branch has at the moment (LoanID by Branch)

USE SKS\_BANK;

CREATE PROCEDURE spUnpaidLoansByBranch

( @BranchID AS int = 0)

AS

BEGIN

if @BranchID = 0

SELECT

C.BranchID,

b.Name,

COUNT(LC.LoanID) NumberOfLoans,

SUM(L.Balance) AS TotalUnpaidPerBranch

FROM

Customers C LEFT JOIN

LoanCustomers LC

ON

C.CustomerID = LC.CustomerID

LEFT JOIN Loans L

ON

L.LoanID = LC.LoanID

left join Branches b

ON

b.BranchID = c.BranchID

WHERE Balance > 0

GROUP BY C.BranchID, b.Name

else

SELECT

C.BranchID,

b.Name,

COUNT(LC.LoanID) NumberOfLoans,

SUM(L.Balance) AS TotalUnpaidPerBranch

FROM

Customers C LEFT JOIN

LoanCustomers LC

ON

C.CustomerID = LC.CustomerID

LEFT JOIN Loans L

ON

L.LoanID = LC.LoanID

left join Branches b

ON

b.BranchID = c.BranchID

WHERE Balance > 0

and c.branchid = @BranchID

GROUP BY C.BranchID, b.Name

END;

exec spUnpaidLoansByBranch

exec spUnpaidLoansByBranch 1

CREATE PROC spAccountsByBankerID (@BankerID AS INT = 0)

AS

BEGIN

IF @BankerID = 0

THROW 50001, 'Please enter the Banker ID', 1;

ELSE

-- Account numbers by banker

SELECT C.BankerID, C.name, AC.AccountID AS AccountNumber

FROM

Customers C LEFT JOIN Employees ON C.BankerID = Employees.EmployeeID

LEFT JOIN AccountCustomers AC ON AC.CustomerID = C.CustomerID

WHERE

BankerID = @BankerID

order by name asc

end

EXEC spAccountsByBankerID

EXEC spAccountsByBankerID 8

--User Defined function

CREATE FUNCTION getPaymentBranchAmountByDate (

@BranchID AS Int, @PaymentDate AS DATE)

RETURNS Money

AS begin

RETURN (

select

sum(p.amount) as Total

from payments p

left join loans l on p.loanid = L.LOANID

left join loancustomers lc on lc.loanid = l.loanid

left join customers c on c.customerid = lc.customerid

left join branches b on c.branchid = b.branchid

where c.branchid = @BranchID and p.date = @PaymentDate

group by b.name, p.date

)

end;

select dbo.getPaymentBranchAmountByDate (1, '2021-12-16')

--Step 2. Create different set of triggers (minimum 2 numbers) to monitor the different DML and DDL activates in the database

-- Trigger to upload the balance on Loans table once you do a payment on Payments table

CREATE TRIGGER tgUpdateLoanBalance

ON Payments

after INSERT

AS

BEGIN

declare @Value as money;

declare @LoanID as int;

select @Value = amount from inserted;

select @LoanID = LoanID from inserted;

update loans set balance = balance - @value where loanID = @LoanID

END;

select \* from loans

select \* from payments

select \* from loancustomers

select \* from customers

INSERT INTO payments (Loanid,amount, date) values (1, 1000, getdate())

-- Creating trigger preventing to drop or alter a table

--ENABLE TRIGGER safety on database

CREATE TRIGGER

safety

ON DATABASE

FOR DROP\_TABLE, ALTER\_TABLE

AS

PRINT 'Contact your Database admin to disable the safety trigger!'

ROLLBACK;

ALTER TABLE Accounts

ADD FOREIGN KEY (AccountTypeID) REFERENCES AccountTypes(AccountTypeID);

--Step 3 Create index based on frequently used attribute for three of any table

-- Composite Clustered index

--a

ALTER TABLE Employees

DROP CONSTRAINT FK\_\_Employees\_\_Offic\_\_71D1E811

--b

--Delete the Office default index through GUI

Graphical user interface, text, application

Description automatically generated

CREATE Clustered INDEX cuix\_Offices

ON Offices (Name, CityID);

--c

ALTER TABLE Offices

ADD Primary KEY (OfficeID)

--d

ALTER TABLE Employees

ADD FOREIGN KEY (OfficeID) REFERENCES Offices(OfficeID);

-- Replace the default Clustered index with non key attribute

--a

ALTER TABLE Accounts

DROP CONSTRAINT FK\_\_Accounts\_\_Accoun\_\_46E78A0C

--b

--Delete the AccountTypes default index through GUI

Graphical user interface, text, application

Description automatically generated

CREATE Clustered INDEX cuix\_AccountTypes

ON AccountTypes (Description);

--c

ALTER TABLE AccountTypes

ADD Primary KEY (AccountTypeID)

--d

ALTER TABLE Accounts

ADD FOREIGN KEY (AccountTypeID) REFERENCES AccountTypes(AccountTypeID);

--non clustered Composite index

CREATE INDEX nix\_Account

ON Accounts (BranchID, AccountTypeid);

--Step 4. Create different level of users and assign appropriate privilege.

create login

customer\_1

with

password = 'customer'

create user

customer\_1

create role Customers

alter role customers add member customer\_1

-- Grant update and select access to customers for accounts table

Grant SELECT, UPDATE

ON ACCOUNTS

TO CUSTOMERS

-- Grant update and select access to customers for Customers table

Grant SELECT, UPDATE

ON Customers

TO CUSTOMERS

-- Grant update and select access to customers for Loans table

Grant SELECT, UPDATE

ON Loans

TO CUSTOMERS

-- Grant update and select access to customers for Payments table

Grant SELECT, UPDATE

ON Payments

TO CUSTOMERS

-- Accountant user

create login

accountant\_1

with

password = 'accountant'

create user

accountant\_1

create role Accountant

alter role Accountant add member Accountant\_1

Grant SELECT, update, delete ON AccountCustomers TO Accountant

Grant SELECT, update, delete ON AccountInterest TO Accountant

Grant SELECT, update, delete ON AccountOverdrafts TO Accountant

Grant SELECT ON Accounts TO Accountant

Grant SELECT, update, delete ON AccountTypes TO Accountant

Grant SELECT, update, delete ON Branches TO Accountant

Grant SELECT, update, delete ON Cities TO Accountant

Grant SELECT, update, delete ON Customers TO Accountant

Grant SELECT, update, delete ON Employees TO Accountant

Grant SELECT, update, delete ON EmployeeTypes TO Accountant

Grant SELECT, update, delete ON LoanCustomers TO Accountant

Grant SELECT ON Loans TO Accountant

Grant SELECT, update, delete ON Offices TO Accountant

Grant SELECT ON Payments TO Accountant

-- Prove of access to customer

--Customers should not have access to the queries below

delete from loans where loanid = 5;

create role Accountant

select \* from cities

-- Proof of access to accountant

--This should work with granted access

update cities set name = 'teamwork' where cityid= 1

update cities set name = 'Vancouver' where cityid= 1

select \* from cities

--Access to this should not be allowed

INSERT INTO payments (Loanid,amount, date) values (1, 1000, getdate())

-- Step 5: Recovery Model and Backup

alter database SKS\_Bank set recovery full with no\_wait;

backup database SKS\_BANK TO DISK = 'D:\Software Development\DATA2201\FullSKS.bak'