# DataBase IceCreamShop

Sarah Angelica Carvalho Sobral

- **Create DataBase;**

```
CREATE DATABASE IceCreamShop
```

- **Create Tables;**

```
USE IceCreamShop;

CREATE TABLE Company (
		IdCompany int Identity Primary Key,
		NameCompany varchar(50) not null,
		Status int not null DEFAULT 1,
		FlAuthoritativeReceipt bit not null
		Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)

CREATE TABLE  Create Table Office (
		IdOffice int Identity Primary Key,
		NameOffice varchar(50) not null,
		DescriptionOffice varchar(255),
		Discount decimal(5,2),
		CompanyId int not null,
		Status int not null DEFAULT 1,
		Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
ALTER TABLE Office
ADD CONSTRAINT FK_Office_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)

CREATE TABLE Address (
		IdAddress int Identity Primary Key,
		Cep char(8) not null,
		Logradouro varchar(255) not null,
		Numero varchar(5) not null,
		Complemento varchar(255),
		Bairro varchar(50) not null,
		Cidade varchar(50) not null,
		Uf char(2) not null,
)

CREATE TABLE Employee (
		IdEmployee int Identity Primary Key,
		NameEmployee varchar(50) not null,
		Birth date not null,
		Admission date not null,
		Salary decimal(7,2) not null,
		AddressId int not null,
		OfficeId int not null,
		CompanyId int not null,
		HaveLogin bit not null,
                Permission int,
                LoginUser varchar(10),
                PasswordUser varchar(255),
                Status int not null DEFAULT 1,
                Created datetime not null DEFAULT CURRENT_TIMESTAMP

)
ALTER TABLE Employee
ADD CONSTRAINT FK_Employee_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)
ALTER TABLE Employee
ADD CONSTRAINT FK_Employee_Address
FOREIGN KEY (AddressId) REFERENCES Address(IdAddress)
ALTER TABLE Employee
ADD CONSTRAINT FK_Employee_Office
FOREIGN KEY (OfficeId) REFERENCES Office(IdOffice)

CREATE TABLE Phone (
                IdPhone int Identity Primary Key,
                TypePhone int not null,
                DDD char(2) not null,
                Number varchar(9) not null,
                Status int not null DEFAULT 1,
                EmployeeId int not null,
)
ALTER TABLE Phone
ADD CONSTRAINT FK_Phone_Employee
FOREIGN KEY (EmployeeId) REFERENCES Employee(IdEmployee)

CREATE TABLE Category (
                IdCategory int Identity Primary Key,
                NameCategory varchar(50) not null,
                DescriptionCategory varchar(255),
                CompanyId int not null,
                Status int not null DEFAULT 1,
                Created datetime not null DEFAULT CURRENT_TIMESTAMP

)
ALTER TABLE Category
ADD CONSTRAINT FK_Category_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)

CREATE TABLE UnitMeasure (
                IdUnitMeasure int Identity Primary Key,
                NameUnitMeasure varchar(50) not null,
                DescriptionUnitMeasure varchar(255),
                CompanyId int not null,
                Status int not null DEFAULT 1,
                Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
```

```
ALTER TABLE UnitMeasure
ADD CONSTRAINT FK_UnitMeasure_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)

CREATE TABLE Product (
                IdProduct int Identity Primary Key,
                NameProduct varchar(50) not null,
                DescriptionProduct varchar(255),
                CostPrice decimal(7,2) not null,
                SalePrice decimal(7,2) not null,
                MinStock int not null,
                SellNegative bit not null,
                AmountStock int not null,
                CategoryId int not null,
                UnitMeasureId int not null,
                CompanyId int not null,
                Status int not null DEFAULT 1,
                Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
ALTER TABLE Product
ADD CONSTRAINT FK_Product_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)
ALTER TABLE Product
ADD CONSTRAINT FK_Product_Category
FOREIGN KEY (CategoryId) REFERENCES Category(IdCategory)
ALTER TABLE Product
ADD CONSTRAINT FK_Product_UnitMeasure
FOREIGN KEY (UnitMeasureId) REFERENCES UnitMeasure(IdUnitMeasure)

CREATE TABLE EntryStock (
                IdStock int Identity Primary Key,
                FabricationDate date not null,
                ExpirationDate date not null,
                ProductBatch varchar(10) not null,
                Amount int not null,
                ProductId int not null,
                CompanyId int not null,
                Status int not null DEFAULT 1,
                Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
ALTER TABLE EntryStock
ADD CONSTRAINT FK_Stock_Campany
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)
ALTER TABLE EntryStock
ADD CONSTRAINT FK_Stock_Product
FOREIGN KEY (ProductId) REFERENCES Product(IdProduct)

CREATE TABLE DebitCard (
                IdDebitCard int Identity Primary Key,
                NameDebitCard varchar(50) not null,
		DescriptionDebitCard varchar(255),
		RateDebitCard decimal(5,2) not null,
		CompanyId int not null,
		Status int not null DEFAULT 1,
		Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
ALTER TABLE DebitCard
ADD CONSTRAINT FK_DebitCard_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)

CREATE TABLE CreditCard (
		IdCreditCard int Identity Primary Key,
		NameCreditCard varchar(50) not null,
		DescriptionCreditCard varchar(255),
		RateCreditCard decimal(5,2) not null,
		CompanyId int not null,
		Status int not null DEFAULT 1,
		Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
ALTER TABLE CreditCard
ADD CONSTRAINT FK_CreditCard_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)

CREATE TABLE Sale (
		IdSale int Identity Primary Key,
		CompanyId int not null,
		EmployeeId int not null,
		TotalPrice decimal(7,2) not null,
		Status int not null DEFAULT 1,
		Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
ALTER TABLE Sale
ADD CONSTRAINT FK_Sale_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)
ALTER TABLE Sale
ADD CONSTRAINT FK_Sale_User
FOREIGN KEY (EmployeeId) REFERENCES Employee(IdEmployee)

CREATE TABLE Payment (
		IdPayment int Identity Primary Key,
		SaleId int not null,
		TypePayment int not null,
		DebitCardId int,
		TotalPrice decimal(7,2) not null,
		InstallmentPrice decimal(7,2) not null,
		DiscontApply decimal(7,2) null,
		CreditCardId int,
		CompanyId int not null,
		EmployeeId int not null,
		CodePaymentCard varchar(50) null,
                InstallmentNumber int not null,
                ForecastDatePayment date not null,
                Status int not null DEFAULT 1,
                Created datetime not null  DEFAULT CURRENT_TIMESTAMP
)
ALTER TABLE Payment
ADD CONSTRAINT FK_Payment_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)
ALTER TABLE Payment
ADD CONSTRAINT FK_Payment_Employee
FOREIGN KEY (EmployeeId) REFERENCES Employee(IdEmployee)
ALTER TABLE Payment
ADD CONSTRAINT FK_Payment_DebitCard
FOREIGN KEY (DebitCardId) REFERENCES DebitCard(IdDebitCard)
ALTER TABLE Payment
ADD CONSTRAINT FK_Payment_CreditCard
FOREIGN KEY (CreditCardId) REFERENCES CreditCard(IdCreditCard)
ALTER TABLE Payment
ADD CONSTRAINT FK_Payment_Sale
FOREIGN KEY (SaleId) REFERENCES Sale(IdSale)

CREATE TABLE SaleProduct (
                IdSaleProduct int Identity Primary Key,
                SaleId int not null,
                ProductId int not null,
                Amount int not null DEFAULT 0,
                Status int not null DEFAULT 1,
)
ALTER TABLE SaleProduct
ADD CONSTRAINT FK_SaleProduct_Sale
FOREIGN KEY (SaleId) REFERENCES Sale(IdSale)
ALTER TABLE SaleProduct
ADD CONSTRAINT FK_SaleProduct_Product
FOREIGN KEY (ProductId) REFERENCES Product(IdProduct)

CREATE TABLE Log (
                IdLog int Identity Primary Key,
                Old varchar(255),
                New varchar(255) not null,
                Who int not null,
                Created datetime not null DEFAULT CURRENT_TIMESTAMP,
                CompanyId int,
                EmployeeId int,
                OfficeId int,
                CategoryId int,
                UnitMeasureId int,
                ProductId int,
                SaleId int,
                PaymentId int,
                CreditCardId int,
                EntryStockId int,
                DebitCardId int
)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Company
FOREIGN KEY (CompanyId) REFERENCES Company(IdCompany)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Office
FOREIGN KEY (OfficeId) REFERENCES Office(IdOffice)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Who
FOREIGN KEY (Who) REFERENCES Employee(IdEmployee)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Employee
FOREIGN KEY (EmployeeId) REFERENCES Employee(IdEmployee)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Category
FOREIGN KEY (CategoryId) REFERENCES Category(IdCategory)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_UnitMeasure
FOREIGN KEY (UnitMeasureId) REFERENCES UnitMeasure(IdUnitMeasure)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Product
FOREIGN KEY (ProductId) REFERENCES Product(IdProduct)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Sale
FOREIGN KEY (SaleId) REFERENCES Sale(IdSale)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_Payment
FOREIGN KEY (PaymentId) REFERENCES Payment(IdPayment)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_CreditCard
FOREIGN KEY (CreditCardId) REFERENCES CreditCard(IdCreditCard)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_DebitCard
FOREIGN KEY (DebitCardId) REFERENCES DebitCard(IdDebitCard)
ALTER TABLE Log
ADD CONSTRAINT FK_Log_EntryStock
FOREIGN KEY (EntryStockId) REFERENCES EntryStock(IdStock)
```

- **Create Trigger**

This trigger ensures that registered employees who do not have a login do not have permission, username and password registered in the database. And if the registered employee has access to the system, he needs to have this access data registered, if not, it will give an error. When inserting the new employee, the trigger returns an id. Due to the Asp.Net Add method, it requires the trigger to return something so that there is no error when the trigger executes a select after inserting the data.

```
CREATE TRIGGER tgr_Login_Employee on Employee
        INSTEAD OF INSERT
        AS
        BEGIN
                DECLARE
                        @name VARCHAR(50),
                        @birth date,
                        @admission date,
                        @salary decimal(7,2),
                        @addressId int,
                        @officeId int,
                        @companyId int,
                        @havelogin BIT,
                        @login VARCHAR(10),
                        @permission INT,
                        @password VARCHAR(255),
                        @status int

                SELECT @name = NameEmployee, @birth = Birth, @admission
                = Admission, @salary = Salary, @addressId = AddressId,
                @officeId = OfficeId, @companyId = CompanyId, @permission =
                Permission, @status = Status, @havelogin = Havelogin, @login =
                LoginUser, @password = PasswordUser FROM INSERTED;


                IF(@havelogin = 1 AND (@password = '' OR @password IS
                NULL) AND (@login = '' OR @login IS NULL) AND (@permission
                = 0))

                        THROW 51000, 'NOT VALID PASSWORD', 1;

                ELSE IF(@havelogin = 0)
                        BEGIN
                                INSERT INTO Employee (NameEmployee, Birth,
                                Admission, Salary, AddressId, OfficeId, CompanyId,
                                HaveLogin, Permission, LoginUser, PasswordUser,
                                Status)
                                VALUES      (@name, @birth, @admission,
                                @salary, @addressId, @officeId, @companyId,
                                @havelogin, null, null, null, @status);

                                SELECT IdEmployee From INSERTED;
                        END
                ELSE
```

```
                    BEGIN
                            INSERT INTO Employee (NameEmployee, Birth,
                            Admission, Salary, AddressId, OfficeId, CompanyId,
                            HaveLogin, LoginUser, Permission, PasswordUser,
                            Status)
                            VALUES (@name, @birth, @admission, @salary,
                            @addressId, @officeId, @companyId, @havelogin,
                            @login, @permission, @password, @status)

                            SELECT IdEmployee From INSERTED;
               END
END
```

- **Insert Data in the Database**

The superadmin data must already be registered in the database, in order to be able to access the entire system. Only a superadmin is recommended. You can registered others datas in these or other tables.

```
Insert Into Company (NameCompany, Status, FlAuthoritativeReceipt) Values
('FirstCompany', 1, 1)

        Select * From Company

Insert Into Office (NameOffice, DescriptionOffice, Discount, CompanyId,Status)
Values ('Gerente', 'Responsável Geral', 1, 1,1)

        Select * From Office

Insert Into Address (Cep, Logradouro, Numero, Bairro, Cidade, Uf)
VALUES ('49001078', 'Rua x', '30', 'Aruana', 'Aracaju', 'SE')

        Select * From Address

INSERT Into Employee (NameEmployee, Birth, Admission, Salary, AddressId,
OfficeId,
CompanyId, HaveLogin, Permission, LoginUser, PasswordUser, Status)

VALUES ('SuperAdmin', '1990-11-26', '2020-01-01', 10000.00, 1, 1, 1, 1, 1,
'superadmin',
'C7AD44CBAD762A5DA0A452F9E854FDC1E0E7A52A38015F23F3EAB1D80
B931DD472634DFAC71CD34EBC35D16AB7FB8A90C81F975113D6C7538D
C69DD8DE9077EC', 1) //password admin

        Select * From Employee
```

- **Create Procedures**

These two procudures, will be executed by Jobs

- **Update Quantity of products in stock**

The first Job, will execute the first procedure, this procedure will perform the second procedure for each company. Job will update the quantity of product in stock (this record belongs to the product table), where it will be executed every hour, capturing all stock entries that did not occur in this period and are not checked, and will capture all sales that occurred within the same period and were not checked to subtract from the stock entries, updating the quantity of products in the stock as close to the real as possible. The stock will not be updated automatically, but periodically.

```
CREATE PROCEDURE GetCompaniesToUpdateStock
AS
	BEGIN TRANSACTION GetCompanies
		BEGIN TRY
			DECLARE @IdCompany INT;
			DECLARE @GetCompany CURSOR;

			SET @GetCompany = CURSOR FOR
			SELECT IdCompany FROM Company

			OPEN @GetCompany
			FETCH NEXT FROM @GetCompany INTO @IdCompany

			WHILE @@FETCH_STATUS = 0
			BEGIN
				EXECUTE UpdateAmountProduct @CompanyId =
@IdCompany

				FETCH NEXT
				FROM @GetCompany INTO @IdCompany
			END

			CLOSE @GetCompany
			DEALLOCATE @GetCompany

			COMMIT TRANSACTION GetCompanies
	END TRY

	BEGIN CATCH
		ROLLBACK TRANSACTION GetCompanies
	END CATCH
```

```
GO

CREATE PROCEDURE UpdateAmountProduct @CompanyId int
AS
        BEGIN TRANSACTION  UpdateAmountProduct
                BEGIN TRY
        /*Step 1 IF Exits DROP Temporary Table #UptadeAmount*/
                IF OBJECT_ID('tempdb.dbo.#UptadeAmount') IS NOT NULL
                        DROP TABLE dbo.#UptadeAmount;

        /*Step 2 CREATE Temporary Table #UptadeAmount*/
                CREATE TABLE #UptadeAmount
                (
                        IdTemp int Identity,
                        ProductId int not null,
                        AmountToUpdate int not null Default 0
                );

        /*Step 3 Get amount1 in entryStock and INSERT in #UptadeAmount*/
                INSERT INTO #UptadeAmount (ProductId, AmountToUpdate)
                        Select ES.productId, Sum(ES.amount) from EntryStock as
ES
                                where status = 1
                                and CompanyId = @CompanyId
                                group by ProductId

        /*Step 4 Upadate Stock Status to 2*/
                DECLARE @IdStock INT;
                DECLARE @GetIdStock CURSOR;

                SET @GetIdStock = CURSOR FOR
                SELECT IdStock FROM EntryStock where status = 1 and
CompanyId = @CompanyId

                OPEN @GetIdStock
                FETCH NEXT FROM @GetIdStock INTO @IdStock

                WHILE @@FETCH_STATUS = 0
                BEGIN
                        UPDATE EntryStock SET Status = 2 WHERE IdStock =
@IdStock

                        FETCH NEXT
                        FROM @GetIdStock INTO @IdStock
                END

                CLOSE @GetIdStock
                DEALLOCATE @GetIdStock

                -----------------------------------------
```

```
		/*Step 5 Get amount2 in SALEPRODUCT and INSERT OR
UPDATE*/
		DECLARE @IdSaleProduct INT;
		DECLARE @IdProduct INT;
		DECLARE @Amount INT;
		DECLARE @GetProductId CURSOR;

		SET @GetProductId = CURSOR FOR
		Select SP.ProductId, Sum(SP.amount) from SaleProduct as SP
			inner join Sale as S on S.IdSale = SP.SaleId
			and S.CompanyId = @CompanyId
			and S.Status = 2 /*finished*/
			and SP.Status = 1
			group by SP.ProductId

		OPEN @GetProductId
		FETCH NEXT FROM @GetProductId INTO @IdProduct,
@Amount

		WHILE @@FETCH_STATUS = 0
		BEGIN
			IF (NOT EXISTS(SELECT ProductId, AmountToUpdate
FROM #UptadeAmount WHERE ProductId = @IdProduct))
				BEGIN
					INSERT INTO #UptadeAmount (ProductId,
AmountToUpdate)
						VALUES (@IdProduct, @Amount*-1)
				END
			ELSE
				BEGIN
					Update #UptadeAmount
						Set AmountToUpdate =
AmountToUpdate - @Amount
						where ProductId = @IdProduct

				END

			/*Step 6 Upadate SaleProduct Status to 2*/
			UPDATE SaleProduct
			SET Status = 2
			From SaleProduct as SP
			inner Join Sale as S on S.IdSale = SP.SaleId and
S.CompanyId = @CompanyId and S.Status = 2 and SP.Status = 1

			FETCH NEXT
			FROM @GetProductId INTO @IdProduct, @Amount
		END

		CLOSE @GetProductId
		DEALLOCATE @GetProductId
		-----------------------------------------
		/*Step 7 Get SUBTRACTION RESULT in table #UptadeAmount
and UPDATE Amount Product*/
		DECLARE @IdProd INT;
		DECLARE @AmountUpdate INT;
		DECLARE @GetId Cursor;

		Set @GetId = Cursor for
		Select ProductId, AmountToUpdate From #UptadeAmount

		OPEN @GetId
		FETCH NEXT FROM @GetId INTO @IdProd, @AmountUpdate
		While @@FETCH_STATUS = 0
		BEGIN
			Update Product SET AmountStock = AmountStock +
@AmountUpdate
			Where IdProduct =  @IdProd

			FETCH NEXT FROM @GetId INTO @IdProd,
@AmountUpdate
		END

		CLOSE @GetId
		DEALLOCATE @GetId

		COMMIT TRANSACTION UpdateAmountProduct

	END TRY

	BEGIN CATCH
		ROLLBACK TRANSACTION UpdateAmountProduct
	END CATCH
```

- **Change Sale Status to Expired**

The second job will be performed daily, at the opening hours of the ice cream shop (example: 8:00 am to 9:00 pm) every 30 minutes, will execute the first procedure that will execute the second procedure for each company, which changes the status of sales that have not been finalized or canceled with or more than 30 minutes for expired status.

```
CREATE PROCEDURE GetCompaniesToExpireSale
AS
	BEGIN TRANSACTION GetCompanies
		BEGIN TRY
			DECLARE @IdCompany INT;
```

```
                    DECLARE @GetCompany CURSOR;

                    SET @GetCompany = CURSOR FOR
                    SELECT IdCompany FROM Company

                    OPEN @GetCompany
                    FETCH NEXT FROM @GetCompany INTO @IdCompany

                    WHILE @@FETCH_STATUS = 0
                    BEGIN
                        EXECUTE ChangeStatusSales @CompanyId =
@IdCompany


                        FETCH NEXT
                        FROM @GetCompany INTO @IdCompany
                    END

                    CLOSE @GetCompany
                    DEALLOCATE @GetCompany

                    COMMIT TRANSACTION GetCompanies
        END TRY

        BEGIN CATCH
                ROLLBACK TRANSACTION GetCompanies
        END CATCH
GO

CREATE PROCEDURE [dbo].[ChangeStatusSales] @CompanyId int
AS
        BEGIN TRANSACTION ChangeStatusSales

                BEGIN TRY



        /*Step 1 Find Sales with Status 2*/

                DECLARE @IdSale INT;

                DECLARE @GetSales CURSOR;


                SET @GetSales = CURSOR FOR

                SELECT IdSale FROM Sale as S Where S.CompanyId =
@CompanyId and S.Status = 1/*PENDING*/ and
DATEDIFF(mi,S.Created,GETDATE()) >= 30;
```

```
        OPEN @GetSales
        FETCH NEXT FROM @GetSales INTO @IdSale

        WHILE @@FETCH_STATUS = 0
        BEGIN
            UPDATE Sale SET Status = 3 WHERE IdSale = @IdSale

            FETCH NEXT
            FROM @GetSales INTO @IdSale
        END

        CLOSE @GetSales
        DEALLOCATE @GetSales

        COMMIT TRANSACTION ChangeStatusSales
    END TRY

    BEGIN CATCH
        ROLLBACK TRANSACTION ChangeStatusSales
    END CATCH
```

- **Change Payment Status to Expired**

The third job will be performed once a day, will execute the first procedure that will execute the second procedure for each company that the flag FlAuthoritativeReceipt is false (company that have this flag true, all payment is mark as pay), which changes the status of payments that have not been receved and the forecast date payment already expired.

```
CREATE PROCEDURE GetCompaniesToExpirePayment
AS
BEGIN TRANSACTION GetCompanies
```

```
                BEGIN TRY
                        DECLARE @IdCompany INT;
                        DECLARE @GetCompany CURSOR;

                        SET @GetCompany = CURSOR FOR
                        SELECT IdCompany FROM Company Where
FlAuthoritativeReceipt = 0

                        OPEN @GetCompany
                        FETCH NEXT FROM @GetCompany INTO @IdCompany

                        WHILE @@FETCH_STATUS = 0
                        BEGIN
                                EXECUTE ExpiredPayment @CompanyId =
@IdCompany

                                FETCH NEXT
                                FROM @GetCompany INTO @IdCompany
                        END

                        CLOSE @GetCompany
                        DEALLOCATE @GetCompany

                        COMMIT TRANSACTION GetCompanies
        END TRY

        BEGIN CATCH
                ROLLBACK TRANSACTION GetCompanies
END CATCH
GO
```

```
CREATE PROCEDURE [dbo].[ExpiredPayment] @CompanyId int

AS

BEGIN TRANSACTION ExpiredPayment

                BEGIN TRY


        /*Step 1 Find Sales with Status 2*/

                DECLARE @IdPayment INT;

                DECLARE @GetPayment CURSOR;


                SET @GetPayment = CURSOR FOR

                SELECT IdPayment FROM Payment as P Where P.CompanyId =
@CompanyId and P.Status = 2/*Payable*/ and
DATEDIFF(DAY,P.forecastDatePayment,GETDATE()) >= 1;


                OPEN @GetPayment

                FETCH NEXT FROM @GetPayment INTO @IdPayment


                WHILE @@FETCH_STATUS = 0

                BEGIN

                        UPDATE Payment SET Status = 3 WHERE IdPayment =
@IdPayment

                        FETCH NEXT

                        FROM @GetPayment INTO @IdPayment

                END


                CLOSE @GetPayment

                DEALLOCATE @GetPayment


                COMMIT TRANSACTION ExpiredPayment

        END TRY
```

```
BEGIN CATCH
	ROLLBACK TRANSACTION ExpiredPayment
END CATCH
```