CSIS3714 FINAL-YEAR PROJECT

UBT eTolls

CSIS3714

MAY 19, 2017

MOLORANE MOTHUSI MICHAEL
2014098616

Contents

Maintain (add,edit) qualifications using one stored procedure

```
CREATE PROCEDURE spAddEditQualifications
@DegreeID int = -1,
@DegreeName NVarchar(100),
@DegreeDesc NVarchar(100)
AS
BEGIN
        BEGIN TRY
                BEGIN TRANSACTION
                        IF EXISTS(SELECT DegreeID FROM Degree WHERE DegreeID = @DegreeID)
                                        BEGIN
                                                UPDATE Degree SET
                                                        DegreeName = @DegreeName,
                                                        DegreeDesc = @DegreeDesc
                                                        WHERE DegreeID = @DegreeID
                                        END
                                ELSE
                                        BEGIN
                                                INSERT INTO Degree(DegreeName,DegreeDesc)
                                                        VALUES(@DegreeName,@DegreeDesc)
                                        END
                COMMIT TRANSACTION
        END TRY
        BEGIN CATCH
                ROLLBACK
        END CATCH
END
```

Maintain (add,edit and delete) benefits stored procedure

```
CREATE PROCEDURE spAddEditQualifications
@DegreeID int = -1,
@DegreeName NVarchar(100),
@DegreeDesc NVarchar(100)
AS
BEGIN
        BEGIN TRY
                BEGIN TRANSACTION
                        IF EXISTS(SELECT DegreeID FROM Degree WHERE DegreeID = @DegreeID)
                                        BEGIN
                                                UPDATE Degree SET
                                                        DegreeName = @DegreeName,
                                                        DegreeDesc = @DegreeDesc
                                                        WHERE DegreeID = @DegreeID
                                        END
                                ELSE
                                        BEGIN
                                                INSERT INTO Degree(DegreeName,DegreeDesc)
                                                        VALUES(@DegreeName,@DegreeDesc)
                                        END

                COMMIT TRANSACTION
        END TRY
        BEGIN CATCH
                ROLLBACK
        END CATCH
END
```

```
CREATE PROCEDURE spDeleteBenefit
@BenefitID int
AS
BEGIN
        BEGIN TRY
                BEGIN TRANSACTION
                                DELETE FROM Benefit WHERE BenefitID = @BenefitID
                COMMIT TRANSACTION
        END TRY
        BEGIN CATCH
                ROLLBACK --Undo changes
        END CATCH
END
```

Maintain (add,edit) gantries stored procedure

```
CREATE PROCEDURE spAddEditGantries
@GantryID int = -1,
@Gantry_Name NVarchar(100),
@Gantry_Desc NVarchar(100),
@Gantry_GPSLocation NVarchar(100),
@ROfficeID INT,
@TRateID INT
AS
BEGIN
        BEGIN TRY
                BEGIN TRANSACTION
                        IF EXISTS(SELECT GantryID FROM Gantry WHERE GantryID = @GantryID)
                                        BEGIN
                                                UPDATE Gantry SET
                                                        Gantry_Name = @Gantry_Name,
                                                        Gantry_Desc = @Gantry_Desc,
                                                        Gantry_GPSLocation =
@Gantry_GPSLocation,
                                                        ROfficeID = @ROfficeID,
                                                        TRateID = @TRateID
                                                        WHERE GantryID = @GantryID
                                        END
                                ELSE
                                        BEGIN
                                                INSERT INTO
Gantry(Gantry_Name,Gantry_Desc,Gantry_GPSLocation,ROfficeID,TRateID)

        VALUES(@Gantry_Name,@Gantry_Desc,@Gantry_GPSLocation,@ROfficeID,@TRateID)
                                        END

                COMMIT TRANSACTION
        END TRY
```

```
       BEGIN CATCH
              ROLLBACK
       END CATCH
END
```

Maintain(add) a vehicle fine stored procedure

```
CREATE PROCEDURE spAddVehicleFine
@VehicleID int,
@FineID INT = -1
AS
BEGIN
       BEGIN TRY
              BEGIN TRANSACTION
                     --IF the Fine is not specified (FineID = -1),
                     --Check the default fine from Fines Entity
                            IF(@FineID = -1)
                                   BEGIN
                                          DECLARE @DefaultFineID INT

                                          SELECT @DefaultFineID = FineID FROM Fines
WHERE Fine_IsDefault = 1

                                          IF(@@ROWCOUNT >= 1)
                                          BEGIN
-- THE FOLLOWING FIELDS ARE NOT SPECIFIED BECAUSE
-- Vehicle_FineId -> is an Identity Key
-- VF_IsPaid -> has a default value 0 at fine creation, meaning fine not paid yet
-- VF_DateTime -> configured to take CurrentTimeStamp from SQL Server

                                          INSERT INTO VehicleFine(VehicleID,FineID)
                                                 VALUES(@VehicleID,@DefaultFineID)
                                          END
                                   END
                            ELSE
                                   BEGIN
                                          INSERT INTO VehicleFine(VehicleID,FineID)
                                                        VALUES(@VehicleID,@FineID)
                                   END
              COMMIT TRANSACTION
       END TRY
       BEGIN CATCH
              ROLLBACK --Undo changes
       END CATCH
END
```

Create (add) Toll transaction stored procedure

```
ALTER PROCEDURE spAddTollTransaction
@TT_Amount DECIMAL(18,2),
@TT_VehicleRegistration NVarchar(20) = NULL,
@TTagID INT,
@DRateID INT,
@GantryID INT,
@FineID INT  = -1, --When FineID is not specified set it to -1

--Payment Details if payment is made(@@isPaymentMade = 1)
@isPaymentMade BIT = 1, -- Assume payment is also made
```

```
@PaymentMethodID INT = -1, --Assume no paymentmethod
@ActualAmountPaid DECIMAL(18,2) = 0.00
AS
BEGIN
       BEGIN TRY
              BEGIN TRANSACTION

                     DECLARE @TTOnDiscount INT = 0;  --Assume No dicount rate
                     DECLARE @TagIDDetected INT = -1; --Assume No tag was detected
                     DECLARE @NewTTID INT
                     DECLARE @NewPayID INT

                     --Check if TransactionDate Falls within Discount Rate
                     Select DRateID FROM DiscountRate
                                   WHERE DRateID = @DRateID AND
                                   GETDATE() BETWEEN DRate_StartDateTime AND
DRate_EndDateTime

                     IF(@@ROWCOUNT >= 1)
                                   SET @TTOnDiscount = 1;

                     --Check IF Tag is detected(IF TTagID exists in TollTag Entity)
                     IF(EXISTS(SELECT TTagID FROM TollTag WHERE TTagID = @TTagID))
                            SET @TagIDDetected = @TTagID

                     -- THE FOLLOWING FIELDS ARE NOT SPECIFIED BECAUSE
                     -- TTID -> is an Identity Key
                     -- TT_DateTime -> configured to take CurrentTimeStamp from SQL
Server
                     -- TT_IsPaid -> has a default value 0 at transaction creation,
meaning transaction not paid yet
                     IF(@TagIDDetected = -1)
                            INSERT INTO
TollTransaction(TT_Amount,TT_OnDiscount,TT_VehicleRegistration,TTagID,PayID,DRateID,Ga
ntryID)

       VALUES(@TT_Amount,@TTOnDiscount,@TT_VehicleRegistration,NULL,NULL,@DRateID,@Gan
tryID);
                     ELSE
                            INSERT INTO
TollTransaction(TT_Amount,TT_OnDiscount,TT_VehicleRegistration,TTagID,PayID,DRateID,Ga
ntryID)

       VALUES(@TT_Amount,@TTOnDiscount,@TT_VehicleRegistration,@TagIDDetected,NULL,@DR
ateID,@GantryID);

                     SELECT @NewTTID = SCOPE_IDENTITY()

                     --If payment is also made
                     --If yes meaning (@@isPaymentMade = 1)
                     IF(@isPaymentMade = 1)
                     BEGIN
                            INSERT INTO Payment(Pay_Amount,PayMethodID)
VALUES(@ActualAmountPaid,@PaymentMethodID)

                            SELECT @NewPayID = SCOPE_IDENTITY()

                            UPDATE TollTransaction SET TT_IsPaid = 1,PayID = @NewPayID
WHERE TTID = @NewTTID
                     END

                     --IF No tag was detected issue a fine
                     --Given VehicleRegistration Number, Find VehicleID that has the
given registration number
                     --Call a stored procedure (spAddVehicleFine) to issue a fine
                     IF(@TagIDDetected = -1)
                     BEGIN
                            DECLARE @VehicleID INT;

                            SELECT @VehicleID = VehicleID FROM VehicleRegistration
                                                        WHERE Vehicle_Registration =
@TT_VehicleRegistration

                            IF(@@ROWCOUNT >= 1)
                                   EXEC spAddVehicleFine @VehicleID,@FineID
                            ELSE
                                   ROLLBACK
                     END
              COMMIT TRANSACTION
       END TRY
       BEGIN CATCH
              ROLLBACK --Undo changes
              SELECT ERROR_MESSAGE(),ERROR_LINE(),ERROR_PROCEDURE(),ERROR_NUMBER()
       END CATCH
END
```

Maintain (add,edit) Vehicle registration stored procedure

```
CREATE PROCEDURE spAddEditVehicleRegistration
@VehicleID INT,
@Vehicle_Name NVarchar(100),
@Vehicle_Desc NVarchar(100),
@Vehicle_Registration NVarchar(20),
@CustomerID INT,
@MunID INT
AS
BEGIN
       BEGIN TRY
              BEGIN TRANSACTION
                            IF EXISTS(SELECT VehicleID FROM VehicleRegistration WHERE
VehicleID = @VehicleID)
                                   BEGIN

                                          DECLARE @PreviousCustmerID NVarchar(20)

                                          --Get Old CustomerID
                                          SELECT @PreviousCustmerID = CustomerID
                                                        FROM VehicleRegistration
                                                     WHERE VehicleID = @VehicleID

                                          --Check if vehicle is not changing from one
owner to another
                                          --IF the vehicle is not changing
ownership(Previous CustomerID = Current given @CustomerID)
                                          --Update vehicle details
                                          IF( @PreviousCustmerID = @CustomerID )
                                                 BEGIN
                                                        UPDATE VehicleRegistration
                                                               SET Vehicle_Name =
@Vehicle_Name,Vehicle_Desc = @Vehicle_Desc, Vehicle_Registration =
@Vehicle_Registration,MunID = @MunID
                                                                      WHERE VehicleID =
@VehicleID AND CustomerID = @CustomerID
```

```
                                                    END
                                             ELSE
                                                    BEGIN
                                                           --IF the vehicle is changing
ownership(Previous CustomerID <> Current given @CustomerID)
                                                           --Then, we can have two OPTIONS
                                                           -- 1. Update the CustomerID to
the New owner, thereby Inheriting the fines of the previous owner
                                                           -- 2. Or, Insert a new vehicle
registration thereby not inheriting the fines

                                                           -- OPTION 1
                                                           UPDATE VehicleRegistration
                                                                  SET Vehicle_Name =
@Vehicle_Name,Vehicle_Desc = @Vehicle_Desc,MunID = @MunID,CustomerID = @CustomerID
                                                                         WHERE
Vehicle_Registration = @Vehicle_Registration


                                                           -- OPTION 2
                                                           --Inserting new record INTO
Vehicle Registration will
                                                           --     *Generate new Vehicle ID
for new owner
                                                           --     *With new CustomerID
                                                           --     *But VehicleRegistration
no is for the previous owner
                                                           --     *This will prevent new
owner from inheriting the fines of the previous owner
                                                           --         Because the new
Customer has new VehicleID which is a foreign Key in VehicleFines
                                                           INSERT INTO
VehicleRegistration(Vehicle_Name,Vehicle_Desc,Vehicle_Registration,CustomerID,MunID)

       VALUES(@Vehicle_Name,@Vehicle_Desc,@Vehicle_Registration,@CustomerID,@MunID)


                                                    END

                                      END
                               ELSE
                               BEGIN
                                      INSERT INTO
VehicleRegistration(Vehicle_Name,Vehicle_Desc,Vehicle_Registration,CustomerID,MunID)

       VALUES(@Vehicle_Name,@Vehicle_Desc,@Vehicle_Registration,@CustomerID,@MunID)
                               END

              COMMIT TRANSACTION
       END TRY
       BEGIN CATCH
              ROLLBACK
       END CATCH
END
```

Register a new user **stored procedure** to assign a new user default role

```
CREATE PROCEDURE spAddUser
@UserName NVarchar(100),
@Password NVarchar(100)
AS
BEGIN
	BEGIN TRY
		BEGIN TRANSACTION
			DECLARE @HashedPassword NVarchar(100)
			DECLARE @NewUserID INT
			DECLARE @RoleId NVarchar(100)

			-- Get Default RoleId
			SELECT @RoleId = RoleId FROM UserRole WHERE IsDefault = 1

			SET @HashedPassword = sys.fn_varbintohexsubstring(0,
HashBytes('SHA1', @Password), 1, 0)
			--After INSERT the trigger called 'trAssignUserRole' will be
executed
			--To assign a user to a default Role
			INSERT INTO "User"(UserName,UserPassword,RoleId)
						VALUES(@UserName,@HashedPassword,@RoleId)

			SELECT @NewUserID = SCOPE_IDENTITY()

		COMMIT TRANSACTION
	END TRY
	BEGIN CATCH
		ROLLBACK
	END CATCH
END
```

--If I used a trigger to give the new user default role

```
ALTER TRIGGER trAssignUserRole
ON "User"
FOR INSERT
AS
BEGIN
	DECLARE @UserID INT

	DECLARE @RoleId NVarchar(100) --RoleId of the default Role

	SELECT @UserID = UserID FROM inserted

	-- Get Default RoleId
	SELECT @RoleId = RoleId FROM UserRole WHERE IsDefault = 1

	--Assign a new user default Role
	UPDATE "User" SET RoleId = @RoleId WHERE UserID = @UserID

END
```

Creating new staff record stored procedure

```
CREATE PROCEDURE spAddStaffRecord
@Staff_PNumber NVarchar(13),
@Staff_IDNumber NVarchar(13),
@Staff_FName NVarchar(100),
@Staff_LName NVarchar(100),
@Staff_Type NVarchar(20), -- Manager/Engineer/Support
@Staff_Income DECIMAL(18,2),
@ManagerID INT = null,
@ROfficeID INT = null,

@RoleID INT, -- RoleID of new staff if is(Manager/Support)

--Egineer Details if this staff is an engineer
@YearsOFExperince INT,
@DegreeID INT
AS
BEGIN
       BEGIN TRY
              BEGIN TRANSACTION

                     DECLARE @StaffID INT;

                     --IF ID Number is not valid
                     --Terminate the transaction and stop
                     IF(dbo.isValidIDNumber(@Staff_IDNumber) = 0)
                            RETURN

                     --Insert new Staff
                     INSERT INTO
Staff(Staff_PNumber,Staff_FName,Staff_LName,Staff_IDNumber,Staff_Type,Staff_Income,Man
agerID,ROfficeID)

VALUES(@Staff_PNumber,@Staff_IDNumber,@Staff_FName,@Staff_LName,@Staff_Type,@Staff_Inc
ome,@ManagerID,@ROfficeID)

                     --Get new StaffID
                     SELECT @StaffID = SCOPE_IDENTITY()

                     -- IF Staff is an Engineer
                     -- Record Engineer details
                     IF(@Staff_Type = 'Engineer')
                     BEGIN

                            INSERT INTO
Engineer(StaffID,Eng_YearsOfExperience,DegreeID)
                                      VALUES(@StaffID,@YearsOFExperince,@DegreeID)

                            IF(@@ROWCOUNT = 0)
                                   ROLLBACK; RETURN
                     END

                     -- IF Staff is a Manager
                     -- Record Manager details
                     IF(@Staff_Type = 'Manager')
                     BEGIN

                            --Get the number of staff managed by new staff
                            DECLARE @Man_StaffManaged INT;
                            SELECT @Man_StaffManaged = COUNT(StaffID) FROM Staff WHERE
ManagerID = @StaffID

                            --If inserting staff for the first time
                            INSERT INTO
Manager(StaffID,ManagerRoleId,Man_StaffManaged)
                                      VALUES(@StaffID,@RoleID,@Man_StaffManaged)

                            --If Staff already exist it could be updated
                            --As follows
                            UPDATE Manager SET Man_StaffManaged = @Man_StaffManaged
WHERE StaffID = @StaffID

                            --Check whether transaction was successful depending on
whether
                            --it was an insert(new staff) or an update of (existing
staff)
                            IF(@@ROWCOUNT = 0)
                                   ROLLBACK; RETURN
                     END

                     -- IF Staff is a Support
                     -- Record Support details
                     IF(@Staff_Type = 'Support')
                     BEGIN

                            INSERT INTO Support(StaffID,SupportRoleId)
                                      VALUES(@StaffID,@RoleID)

                            IF(@@ROWCOUNT = 0)
                                   ROLLBACK; RETURN
                     END

              COMMIT TRANSACTION
       END TRY
       BEGIN CATCH

              ROLLBACK;

              SELECT ERROR_MESSAGE(),
                            ERROR_LINE(),
                            ERROR_NUMBER(),
                            ERROR_PROCEDURE()
       END CATCH
END
```

ID Number Verification Function stored procedure

```
CREATE FUNCTION isValidIDNumber
(
       @StaffIDNumber NVarchar(13)
)
RETURNS BIT
AS
BEGIN
       Declare @DOB Varchar(6)
       DECLARE @LastSevenDigits Varchar(7)

       --Pattern to validate the last seven digits of an ID number
       DECLARE @Pattern NVarchar(35) = '[0-9][0-9][0-9][0-9][0-1][8-9][0-9]'

       --Get the first six digits of an ID number
       SET @DOB = LEFT(@StaffIDNumber,6)

       --Get the last seven digits of an ID number
       SET @LastSevenDigits = RIGHT(@StaffIDNumber,7)


       IF(ISDATE(@DOB) = 1 AND @LastSevenDigits LIKE @Pattern)
              RETURN 1
```

```
	RETURN 0;
END
```

SMS Daily Reminder Stored Procedure

```
CREATE PROCEDURE spSMSDailyReport
AS
BEGIN

	DECLARE @id NVarchar(13)
	DECLARE @Email NVarchar(100)
	DECLARE @FullNames NVarchar(100)
	DECLARE @CellNumber NVarchar(20)
	DECLARE @TTagID INT
	DECLARE @DaysBalance DECIMAL(18,2)
	DECLARE @TotalBalance DECIMAL(18,2)

	DECLARE @MessageBody NVarchar(1000);

	SELECT TOP 1 @id  =  c.CustomerID, @Email = c.Cus_Email,@FullNames = 
c.Cus_FName+' '+c.Cus_LName,@CellNumber = c.Cus_ContactNumber,
				@TTagID = tt.TTagID
				FROM Customer c
				JOIN TollTag tt
				ON c.CustomerID = tt.CustomerID
				JOIN TollTransaction tl
				ON tt.TTagID = tl.TTagID
				ORDER BY c.CustomerID

	WHILE @id IS NOT NULL
	BEGIN

	  SELECT @DaysBalance = (SUM(tt.TT_Amount) - SUM(p.Pay_Amount))
				   FROM TollTransaction tt
				   JOIN Payment p
				   ON tt.PayID = p.PayID
				   WHERE tt.TTagID = @TTagID
				   AND Convert(date,tt.TT_DateTime) = Convert(date,
getdate())
				   GROUP BY tt.TT_DateTime


	  SELECT @TotalBalance = (SUM(tt.TT_Amount) - SUM(p.Pay_Amount))
				   FROM TollTransaction tt
				   JOIN Payment p
				   ON tt.PayID = p.PayID
				   WHERE tt.TTagID = @TTagID
				   GROUP BY tt.TTagID

		--Check whether the aggregate values are NULL
		--IF NULL set to 0.00
		IF(@DaysBalance IS NULL)
			SET @DaysBalance = 0.00

		IF(@TotalBalance IS NULL)
			SET @TotalBalance = 0.00

		SET @MessageBody = @CellNumber + ' , '+@FullNames+' you have 
'+Convert(varchar(10),@DaysBalance)+
					' for '+ Convert(Varchar(13),Convert(date,
getdate()))+' and a total toll balance of '+Convert(varchar(10),@TotalBalance)
```

```
         ---SEND EMAIL TO CUSTOMER
              EXEC msdb.dbo.sp_send_dbmail
               @profile_name = 'SMSDailyReport',
               @recipients = @Email,
               @body = @MessageBody,
               @subject = 'Tolls Owing Reminder';
         ---END MAIL

         --IF we also want to return the list to calling application
         SELECT @CellNumber as CellNumber,@FullNames as FullNames,@DaysBalance as 
BalanceToday,@TotalBalance as TotalBalance

         SELECT  TOP 1 @id  =  c.CustomerID, @Email = c.Cus_Email,@FullNames = 
c.Cus_FName+' '+c.Cus_LName,@CellNumber = c.Cus_ContactNumber,
                            @TTagID = tt.TTagID
                            FROM Customer c
                            JOIN TollTag tt
                            ON c.CustomerID = tt.CustomerID
                            JOIN TollTransaction tl
                            ON tt.TTagID = tl.TTagID
                            WHERE c.CustomerID > @id
                            ORDER BY c.CustomerID
         IF @@ROWCOUNT = 0
         BREAK
       END
END
```

Function - Search a customer based on part of their surname

```
CREATE FUNCTION SearchCustomer
(
       @CustomerSurname NVarchar(100)
)
RETURNS TABLE
AS
       RETURN (SELECT * FROM Customer WHERE Cus_LName LIKE '%'+@CustomerSurname+'%')
```

Invalid ID Report Stored procedure

```
CREATE PROCEDURE spInvalidIDsReport
AS
BEGIN
       SELECT * FROM Customer WHERE dbo.isValidIDNumber(CustomerID) = 0
END
```

Non-compliant Report Stored procedure

```
CREATE PROCEDURE spNoneComplianceReport
AS
BEGIN
       SELECT c.Cus_FName as FirstName,c.Cus_LName as LastName,c.Cus_Email as 
Email,c.Cus_ContactNumber as ContactNumber
       FROM Customer c
       INNER JOIN VehicleRegistration vg
       ON c.CustomerID = vg.CustomerID
       INNER JOIN VehicleFine vf
```

```
        ON vg.VehicleID = vf.VehicleID
        LEFT JOIN TollTag tt
        ON c.CustomerID = tt.CustomerID
        WHERE tt.CustomerID IS NULL
        ORDER BY c.Cus_LName,c.Cus_FName
END
```

Popular Region Report stored procedure

```
CREATE PROCEDURE spPopularRegionReport
AS
BEGIN
        SELECT ro.ROffice_Name as RegionNane,COUNT(tt.TTID) as TotalTolls
        FROM RegionOffice ro
        JOIN Gantry g
        ON g.ROfficeID = ro.ROfficeID
        JOIN TollTransaction tt
        ON g.GantryID = tt.GantryID
        GROUP BY ro.ROffice_Name
END
```

Customer Account stored procedure

```
CREATE PROCEDURE spCustomerAccount
@CustomerID NVarchar(13)
AS
BEGIN

        DECLARE @VehicleReg NVarchar(30)
        DECLARE @DateTime DATETIME
        DECLARE @Charge DECIMAL(18,2)
        DECLARE @IsPaid varchar
        DECLARE @EAccount INT
        DECLARE @Balance DECIMAL(18,2)
        DECLARE @Cus_Area NVarchar(40)
        DECLARE @Cus_City NVarchar(40)
        DECLARE @Cus_StreetName NVarchar(60)
        DECLARE @Cus_StreetNumber NVarchar(40)
        DECLARE @Cus_FName NVarchar(40)
        DECLARE @Cus_LName NVarchar(40)
        DECLARE @PrintDate DATETIME
        DECLARE @EAccountFromSystem INT


        -- Assume the Dynamic temporary Table Called SystemVariable has already been
Created
        -- AND ITS LAST Value is 20993
        CREATE TABLE #SystemVariable(EAccount INT)

        -- Insert virtual value to represent last stored value
        INSERT INTO #SystemVariable VALUES(20993)

        --Extract the last stored value from the Temporary table SystemVariable
        --Store it in Variable @EAccount
        SELECT @EAccountFromSystem = EAccount FROM #SystemVariable

        CREATE TABLE #TollPayments(VReg Nvarchar(20),TT_DateTime DATETIME,Charge
DECIMAL(18,2),IsPaid varchar,
        Balance DECIMAL(18,2),Area nvarchar(40),City nvarchar(40),StreetName
NVarchar(40),StreetNumber NVarchar(10),
       FName NVarchar(40),LName NVarchar(40),EAccount Nvarchar(20),PrintDate DATETIME)

       DECLARE CustomerAccount CURSOR FOR
              SELECT vr.Vehicle_Registration as 'Vehicle Reg',
                            Convert(DATETIME,tt.TT_DateTime) as 'Date/Time',
                            tt.TT_Amount as Charge,
                            CASE tt.TT_IsPaid
                                   WHEN 0 THEN 'N'
                                   ELSE 'Y'
                            END as Paid,
                            -1*(SELECT SUM(TT_Amount) FROM TollTransaction WHERE
TTagID = tt.TTagID AND TT_IsPaid = 0) as Balance,
                            c.Cus_Area,
                            c.Cus_City,
                            c.Cus_StreetName,
                            c.Cus_StreetNumber,
                            c.Cus_FName,
                            c.Cus_LName,
                            @EAccountFromSystem as EAccount,
                            Convert(DATE,GETDATE()) as PrintDate
                            FROM TollTransaction tt
                            JOIN  TollTag tl
                            ON tt.TTagID = tl.TTagID
                            JOIN Customer c
                            ON tl.CustomerID = c.CustomerID
                            JOIN VehicleRegistration vr
                            ON c.CustomerID = vr.CustomerID
                            WHERE c.CustomerID = @CustomerID AND
DATEDIFF(DAY,CONVERT(DATETIME,tt.TT_DateTime),GETDATE()) > 30

       OPEN CustomerAccount

       FETCH NEXT FROM CustomerAccount INTO
@VehicleReg,@DateTime,@Charge,@IsPaid,@Balance,@Cus_Area,@Cus_City,@Cus_StreetName,

       @Cus_StreetNumber,@Cus_FName,@Cus_LName,@EAccount,@PrintDate

       WHILE(@@FETCH_STATUS = 0)
       BEGIN

              --insert record into temporary table
              INSERT INTO #TollPayments VALUES(
@VehicleReg,@DateTime,@Charge,@IsPaid,@Balance,@Cus_Area,@Cus_City,@Cus_StreetName,

       @Cus_StreetNumber,@Cus_FName,@Cus_LName,@EAccount,@PrintDate)

              FETCH NEXT FROM CustomerAccount INTO
@VehicleReg,@DateTime,@Charge,@IsPaid,@Balance,@Cus_Area,@Cus_City,@Cus_StreetName,

       @Cus_StreetNumber,@Cus_FName,@Cus_LName,@EAccount,@PrintDate

       END


       --UPDATE THE Account Number
       -- Increment it by 1
       SET @EAccountFromSystem = @EAccountFromSystem + 1;
       UPDATE #SystemVariable SET EAccount = @EAccountFromSystem

       CLOSE CustomerAccount
       DEALLOCATE CustomerAccount
```

```
	--Return All records in the temporary table to a calling application
	SELECT * FROM #TollPayments
END
```

E-tag Ownership stored procedure

```
CREATE PROCEDURE spETagOwnership
AS
BEGIN
	SELECT  tt.CustomerID,
			c.Cus_FName,
			c.Cus_LName,
			(SELECT COUNT(vr.VehicleID)
				FROM VehicleRegistration vr
				WHERE vr.CustomerID = tt.CustomerID
				GROUP BY vr.CustomerID) as VehiclesOwned,
				COUNT(tt.TTagID) as eTagsPurchased
			FROM TollTag tt
			JOIN Customer c
			ON tt.CustomerID = c.CustomerID
			GROUP BY tt.CustomerID,c.Cus_FName,c.Cus_LName
			HAVING COUNT(tt.TTagID) <> (SELECT COUNT(vr.VehicleID)

	FROM VehicleRegistration vr
	WHERE vr.CustomerID = tt.CustomerID
	GROUP BY vr.CustomerID)
END
```

Out Statnding Tolls Report stored procedure

```
CREATE PROCEDURE spOutStandingTollsReport
AS
BEGIN

	SELECT ro.ROffice_Region, SUM(tt.TT_Amount) as OutStanding
		FROM RegionOffice ro
		JOIN Gantry g
		ON ro.ROfficeID = g.ROfficeID
		JOIN TollTransaction tt
		ON g.GantryID = tt.GantryID
		WHERE tt.TT_IsPaid = 0
		GROUP BY ro.ROffice_Region
END
```

Expired Drivers Licence stored procedure

```
CREATE PROCEDURE spExpiredDriversLicense
AS
BEGIN

	SELECT c.CustomerID,c.Cus_FName,c.Cus_LName,dl.DLicense_ExpireDate as
ExpiredDate
	FROM Customer c
	JOIN DriverLicense dl
	ON c.CustomerID = dl.CustomerID
	WHERE c.CustomerID IN (SELECT CustomerID FROM VehicleRegistration)
	AND dl.DLicense_ExpireDate < Convert(DATE,GETDATE())
```

```
END
```

Staff Per Region Office stored procedure

```
CREATE PROCEDURE spStaffPerRegionOffice
AS
BEGIN

       SELECT ro.ROffice_Name as Region,s.Staff_Type,COUNT(s.StaffID) as Number
               FROM RegionOffice ro
               JOIN Staff s
               ON ro.ROfficeID = s.ROfficeID
               GROUP BY ROffice_Name,Staff_Type
               ORDER BY ROffice_Name,Staff_Type
END
```

Maintain LastLoginDate stored procedure

```
CREATE PROCEDURE spMaintainLastLogin
@UserName NVarchar(50),
@Password NVarchar(200),
@ErrorMessage NVarchar(200) OUTPUT
AS
BEGIN

       --Add Field to the User entity called LastLoginDate--
       /*
              ALTER TABLE "User"
              ADD LastLoginDate DATETIME;
       */

       SELECT *
              FROM "User"
              WHERE UserName = @UserName AND
              UserPassword = sys.fn_varbintohexsubstring(0, HashBytes('SHA1',
@Password), 1, 0)

       IF(@@ROWCOUNT <> 0)
              -- Update LastLoginDate after successful login
              UPDATE "User" SET LastLoginDate = GETDATE() WHERE UserName = @UserName
       ELSE
              SET @ErrorMessage = 'Either the password is incorrect or the Username
does not exist'
END
```

Users inactive for the last month stored procedure

```
CREATE PROCEDURE spInActiveLastMonth
AS
BEGIN
       SELECT * FROM "User" WHERE MONTH(LastLoginDate) =
MONTH(DATEADD(mm,DATEDIFF(mm,0,GETDATE())-1,0))
END
```

Generating Random Password stored procedure

Could not use a FUNCTION because we not allowed to use RAND() inside a user-defined function

```
CREATE PROCEDURE spGeneratePassword
@LengthOfPassword INT
AS
BEGIN

       DECLARE @Password     VARCHAR(20)
       DECLARE @ValidCharacters   VARCHAR(100)
       DECLARE @PasswordIndex    INT
       DECLARE @CharacterIndex   INT


       SET @ValidCharacters =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890*'

       SET @PasswordIndex = 1
       SET @Password = ''

       WHILE @PasswordIndex <= @LengthOfPassword
       BEGIN
        SELECT @CharacterIndex = ABS(CAST(CAST(RAND() AS VARBINARY) AS INT)) %
       LEN(@ValidCharacters) + 1

        SET @Password = @Password + SUBSTRING(@ValidCharacters, @CharacterIndex, 1)
        SET @PasswordIndex = @PasswordIndex + 1
       END
       SELECT @Password
END
```

Create AuditTrail Entity

```
CREATE TABLE AuditTrailId(
AuditTrain_Id int IDENTITY(1,1) PRIMARY KEY,
AuditTrail_TableAffected NVarchar(100),
AuditTrail_PreviousValue NVarchar(100),
AutiTrail_NewValue NVarchar(100),
AuditTrail_UserId NVarchar(100)
)
```

Add new records to the AuditTrail entity Stored Procedure

```
CREATE PROCEDURE spAddAuditTrail
@AuditTrail_TableAffected NVarchar(100),
@AuditTrail_PreviousValue NVarchar(100),
@AutiTrail_NewValue NVarchar(100)
AS
BEGIN
```

```
       INSERT INTO
AuditTrailId(AuditTrail_TableAffected,AuditTrail_PreviousValue,AutiTrail_NewValue,Audi
tTrail_UserId)

       VALUES(@AuditTrail_TableAffected,@AuditTrail_PreviousValue,@AutiTrail_NewValue,
SUSER_SNAME())

END
```

Create a trigger for the table customer

```
CREATE TRIGGER trCustomerAfterUpdate
ON Customer
FOR UPDATE
AS
BEGIN
       DECLARE @AuditTrail_PreviousValue NVarchar(20)
       DECLARE @AuditTrail_NewValue NVarchar(20)


       SELECT @AuditTrail_NewValue = i.Cus_EAccount,
                     @AuditTrail_PreviousValue = d.Cus_EAccount
                     FROM inserted i
                     JOIN deleted d
                     ON i.CustomerID = d.CustomerID

       EXECUTE spAddAuditTrail
'Customer',@AuditTrail_PreviousValue,@AuditTrail_NewValue
END
```

Maintain ErrorLog Stored procedure

```
CREATE PROCEDURE spMaintainErrorLog
@ErrorMessage VARCHAR(5000),
@ErrorSeverity INT,
@ErrorProcedure VARCHAR(100)
AS
BEGIN

       INSERT INTO
ErrorLog(ErrorLog_Date,ErrorLog_Message,ErrorLog_Severity,ErrorLog_Procedure)

       VALUES(GETDATE(),@ErrorMessage,@ErrorSeverity,@ErrorProcedure)

       --How I will Call StoredProcedure(spMaintainErrorLog) from another
StoredProcedure
       --The Below code will be in the body of the calling storedProcedure

       /*
              BEGIN TRY

                     -- Complex code comes in here
                     -- If any error occurs while executing any code segment in
here(TRY block)
                     -- Control will be send to CATCH block to execute statements in
the CATCH block

              END TRY
              BEGIN CATCH

                     -- Normally ROLLBACK statement is placed here, to undo Changes
done in the TRY block
                     -- This is done to maintain data integrtity,leaving a database in
the consistent state

                     **Ultimately i will call the storedProcedure in here as shown
below**

                     DECLARE @ErrorMessage VARCHAR(5000)
                     DECLARE @ErrorSeverity INT
                     DECLARE @ErrorProcedure VARCHAR(100)

                     SELECT @ErrorMessage = ERROR_MESSAGE(), @ErrorSeverity =
ERROR_SEVERITY(),@ErrorProcedure = ERROR_PROCEDURE()

                     EXECUTE spMaintainErrorLog
@ErrorMessage,@ErrorSeverity,@ErrorProcedure

              END CATCH
       */
END
```