

HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION
FACULTY FOR HIGH QUALITY TRAINING



COURSE PROJECT

DATABASE MANAGEMENT SYSTEM

CONVENIENCE STORE

MANAGEMENT

Lecturer: Mr. Nguyen Thanh Son

Class: DBMS330284E_21_2_01CLC

Member: Group 6

Ho Chi Minh, May 17th 2022

MEMBER OF GROUP 6

No.	Name	ID
1	Nguyễn Huỳnh Thanh Toàn	20110420
2	Bùi Ngọc Ánh	20110354
3	Lê Y Thiện	20110403

EVALUATION AND SCORE

EVALUATION:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

SCORE:.....

TABLE OF CONTENTS

PART 1. SYSTEM DESCRIPTION	7
1.1. Application overview	7
1.2. Functions of application	7
1.3. System analysis and design	8
1.4. Expected interface	9
PART 2. DATABASE ANALYSIS AND DESIGN	11
2.1. Model ERD	11
2.2. Model relationships between tables	11
PART 3. SYSTEM CONFIGURATION AND INSTALLATION	13
3.1. Database Creation and Constraints	13
3.2. Creating statements in Stored Procedure	15
3.2.1. Add account.....	15
3.2.2. Add employee.....	15
3.2.3. Update employee	15
3.2.4. Delete employee	16
3.2.5. List employee	16
3.2.6. Find employee	16
3.2.7. Add customer	16
3.2.8. Add product.....	16
3.2.9. Update product	17
3.2.10. Delete product	17
3.2.11. Add invoice	17
3.2.12. Update invoice.....	17

3.2.13. Delete invoice.....	18
3.2.14. Add detail	18
3.2.15. Update detail.....	18
3.2.16. Delete detail.....	18
3.2.17. Add manufacturer	18
3.2.18. Update manufacturer	18
3.2.19. Delete manufacturer	18
3.2.20. Add stock.....	19
3.2.21. Update stock	19
3.2.22. Delete stock	19
3.2.23. Add types.....	19
3.2.24. Update type.....	19
3.2.25. Delete type.....	19
3.3. Database connection.....	20
3.4. Decentralization	20
3.4.1. Database table for authorization.....	21
3.4.2. Decentralized code on the database	21
3.4.3. Decentralized code on C#.....	22
3.5. Triggers and transactions	22
3.5.1. Settings to create login accounts according to user permissions	22
3.5.2. Install login account updates according to user permissions	23
3.5.3. Setting to delete login accounts according to user permissions	24
3.5.4. Setting to update total pay for invoice	24
3.5.5. Setting to decrease total pay for invoice	24

3.5.6.	Setting to check the invoice exists or not.....	24
3.5.7.	Setting to update the invoice	25
3.6.	Function.....	25
3.6.1.	Total number of employees.....	25
3.6.2.	Total number of customers.....	25
3.6.3.	Total number of products	26
3.6.4.	Check account	26
3.6.5.	Find product by name and type	26
3.6.6.	Find customer	26
3.6.7.	Show detail of invoice.....	26
3.6.8.	Show detail	27
3.6.9.	Auto show information of product	27
3.6.10.	Total pay of invoice.....	27
3.6.11.	Search employees by id.....	27
3.6.12.	Employee gender statistics	27
3.7.	View	27
3.7.1.	View for products	27
3.7.2.	View for customers	28
3.7.3.	View for type.....	28
3.7.4.	View for stock	28

PART 1. SYSTEM DESCRIPTION

1.1. Application overview

One of the urgent requirements today is to put information technology into management in order to reduce human labor, save time, increase accuracy, narrow storage space, and avoid loss of storage space, exit and ensure the safety of data.

Putting information technology into the "Convenience Store Management Application" for management is essential, because we spend very little time but get high efficiency, accuracy and convenience quickly. The software can meet the needs of convenience stores, helping to make sales and sales statistics quickly and efficiently.

The application helps users have the most intuitive view of data information. With the manager will be able to manage personnel and products easily without spending too much effort and time. With employees, they will see accurate information about products and functions related to their personal business. Users only need to correctly manipulate the pre-formatted items on the interface, the program ensures the most accurate data is retrieved.

With the "Convenience Store Management Application", we will conduct all operations right on a computer or laptop and can check out much more quickly and conveniently.

1.2. Functions of application

Build convenience store management software to ensure the following functions:

- Add, edit, delete, update employees if there is a change. Search for employees by Employee ID. Calculate total employees.
- Add, edit, update customers if there is a change. Search for customers by phone number. Calculate total customers.
- Add, edit, delete, update products if there is a change. Search products by ID and type. Total products.
- Add supply.
- Create and update new invoices. Search invoices by Invoice ID and by price greater than 50000.
- Add, edit, delete, update detailed invoices
- Decentralize management and employee login.

- Statistics of revenue, profit by day and number of male and female employees.

1.3. System analysis and design

The subject area applies in the convenience store. The functionality focuses on the management of the necessary information of the convenience store. The application is suitable for stores with many employees, managers, etc.

A convenience store needs a database to manage, including the following information:

- **Employee:** Each employee has personal information such as employee name (E-Name), employee's address (E-Address), employee's position (E-Position), employee's phone number (E-Phone), employee's photo (E-Image), employee's birthday (E-Birth), employee's gender (E-Gender), salary (E-Salary) and an employee code (E-ID) used to distinguish it from other employees. An employee is verified by logging in and an employee can confirm multiple invoices.

- **Account:** An employee is identified by logging into his or her account. An account will have an account name (Username), password (Password), employee code (E-ID), email of the account (A-Email) and state of the account (Active). An employee will only have one account.

- **Invoice:** An invoice will be issued by an employee. An invoice consists of the total value of the invoice (I-TotalPay), the invoice printing date (I-Date) and an invoice code (I-ID) used to distinguish between bills. An invoice will be paid by a customer. An invoice will show information including the price (D-Price) and quantity (D-Amount) of one or more products.

- **Customer:** A customer has personal information such as customer name (C-Name), customer phone number (C-Phone), total payment amount (C-TotalPay) and a customer code (C-ID) used to distinguish from other customers. The name of the customer may be null because there are some customers who have not registered their information before paying the invoices. A customer can pay one or more bills at the same time.

- **Product:** A product is recorded by one or more invoices. A product has information such as product name (P-Name), unit name (Unit-Name), price of the product (P-Price), image of the product (P-Image) and a product code (P-ID) used to distinguish one

product from another. A product will belong to a certain product category. A product can be supplied or supplied by a warehouse or a manufacturer

- **Type:** A type can include many different products. A type consists of two pieces of information is a type name (T-Name) and a type code (T-ID) used to distinguish it from another type.

- **Stock:** A stock can hold many products. Information of a stock including the batch ID (Batch ID), import date (ImportDate) and quantity of the item (AmountOfProduct)

- **Manufacture:** A manufacturer can produce many products. A manufacturer consists of two pieces of information is the manufacturer's name (M-Name) and a manufacturer's code (M-ID) used to distinguish it from other manufacturers.

1.4. Expected interface

The login page is used to assign access rights to the Convenience Store Management software including sales staff and store managers.

Employee account interface:

- Home page: helps users access user functions such as creating invoices and viewing product information,...
- Invoice creation page allows sales employee to update invoices in two ways, with customers and without customers.
- The product page allows employee to view list product information.
- Customer page allows employees to add customers

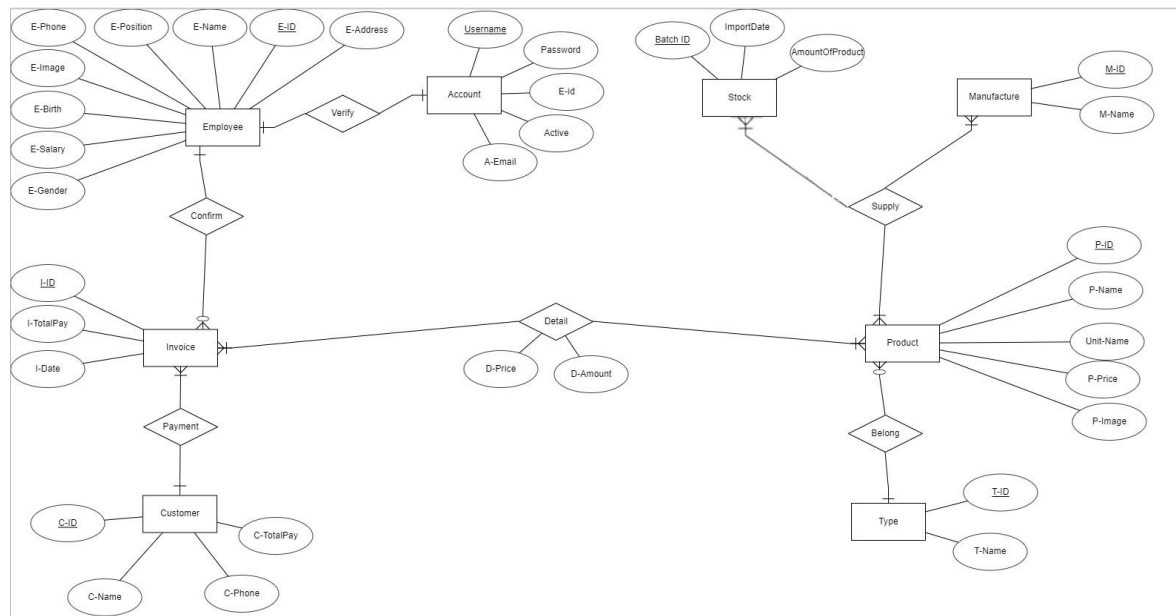
Manager account interface:

- Home page: helps users access user functions such as employee management, invoice management, product management, customer management see gender statistics, add supply, manufacture management, stock management, type management.
- The employee management page displays a list of employees and detailed information about the employee. Management can perform functions such as adding, editing, deleting, updating and searching for employee information in the store by name and id.
- The type management page displays a list of product types. Management can perform functions such as adding, removing and updating product types.

- The product management page displays the product list and detailed information about the product. Management can perform functions such as adding, editing, deleting, updating and searching product information by product name and type product.
- The stock management page allows to enter more information of new product batches such as batch id, import date, amount of product, edit or delete batches.
- Invoice creation page allows management staff to create new invoices and edit, remove invoices.
- Statistical management page showing male and female employees of the store.
- Management can perform functions such as adding, removing and updating manufacture

PART 2. DATABASE ANALYSIS AND DESIGN

2.1. Model ERD



Login (Username, Password, E-ID, Active, Email)

Employee (E-ID, E-Name, E-Position, E-Salary, E-Phone, E-Address, E-Image, E-Gender, E-Brith)

Invoice (I-ID, E-ID, C-ID, I-TotalPay, I-Date)

Customer (C-ID, C-Name, C-TotalPay, C-Phone)

Detail (I-ID, P-ID, D-Price, D-Amount)

Product (P-ID, P-Name, P-Price, T-ID, Unit-Name, P-image)

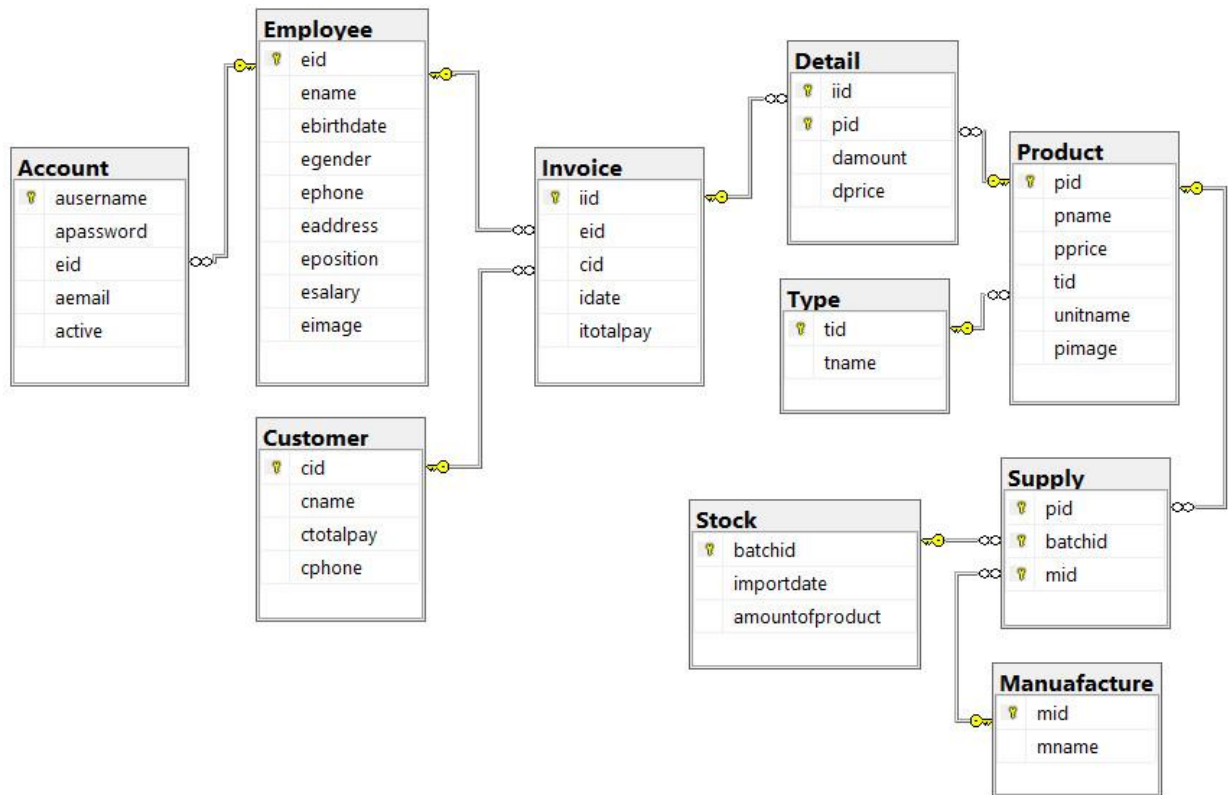
Type (T-ID, T-Name)

Manuafacture (M-ID, M-Name)

Stock (Batch-ID, ImportDate, AmountOfProduct)

Supply (M-ID, P-ID, Batch-ID)

2.2. Model relationships between tables



PART 3. SYSTEM CONFIGURATION AND INSTALLATION

3.1. Database Creation and Constraints

```
CREATE DATABASE StoreManagement
GO
USE StoreManagement
GO
--Phan Code Tao Bang
--Nhan Vien
CREATE TABLE [dbo].[Employee] (
    [eid] nchar(20) NOT NULL,
    [ename] nchar(40) NOT NULL,
    [ebirthdate] date NULL,
    [egender] nchar(15) NULL,
    [ephone] nchar(15) NULL,
    [eaddress] nchar(50) NULL,
    [eposition] nchar(40) NULL,
    [esalary] float NULL,
    [eimage] image NULL,
    CONSTRAINT [PK_Employee] PRIMARY KEY CLUSTERED ([eid] ASC),
)
Go

CREATE TABLE [Account] (
    [username] nchar(15) NOT NULL,
    [apassword] int NOT NULL,
    [eid] nchar(20) NOT NULL,
    [aemail] VARCHAR(100) NULL,
    [active] BIT NULL,
    CONSTRAINT [PK_Account] PRIMARY KEY CLUSTERED ([username] ASC),
    CONSTRAINT [FK_Account_Employee_eid] FOREIGN KEY ([eid]) REFERENCES [Employee] ([eid]) ON
DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [CK_Account_aemail] CHECK ([aemail] like '%@%_._%')
)
Go
--Khach Hang
CREATE TABLE [Customer] (
    [cid] nchar(20) NOT NULL,
    [cname] nchar(40) NULL,
    [ctotalpay] float NULL,
    [cphone] nchar(15) NOT NULL,
    CONSTRAINT [PK_Customer] PRIMARY KEY CLUSTERED ([cid] ASC)
)
Go
--Loai San Pham
CREATE TABLE [Type] (
    [tid] nchar(20) NOT NULL,
    [tname] nchar(20) NOT NULL,
    CONSTRAINT [PK_Type] PRIMARY KEY CLUSTERED ([tid] ASC),
)
Go
--Kho Hang
CREATE TABLE [Stock] (
    [batchid] nchar(20) NOT NULL,
    [importdate] date NULL,
    [amountofproduct] int NULL,
    CONSTRAINT [PK_Stock] PRIMARY KEY CLUSTERED ([batchid] ASC),
```

```

)
Go
--Nha San Xuat
CREATE TABLE [Manuafacture] (
    mid nchar(20) NOT NULL,
    mname nchar(40) NULL,
    CONSTRAINT [PK_Manuafacture] PRIMARY KEY CLUSTERED ([mid] ASC),
)
Go
--San Pham
CREATE TABLE [Product] (
    [pid] nchar(20) NOT NULL,
    [pname] nchar(30) NOT NULL,
    [pprice] float NOT NULL,
    [tid] nchar(20) NOT NULL,
    [unitname] nchar(20) NULL,
    [pimage] image NULL,
    CONSTRAINT [PK_Product] PRIMARY KEY CLUSTERED ([pid] ASC),
    CONSTRAINT [FK_Product_Type_tid] FOREIGN KEY ([tid]) REFERENCES [Type] ([tid]) ON UPDATE
    CASCADE
)
Go
--Hoa Don
CREATE TABLE [Invoice] (
    iid nchar(20) NOT NULL,
    eid nchar(20) NOT NULL,
    cid nchar(20) NULL,
    idate datetime NULL,
    itotalpay float NULL,
    CONSTRAINT [PK_Invoice] PRIMARY KEY CLUSTERED ([iid] ASC),
    CONSTRAINT [FK_Invoice_Employee_eid] FOREIGN KEY ([eid]) REFERENCES [Employee] ([eid]) ON
    UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT [FK_Invoice_Customer_cid] FOREIGN KEY ([cid]) REFERENCES [Customer] ([cid]) ON
    UPDATE CASCADE ON DELETE CASCADE
)
Go
--Chi Tiet Hoa Don
CREATE TABLE [Detail] (
    iid nchar(20) NOT NULL,
    pid nchar(20) NOT NULL,
    dprice float NULL,
    damount int NULL,
    CONSTRAINT [PK_Detail] PRIMARY KEY CLUSTERED ([iid] ASC, [pid]),
    CONSTRAINT [FK_Detail_Invoice_iid] FOREIGN KEY ([iid]) REFERENCES [Invoice] ([iid]) ON
    DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Detail_Product_pid] FOREIGN KEY ([pid]) REFERENCES [Product] ([pid]) ON
    DELETE CASCADE ON UPDATE CASCADE
)
Go
--Quan he 3 ngoi cung cap san pham
CREATE TABLE [Supply] (
    [pid] nchar(20) NOT NULL,
    [batchid] nchar(20) NOT NULL,
    [mid] nchar(20) NOT NULL,
    CONSTRAINT [PK_Supply] PRIMARY KEY CLUSTERED ([pid] ASC, [batchid], [mid]),
    CONSTRAINT [FK_Supply_Supply_batchid] FOREIGN KEY ([batchid]) REFERENCES [Stock]
    ([batchid]) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT [FK_Supply_Supply_mid] FOREIGN KEY ([mid]) REFERENCES [Manuafacture] ([mid]) ON
    DELETE CASCADE ON UPDATE CASCADE,

```

```

    CONSTRAINT [FK_Supply_Supply_pid] FOREIGN KEY ([pid]) REFERENCES [Product] ([pid]) ON
DELETE CASCADE ON UPDATE CASCADE
)
Go

```

3.2. Creating statements in Stored Procedure

3.2.1. Add account

```

CREATE PROC [dbo].[SP_AddAccount]
@user NCHAR(15),
@pass INT,
@type NCHAR(20),
@mail VARCHAR(100),
@status BIT
as
begin
INSERT INTO Account(ausername,apassword,eid,aemail,active)
VALUES(@user,@pass,@type,@mail,@status)
end Go

```

3.2.2. Add employee

```

Create Proc SP_AddEmployee
@id nchar(20),
@name nchar(40),
@birthdate date,
@gender nchar(15),
@phone nchar(15),
@address nchar(50),
@position nchar(40),
@salary float,
@image image
As
Begin
if(year(getdate())-year(@birthdate)>18)
begin
INSERT INTO Employee (eid, ename, ebirthdate, egender, ephone, eaddress, eposition, esalary,
eimage)
Values(@id,@name, @birthdate, @gender, @phone, @address, @position, @salary,@image)
end
else begin
Print('Invalid Age') Rollback TRANSACTION
end
End
Go

```

3.2.3. Update employee

```

Create Proc SP_UpdateEmployee
@id nchar(20),
@name nchar(40),
@birthdate date,
@gender nchar(15),
@phone nchar(15),
@address nchar(50),
@position nchar(40),
@salary float,
@image image
As
Begin

```

```

if(year(getdate())-year(@birthdate)>18)
begin
Update Employee SET ename=@name, ebirthdate=@birthdate, egender=@gender, ephone=@phone,
eaddress=@address, eposition=@position, esalary=@salary, eimage=@image WHERE Employee.eid=@id
end
else begin
print('Invalid Age') Rollback TRAN
end
End
Go

```

3.2.4. Delete employee

```

Create Proc SP_DeleteEmployee
@id nchar(20)
As
Begin Delete From Employee Where Employee.eid=@id
end
Go

```

3.2.5. List employee

```

Create Proc SP_ListEmployee
As
Begin
Select eid AS [Employee ID], ename AS [Name], ebirthdate AS [Birthdate], egender AS
[Gender], ephone AS [Phone], eaddress AS [Address], eposition AS [Position], esalary AS
[Salary], eimage AS [Image] From Employee
end
Go

```

3.2.6. Find employee

```

Create Proc SP_FindEmployee
@id nchar(20)
As
Begin
Select eid From Employee Where eid=@id
End
Go

```

3.2.7. Add customer

```

Create Proc SP_AddCustomer
@c nchar(20),
@name nchar(40),
@total float,
@phone nchar(15)
As
Begin
if(@phone like '0%' AND Len(@phone) = 10)
begin
INSERT INTO Customer (cid, cname, ctotalpay, cphone) Values (@c,@name,@total,@phone)
end
else begin
Print('Invalid Phone') Rollback TRANSACTION
end
End
Go

```

3.2.8. Add product

```

Create Proc SP_AddProduct
@id nchar(20),

```



```

@name nchar(30),
@price float,
@typeid nchar(20),
@unitname nchar(20),
@image image
As
Begin
INSERT INTO Product (pid, pname, pprice, tid,unitname,pimage) Values
(@id,@name,@price,@typeid,@unitname,@image)
End
Go

```

3.2.9. Update product

```

Create Proc SP_UpdateProduct
@id nchar(20),
@name nchar(30),
@price float,
@typeid nchar(20),
@unitname nchar(20),
@image image
As
Begin Update Product Set pname=@name,
pprice=@price,tid=@typeid,unitname=@unitname,pimage=@image Where pid=@id
End
go

```

3.2.10. Delete product

```

Create Proc SP_DeleteProduct
@id nchar(20)
As
Begin Delete From Product Where pid=@id
End
Go

```

3.2.11. Add invoice

```

Create Proc SP_AddInvoice
@i nchar(20),
@e nchar(20),
@c nchar(20),
@date datetime,
@total float
AS
Begin
INSERT INTO Invoice (iid, eid, cid, idate, itotalamount) VALUES (@i,@e, @c, @date, @total)
End
Go

```

3.2.12. Update invoice

```

Create Proc SP_UpdateInvoice
@i nchar(20),
@e nchar(20),
@c nchar(20),
@date datetime,
@total float
As
Begin Update Invoice Set eid=@e,cid=@c,idate=@date,itotalamount=@total WHERE iid=@i
End
Go

```

3.2.13. Delete invoice

```
Create Proc SP_DeleteInvoice
@i nchar(20)
As
Begin DELETE FROM Invoice WHERE iid =@i
End
Go
```

3.2.14. Add detail

```
Create Proc SP_AddDetail
@id nchar(20),
@pid nchar(20),
@price float,
@amount int
As
Begin INSERT INTO Detail (iid, pid, dprice, damount) VALUES (@id,@pid, @price, @amount)
End
Go
```

3.2.15. Update detail

```
Create Proc SP_UpdateDetail
@id nchar(20),
@pid nchar(20),
@price float,
@amount int
As
Begin Update Detail Set pid=@pid,dprice=@price,damount=@amount Where iid=@id
End
Go
```

3.2.16. Delete detail

```
Create Proc [dbo].[SP_DeleteDetail]
@id nchar(20),
@pid NCHAR(20)
AS
Begin Delete From Detail Where iid=@id AND pid =@pid
End
Go
```

3.2.17. Add manufacturer

```
Create Proc SP_AddManuafacture
@mid nchar(20),
@name nchar(40)
As
Begin INSERT INTO Manufacture (mid, mname) VALUES (@mid,@name)
End
Go
```

3.2.18. Update manufacturer

```
Create Proc SP_UpdateManuafacture
@mid nchar(20),
@name nchar(40)
AS
Begin Update Manufacture Set mname=@name Where mid=@mid
End
Go
```

3.2.19. Delete manufacturer

```

Create Proc SP_DeleteManuafacture
@mid nchar(20)
As
Begin Delete From Manuafacture Where mid=@mid
End
Go

```

3.2.20. Add stock

```

Create Proc SP_AddStock
@batid nchar(20),
@date date,
@amount int
As
Begin INSERT INTO Stock (batchid,importdate,amountofproduct) Values (@batid,@date,@amount)
End
Go

```

3.2.21. Update stock

```

Create Proc SP_UpdateStock
@batid nchar(20),
@date date,
@amount int
As
Begin Update Stock Set importdate=@date,amountofproduct=@amount Where batchid=@batid
End
Go

```

3.2.22. Delete stock

```

Create Proc SP_DeleteStock
@batid nchar(20)
As
Begin Delete From Stock Where batchid=@batid
End
Go

```

3.2.23. Add types

```

Create Proc SP_AddTypes
@tid nchar(20),
@name nchar(20)
As
Begin INSERT INTO [Type] (tid,tname) Values (@tid,@name)
End
Go

```

3.2.24. Update type

```

Create Proc SP_UpdateType
@tid nchar(20),
@name nchar(20)
As
Begin Update [Type] Set tname=@name Where tid=@tid
End
Go

```

3.2.25. Delete type

```

Create Proc SP_DeleteType
@tid nchar(20)
As
Begin Delete From [Type] Where tid=@tid
End

```


Go

3.3. Database connection

```
public class MY_DB
{
    //Phan quyen doi voi Manager
    SqlConnection con1 = new SqlConnection(@"Data Source=MSI;Initial
Catalog=StoreManagement;Persist Security Info=True;User
ID=managerAdmin;Password=1");
    public SqlConnection getConnectionStringManager
    {
        get
        {
            return con1;
        }
    }
    public void openConnectionStringManager()
    {
        if ((con1.State == ConnectionState.Closed))
        {
            con1.Open();
        }
    }
    public void closeConnectionStringManager()
    {
        if ((con1.State == ConnectionState.Closed))
        {
            con1.Close();
        }
    }
    //Phan quyen doi voi Employee
    SqlConnection con2 = new SqlConnection(@"Data Source=MSI;Initial
Catalog=StoreManagement;Persist Security Info=True;User
ID=employeeUser;Password=1");
    public SqlConnection getConnectionStringEmployee
    {
        get
        {
            return con2;
        }
    }
    public void openConnectionStringEmployee()
    {
        if ((con2.State == ConnectionState.Closed))
        {
            con2.Open();
        }
    }
    public void closeConnectionStringEmployee()
    {
        if ((con2.State == ConnectionState.Closed))
        {
            con2.Close();
        }
    }
}
```

3.4. Decentralization

3.4.1. Database table for authorization

Account	
	username
	apassword
	eid
	aemail
	active

3.4.2. Decentralized code on the database

```
Create Role employee
--View for employee
GRANT SELECT ON V_CheckAccount TO employee
GRANT SELECT ON V_Product TO employee
GRANT SELECT ON V_Customer TO employee
GRANT SELECT ON V_Invoice TO employee--Function
GRANT SELECT ON V_PersonalInfo TO employee
--Execute
GRANT EXECUTE ON SP_UpdateInvoice TO employee
GRANT EXECUTE ON SP_UpdateInvoiceNotCustomer TO employee
GRANT EXECUTE ON SP_AddInvoice TO employee
GRANT EXECUTE ON SP_AddCustomer TO employee
GRANT EXECUTE ON SP_AddDetail TO employee
GRANT EXECUTE ON SP_DeleteDetail TO employee
GRANT EXECUTE ON SP_UpdateCustomer TO employee
GO

USE StoreManagement
GO

Create ROLE manager
GRANT SELECT ON V_CheckAccount TO manager
GRANT SELECT ON V_Product TO manager
GRANT SELECT ON V_Customer TO manager
GRANT SELECT ON V_PersonalInfo TO manager
GRANT SELECT ON V_Invoice TO manager
GRANT EXECUTE ON SP_AddInvoice TO manager
GRANT EXECUTE ON SP_AddCustomer TO manager
GRANT EXECUTE ON SP_AddDetail TO manager
GRANT EXECUTE ON SP_UpdateCustomer TO manager
GRANT EXECUTE ON SP_AddEmployee TO manager

GRANT EXECUTE ON SP_AddManuaufacture TO manager
GRANT EXECUTE ON SP_AddProduct TO manager
GRANT EXECUTE ON SP_AddStock TO manager
GRANT EXECUTE ON SP_AddSupply TO manager
```

```

GRANT EXECUTE ON SP_AddTypes TO manager
GRANT EXECUTE ON SP_AddAccount TO manager
GRANT EXECUTE ON SP_DeleteCustomer TO manager
GRANT EXECUTE ON SP_DeleteDetail TO manager
GRANT EXECUTE ON SP_DeleteEmployee TO manager
GRANT EXECUTE ON SP_DeleteInvoice TO manager
GRANT EXECUTE ON SP_DeleteManuafacture TO manager
GRANT EXECUTE ON SP_DeleteProduct TO manager
GRANT EXECUTE ON SP_DeleteStock TO manager
GRANT EXECUTE ON SP_DeleteType TO manager
GRANT EXECUTE ON SP_FindEmployee TO manager
GRANT EXECUTE ON SP_ListEmployee TO manager
GRANT EXECUTE ON SP_UpdateCustomer TO manager
GRANT EXECUTE ON SP_UpdateDetail TO manager
GRANT EXECUTE ON SP_UpdateEmployee TO manager
GRANT EXECUTE ON SP_UpdateInvoice TO manager
GRANT EXECUTE ON SP_UpdateManuafacture TO manager
GRANT EXECUTE ON SP_UpdateProduct TO manager
GRANT EXECUTE ON SP_UpdateStock TO manager
GRANT EXECUTE ON SP_UpdateType TO manager
go

```

3.4.3. Decentralized code on C#

```

//Phan quyen tai khoan
SqlConnection con = new SqlConnection(@"Data Source=MSI;Initial
Catalog=StoreManagement;Persist Security Info=True;User ID=" + user +
";Password=" + pass);

```

3.5. Triggers and transactions

3.5.1. Settings to create login accounts according to user permissions

```

CREATE TRIGGER CreateUserLogin ON Account
FOR INSERT
AS
BEGIN
    DECLARE @user NCHAR(15), @password INT, @type NCHAR(20), @db_name NVARCHAR(MAX),
@active BIT
    SET @db_name = DB_NAME()
    SELECT @user = a.username, @password = a.password, @type = a.eid, @active = a.active
    FROM inserted

    EXEC('CREATE LOGIN [' + @user + '] WITH PASSWORD = ''' + @password + ''',
DEFAULT_DATABASE=[' + @db_name + ']')
    EXEC('CREATE USER [' + @user + '] FOR LOGIN [' + @user + ']')

    IF @type LIKE 'manager%'
    BEGIN
        EXEC sp_addrolemember 'db_owner', @user
        EXEC sp_addrolemember 'db_accessadmin', @user
        EXEC sp_addrolemember 'db_securityadmin', @user
        EXEC sp_addrolemember 'manager', @user
        EXEC('USE master; GRANT ALTER ANY LOGIN TO [' + @user + '] WITH GRANT OPTION')
    END
    ELSE IF @type LIKE 'employee%'
    BEGIN
        EXEC sp_addrolemember 'employee', @user
    END

```

```

END
IF @active = 0
    EXEC('ALTER LOGIN [' + @user + '] DISABLE')
ELSE
    EXEC('ALTER LOGIN [' + @user + '] ENABLE')
END
GO

```

3.5.2. Install login account updates according to user permissions

```

CREATE TRIGGER UpdateUserLogin ON Account
FOR UPDATE
AS
BEGIN
    DECLARE @old_active BIT, @new_active BIT, @old_user NCHAR(15), @new_user NCHAR(15),
    @old_password INT, @new_password INT, @type NCHAR(20), @db_name NVARCHAR(MAX)
    SET @db_name = DB_NAME()
    SELECT @old_user = username, @old_password = apassword, @old_active = active
    FROM deleted
    SELECT @new_user = username, @new_password = apassword, @type = eid, @new_active =
active
    FROM inserted

    IF (@new_user = @old_user AND @new_password = @old_password AND @old_active =
@new_active)
        RETURN
    ELSE IF (@new_user <> @old_user) --Quan Ly
        BEGIN
            EXEC('DROP USER [' + @old_user + ']')
            EXEC('DROP LOGIN [' + @old_user + ']')
            EXEC('CREATE LOGIN [' + @new_user + '] WITH PASSWORD = ''' + @new_password +
''', DEFAULT_DATABASE=[ ' + @db_name + ' ]')
            EXEC('CREATE USER [' + @new_user + '] FOR LOGIN [' + @new_user + ']')
            IF @type LIKE 'manager%'
                BEGIN
                    EXEC sp_addrolemember 'db_owner', @new_user
                    EXEC sp_addrolemember 'db_accessadmin', @new_user
                    EXEC sp_addrolemember 'db_securityadmin', @new_user
                    EXEC sp_addrolemember 'manager', @new_user
                    EXEC('USE master; GRANT ALTER ANY LOGIN TO [' + @new_user + '] WITH
GRANT OPTION')
                END
            ELSE IF @type LIKE 'employee%'
                BEGIN
                    EXEC sp_addrolemember 'employee', @new_user
                    EXEC('USE master; GRANT ALTER ANY LOGIN TO [' + @new_user + '] WITH
GRANT OPTION')
                END
            END
            ELSE IF (@new_password <> @old_password) --Ca 2
                BEGIN
                    EXEC('ALTER LOGIN [' + @new_user + '] WITH PASSWORD = ''' + @new_password +
''' OLD_PASSWORD = ''' + @old_password + ''''')
                END
            ELSE --Quan Ly
                BEGIN
                    IF @new_active = 0
                        EXEC('ALTER LOGIN [' + @new_user + '] DISABLE')
                    ELSE
                        EXEC('ALTER LOGIN [' + @new_user + '] ENABLE')
                END
            END
        END
    END

```

```

        END
    END
GO
CREATE TRIGGER DeleteUserLogin ON Account
FOR DELETE
AS
BEGIN
    DECLARE @user NCHAR(15)

    SELECT @user = username
    FROM deleted

    EXEC('DROP USER [' + @user + ']')
    EXEC('DROP LOGIN [' + @user + ']')
END
GO

```

3.5.3. Setting to delete login accounts according to user permissions

```

CREATE TRIGGER DeleteUserLogin ON Account
FOR DELETE
AS
BEGIN
    DECLARE @user NCHAR(15)

    SELECT @user = username
    FROM deleted

    EXEC('DROP USER [' + @user + ']')
    EXEC('DROP LOGIN [' + @user + ']')
END

```

3.5.4. Setting to update total pay for invoice

```

CREATE TRIGGER [dbo].[Update_TotalPay_Invoice] ON [dbo].[Detail]
AFTER INSERT
AS
BEGIN
    DECLARE @totalamount float, @iid nchar(20), @sum float
    Select @iid=iid From inserted
    SELECT @totalamount=damount*dprice FROM inserted
    Select @sum= Sum(itotalpay) From Invoice Where invoice.iid=@iid
    UPDATE Invoice SET itotalpay= @sum + @totalamount Where Invoice.iid=@iid
END
GO

```

3.5.5. Setting to decrease total pay for invoice

```

CREATE TRIGGER [dbo].[Decrease_TotalPay_Invoice] ON [dbo].[Detail]
AFTER DELETE
AS
BEGIN
    DECLARE @totalamount float
    SELECT @totalamount=damount*dprice FROM deleted
    UPDATE Invoice SET itotalpay= itotalpay - @totalamount
END
GO

```

3.5.6. Setting to check the invoice exists or not

```

CREATE TRIGGER [dbo].[CheckInvoice_Exist] ON [dbo].[Detail] INSTEAD OF INSERT
AS
BEGIN

```



```

Declare @tmp int, @tmp1 nchar(20)
Select @tmp1=iid from inserted
DECLARE @totalamount float
DECLARE @pid NCHAR(20), @dprice float, @damount int
SELECT @pid=pid, @dprice=dprice, @damount=damount FROM inserted
SELECT @totalamount = damount*dprice FROM inserted

If Exists( Select iid From Invoice Where iid=@tmp1)
begin

    if Exists(Select pid From Detail Where pid=@pid and iid=@tmp1)
    begin
        Update Detail Set damount=damount+@damount Where pid=@pid and iid=@tmp1
    end
    else
    begin
        INSERT INTO Detail(iid,pid,damount,dprice) VALUES(@tmp1,@pid,@damount,@dprice)
    end
End
Else
    BEGIN
        DECLARE @date datetime
        SET @date =getdate()
        Insert Into Invoice(iid,eid,ideate,itotalpay)
        Values(@tmp1,'manager1',@date,0)
        INSERT INTO Detail(iid,pid,damount,dprice)
        VALUES(@tmp1,@pid,@damount,@dprice)
    END
END
GO

```

3.5.7. Setting to update the invoice

```

CREATE TRIGGER [dbo].[UpdateInvoice] ON [dbo].[Detail]
AFTER Update
AS
BEGIN
    Declare @damount int ,@dprice float,@iid nchar(20),@pid nchar(20),@sum float
    Select @pid=pid,@iid=iid from inserted
    Select @damount=damount,@dprice=dprice,@iid=iid From Detail Where pid=@pid and
    iid=@iid

    Update Invoice Set itotalpay=0 Where Invoice.iid=@iid
    Select @sum= Sum(itotalpay) From Invoice Where invoice.iid=@iid
    UPDATE Invoice SET itotalpay= @sum + @damount*@dprice Where Invoice.iid=@iid
END

```

3.6. Function

3.6.1. Total number of employees

```

CREATE FUNCTION FN_CountEmployee()
RETURNS INT
AS
BEGIN
    DECLARE @SL INT
    SELECT @SL=Count(eid) FROM Employee
    RETURN @SL
END
GO

```

3.6.2. Total number of customers

```

CREATE FUNCTION FN_CountCustomer()
RETURNS INT
AS
BEGIN
    DECLARE @SLKH INT
    SELECT @SLKH=Count(cid) FROM Customer
    RETURN @SLKH
END
GO

```

3.6.3. Total number of products

```

CREATE FUNCTION FN_CountProducts()
RETURNS INT
AS
BEGIN
    DECLARE @SLSP INT
    SELECT @SLSP=Count(pid) FROM Product
    RETURN @SLSP
END
GO

```

3.6.4. Check account

```

CREATE FUNCTION V_CheckAccount(@user NCHAR(15),@pass INT)
RETURNS TABLE
AS
RETURN
(
    select * from Account where ausername = @user and apassword = @pass
)
GO

```

3.6.5. Find product by name and type

```

CREATE FUNCTION FN_FindProduct(@string VARCHAR(MAX))
RETURNS TABLE
AS
RETURN
(
    Select * FROM V_Product WHERE CONCAT([Product Name],[Type ID]) LIKE '%'+@string+'%'
)
GO

```

3.6.6. Find customer

```

CREATE FUNCTION FN_FindCustomer(@Search NCHAR(15))
RETURNS TABLE
AS
RETURN
(
    SELECT cid, cname FROM Customer WHERE cphone =@Search
)
GO

```

3.6.7. Show detail of invoice

```

Create Function FN_ShowDetail(@iid nchar(20))
returns Table
As
Return (
Select *From Detail Where iid=@iid
)
Go

```

3.6.8. Show detail

3.6.9. Auto show information of product

```
Create Function [dbo].[FN_AutoShowProductInfo](@id NCHAR(20))
Returns Table
As
Return
(
Select pname,pprice,unitname, pimage FROM Product WHERE pid=@id
)
GO
```

3.6.10. Total pay of invoice

```
CREATE FUNCTION [dbo].[FN_GetTotalPay](@iid NCHAR(20))
RETURNS float
AS
BEGIN
    DECLARE @total float
    SELECT @total=itotalpay FROM Invoice WHERE iid=@iid
    RETURN @total
END
GO
```

3.6.11. Search employees by id

```
CREATE FUNCTION [dbo].[FN_SearchEmployeeByID](@eid nchar(40) null)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM Employee
    WHERE eid =@eid
)
GO
```

3.6.12. Employee gender statistics

```
CREATE FUNCTION Fn_ListGender(@gen nchar(10))
RETURNS INT
AS
BEGIN
    DECLARE @SL INT
    SELECT @SL=Count(eid) FROM Employee Where egender=@gen
    RETURN @SL
END
Go
```

3.7. View

3.7.1. View for products

```
--Thong tin san pham
CREATE VIEW V_Product AS Select pid AS [Product ID],pname AS [Product Name],pprice AS
[Price], tid AS [Type ID], unitname AS [Unit], pimage AS [Image] From [Product]
GO
--Thong tin san pham (ngoai tru tid va unitname)
```

```
CREATE VIEW V_Product_Invoice AS Select pid AS [Product ID],pname AS [Product Name],pprice  
AS [Price], pimage AS [Image] From [Product]  
GO
```

3.7.2. View for customers

```
CREATE VIEW V_Customer AS SELECT cid AS [ID], cname as [Name], cphone as [Phone] FROM  
[Customer]  
go
```

3.7.3. View for type

```
CREATE VIEW V_Type AS SELECT * FROM [Type]  
GO
```

3.7.4. View for stock

```
CREATE VIEW V_Stock AS SELECT batchid AS [Batch ID], importdate AS [Import Date],  
amountofproduct [Amount Of Product] FROM [Stock]  
GO
```