SWE 207 DATABASE MANAGEMENT SYSTEMS PROJECT

Student 1:

Name: Meryem Berra

Surname: AVŞAR

Number: B201202053

E-mail: meryem.avsar@ogr.sakarya.edu.tr

Student 2:

Name: Hazal

Surname: TUĞRUL

Number: B201202048

E-mail: hazal.tugrul@ogr.sakarya.edu.tr

Student 3:

Name: Ayşe Nur

Surname: YILMAZ

Number: B201202002

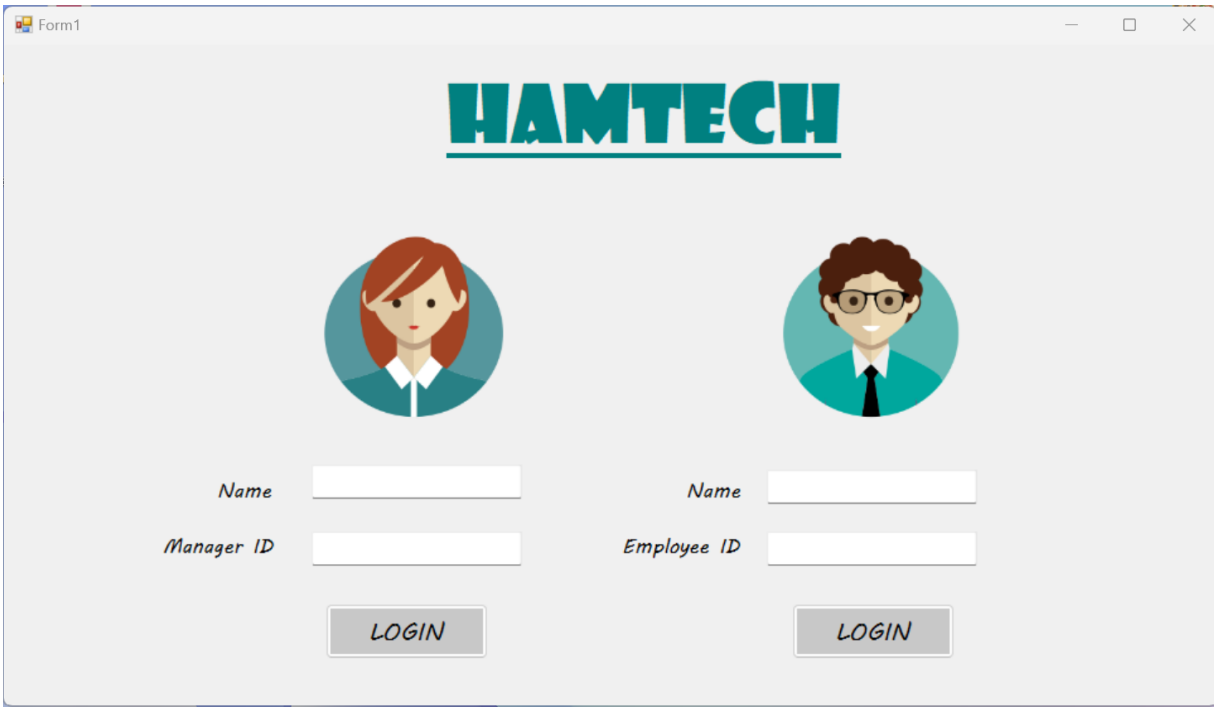
E-mail: ayse.yilmaz28@ogr.sakarya.edu.tr

PROJECT NAME : HAMTECH

1- Introduction of the problem

Our project, called HAMTECH, aims to provide a safe and effective way to manage the information of a large electronics company. This includes managing data on products, employees, and stores. Our application consists of two parts. One part allows employees to view and update product information and prices. The other part includes a system for selling products. Additionally, managers can access important information such as employee, supplier, and customer data all on the same platform, to effectively manage their company.

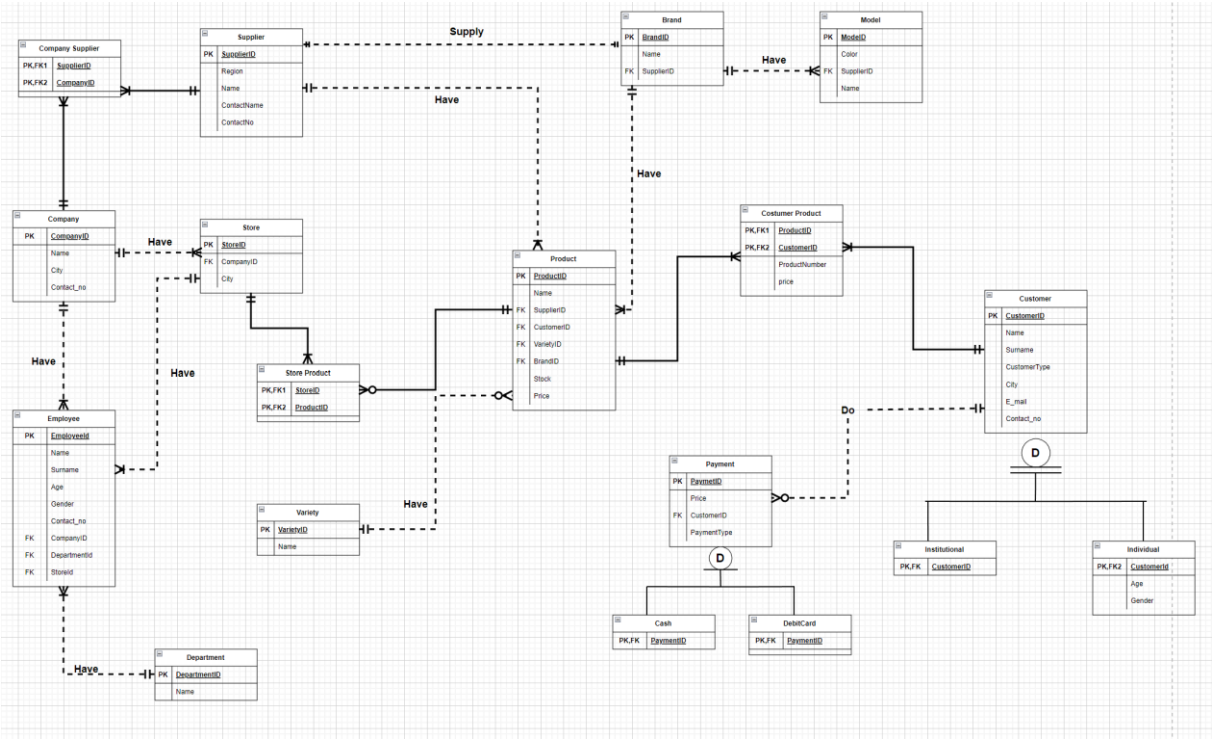
LOGIN PAGE:



2- Business Rules

- The company has a Federal Tax ID, Name, contact number. Our company is distinguished by Federal Tax ID from another companies.
- Each employee has an ID, name, surname, age, gender, contact number.
- Each department has a department ID, name.
- There is a store in every province and every store has an ID and city.
- Each Brand has an ID and Name.
- Each customer has an identity number, name, surname, type, city, e-mail and contact no.
- There are products in each store. Each product has an ID, name, price, stock, and price.
- Each Model has an ID, color, Name.
- Payment can take place two ways. Each payment has an ID, price, and type.
- Each Supplier has an ID, name, region, contact name, contact no
- Each Variety has an ID, Name
- A product can be sold in any or many stores. A store must sell at least one product or can sell many products.
- A product must be supplied by one supplier. A supplier can supply at least one or many products.
- A product can have only one variety and a variety can have any or many products.
- A product can have only one brand, but one brand can have one or many products.
- A brand can have one or many models. A model must have one brand.
- A customer must be under only one of two categories, individual or institutional. There cannot be any other way.
- A customer can buy any or many products. A product can be bought by zero or one customer.
- A customer can do any or many payments. A payment can be done only one customer.
- A payment can be done either with a debit card or cash, and there can be a payment other than debit card or cash.
- A supplier can supply only one brand and a brand can be supplied only one supplier.
- The company can have at least one or many employees. An employee has only one company.
- The company must have at least one store, but it can also have more than one store. A store has only one company.
- A company can have at least one or many suppliers. A supplier can have one or many company.
- Each employee can have in only one department at the same time. A department must have at least one employee, but there can also be more than one employee.
- An employee can have just one store. A store can have one or many employees.

3- Entity Relationship Model



4- Textual Representation

Brand (BrandID : Integer, Name : Character Varying, SupplierID: Integer)

Model (ModelID : Integer, Name : Character varying, SupplierID : Integer, Color : character varying)

Company (CompanyID : Integer, Name : Character varying , Contect_no : character varying)

Supplier (SupplierID : Integer, name : character varying, Contact_person: character varying, Contact_No : character varying, region : character varying)

Company Supplier (SuplierID : integer, CompanyID : integer)

Customer (CustomerID : integer, Name : character varying, Surname : character varying, CostumerType : character varying, City : character varying, E_mail : character varying, Contact_no : character varying)

Institutional (CostumerId : integer)

Individual (CostumerId : integer, Age : Integer, Gender : character varying)

Payment (PaymentID : integer, price : integer, CustomerID : integer, PaymentType : character varying)

Cash (PaymentID : integer)

DebitCard (PaymentID : integer)

Store (StoreID : integer, CompanyID : integer, City : character varying)

Store Product (StoreID : integer, ProductID : integer)

Product (ProductID : integer, VarietyID : integer, SupplierID : integer, CustomerID : integer, BrandID : integer, Name : character varying, Stock : Integer, Price : Integer)

Departmant (DepartmentID : integer, Name : character varying)

Employee (EmployeeID : integer, Name : character varying, Surname : character varying, Age : integer, Gender : character varying, Contact_no : character varying, CompanyID : integer, DepartmantID : integer, StoreID : integer)

Variety (VarietyID : integer, Name : character varyin

5- SQL Statements

```
CREATE SCHEMA payment;

CREATE TABLE payment.payment

(

    payment_id serial NOT NULL,

    customer_id serial NOT NULL,

    price integer NOT NULL,

    variety_id serial,

    payment_type character varying,

    PRIMARY KEY (payment_id)

);


ALTER TABLE payment.payment

    ADD CONSTRAINT "paymentFK" FOREIGN KEY ("customer_id")

    REFERENCES customer.customer ("customerID");
```

```
ALTER TABLE payment.payment
    ADD CONSTRAINT "paymentFK1" FOREIGN KEY ("variety_id")
    REFERENCES variety  ("variety_id");

CREATE TABLE payment.cash
(
    payment_id serial,
    PRIMARY KEY (payment_id)
);
ALTER TABLE payment.cash
    ADD CONSTRAINT "cashFK" FOREIGN KEY ("payment_id")
    REFERENCES payment.payment ("payment_id");

CREATE TABLE payment.debit_card
(
    payment_id serial,
    PRIMARY KEY (payment_id)
);
ALTER TABLE payment.debit_card
    ADD CONSTRAINT "debitFK" FOREIGN KEY ("payment_id")
    REFERENCES payment.payment ("payment_id");

CREATE SCHEMA customer;
CREATE TABLE customer.customer
(
    customer_id integer,
    name character varying ,
    surname character varying ,
```

```
    customer_type character varying,,
    city character varying,
    e_mail character varying;
   contact_no character varying;
    CONSTRAINT customer_pkey PRIMARY KEY (customer_id)
);

CREATE TABLE customer.individual
(
    customer_id serial NOT NULL,
    age integer,
    gender character varying,
    PRIMARY KEY (customer_id)
);

ALTER TABLE customer.individual
    ADD CONSTRAINT "individualFK" FOREIGN KEY ("customer_id")
    REFERENCES customer.customer("customer_id");

CREATE TABLE customer.institutional
(
    customer_id serial NOT NULL,
    PRIMARY KEY (customer_id)
);

ALTER TABLE customer.institutional
    ADD CONSTRAINT " institutionalFK" FOREIGN KEY ("customer_id")
    REFERENCES customer.customer("customer_id");
```

```
CREATE TABLE public.product
(
    product_id integer NOT;
    name character varying ,
    price integer,
    stock integer,
    supplier_id integer;
    variety_id integer,
    CONSTRAINT product_pkey PRIMARY KEY (product_id)
);

ALTER TABLE product
    ADD CONSTRAINT " product FK" FOREIGN KEY ("supplier_id")
    REFERENCES supplier("supplier_id");

ALTER TABLE product
    ADD CONSTRAINT " product FK2" FOREIGN KEY ("brand_id")
    REFERENCES brand("brand_id");

ALTER TABLE product
    ADD CONSTRAINT " product FK3" FOREIGN KEY ("variety_id")
    REFERENCES variety ("variety _id");

CREATE TABLE brand
(
    brand_id integer,
    name character varying ,
    supplier_id integer
);
```

```
ALTER TABLE brand
    ADD CONSTRAINT " brandFK" FOREIGN KEY ("supplier_id")
    REFERENCES supplier("supplier_id");

CREATE TABLE company
(
    company_id serial NOT NULL,
    name character varying,
    contact_no character varying,
    city character varying,
    PRIMARY KEY (company_id)
);

CREATE TABLE department
(
    department_id serial NOT NULL,
    name character varying,
    PRIMARY KEY (department_id)
);

CREATE TABLE employee
(
    employee_id serial NOT NULL,
    name character varying,
    surname character varying,
    age integer,
    gender character varying,
    contact_no character varying,
    company_id serial ,
```

```
    store_id serial  ,
    department_id serial,
    salary integer,
    PRIMARY KEY (employee_id)
);

ALTER TABLE employee
    ADD CONSTRAINT " employeeFK" FOREIGN KEY ("company_id")
    REFERENCES company("compyany_id");

ALTER TABLE employee
    ADD CONSTRAINT " employeeFK2" FOREIGN KEY ("store_id")
    REFERENCES store ("store _id");

ALTER TABLE employee
    ADD CONSTRAINT " employeeFK3" FOREIGN KEY ("department_id")
    REFERENCES department ("department _id");


CREATE TABLE model
(
    model_id serial NOT NULL,
    name character varying,
    color character varying,
    supplier_id integer,
    PRIMARY KEY (model_id)
);
```

```
ALTER TABLE model
    ADD CONSTRAINT " modelFK" FOREIGN KEY ("supplier_id")
    REFERENCES supplier("supplier_id");



CREATE TABLE public.store
(
    store_id serial NOT NULL,
    city character varying,
    company_id serial,
    PRIMARY KEY (store_id)
);

ALTER TABLE store
    ADD CONSTRAINT " storeFK" FOREIGN KEY ("company_id")
    REFERENCES company (“company_id");

CREATE TABLE public.supplier
(
    supplier_id serial NOT NULL,
    name character varying,
    region character varying,
    contact_no character varying,
    contact_person character varying,
    PRIMARY KEY (supplier_id)
);
```

```
CREATE TABLE public.variety
(
    variety_id serial NOT NULL,
    name character varying,
    PRIMARY KEY (variety_id)
);
```

6-Stored procedure

1)

```
CREATE OR REPLACE FUNCTION public.add_tax(payment_type text, ucret integer, adet integer)
 RETURNS integer
 LANGUAGE plpgsql
AS $function$
BEGIN
 IF payment_type = 'debit' THEN
  RETURN (ucret + (ucret * 0.1))*adet;
  ELSE
    RETURN ucret*adet;
  END IF;
END;
$function$
```

2)

```
 CREATE OR REPLACE FUNCTION public.apply_discount()
 RETURNS TABLE(_product_id integer, namee character varying, pricee integer, stockk integer)
 LANGUAGE plpgsql
AS $function$
```

```
BEGIN
  UPDATE product SET price = price * 0.9;
  RETURN Query
  Select "product_id","name","price","stock" from product;
END;
$function$
```

3)

```
 CREATE OR REPLACE FUNCTION public.login(_employee_id integer, _name character varying)
 RETURNS integer
 LANGUAGE plpgsql
AS $function$
BEGIN
   if (SELECT count(*) from public.employee where "name"=_name
and"employee_id"=_employee_id)>0 then
      return 1;
   else
      return 0;
   end if;
end
   $function$
```

4)

```
CREATE OR REPLACE FUNCTION public.searchproduct(ename text)
 RETURNS TABLE(_productid integer, _productname character varying, _price integer, _stock
integer)
 LANGUAGE plpgsql
AS $function$
```

```
BEGIN
    RETURN QUERY SELECT "product_id","name","price","stock" FROM product
                WHERE "name"=ename;
END;
$function$
```

5)

```
CREATE OR REPLACE FUNCTION public.sort_products_by_price()
 RETURNS TABLE(naame character varying, ppprice integer, ssstock integer)
 LANGUAGE plpgsql
AS $function$
BEGIN
  RETURN QUERY
    SELECT "name",price,stock
    FROM product
    ORDER BY price ASC;
END;
$function$
```

7-Trigger

1)

```
CREATE TABLE stock (
  name TEXT PRIMARY KEY,
  total_stock INTEGER
);

CREATE OR REPLACE FUNCTION update_stock_table()
RETURNS TRIGGER AS $$
```

```
BEGIN
  IF NOT EXISTS (SELECT 1 FROM stock WHERE name = NEW.name) THEN
    INSERT INTO stock (name, total_stock) VALUES (NEW.name, NEW.stock);
  ELSE
    UPDATE stock SET total_stock = total_stock + NEW.stock WHERE name = NEW.name;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_stock_table
AFTER INSERT ON products
FOR EACH ROW
EXECUTE PROCEDURE update_stock_table();
```

2)

```
create TABLE hire_info(
"info" char VARYING(150)
);
 CREATE OR REPLACE FUNCTION public.update_hire_info()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
BEGIN
  IF NEW.id > 10 THEN
     Update hire_info Set info ='Sıkıntı yok';
  ELSE
  Update hire_info SET info='İşçi al';
  END IF;
```

```
  RETURN NULL;
END;
$function$

CREATE TRIGGER update_hire_info
AFTER UPDATE ON public.employee_count
FOR EACH ROW EXECUTE FUNCTION update_hire_info()
```

3)

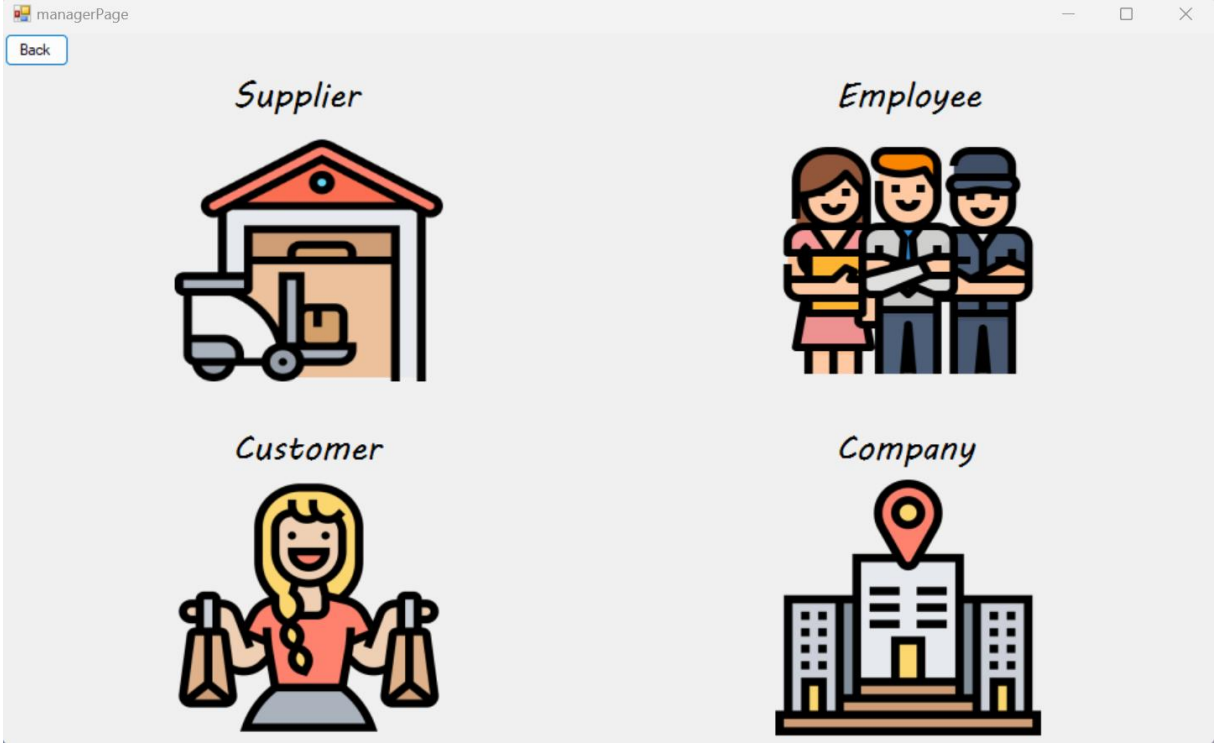
```
create table employee_count(
id serial PRIMARY KEY,
  status text
);
INSERT INTO employee_count (id)
SELECT COUNT(*) FROM employee;

CREATE OR REPLACE FUNCTION update_employee_count()
RETURNS TRIGGER AS $$
BEGIN
  -- Update the count column in the employee_count table
  UPDATE employee_count SET id = (SELECT COUNT(*) FROM employee);

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_employee_count_trigger
AFTER INSERT OR DELETE ON employee
FOR EACH ROW EXECUTE PROCEDURE update_employee_count();
```

4)

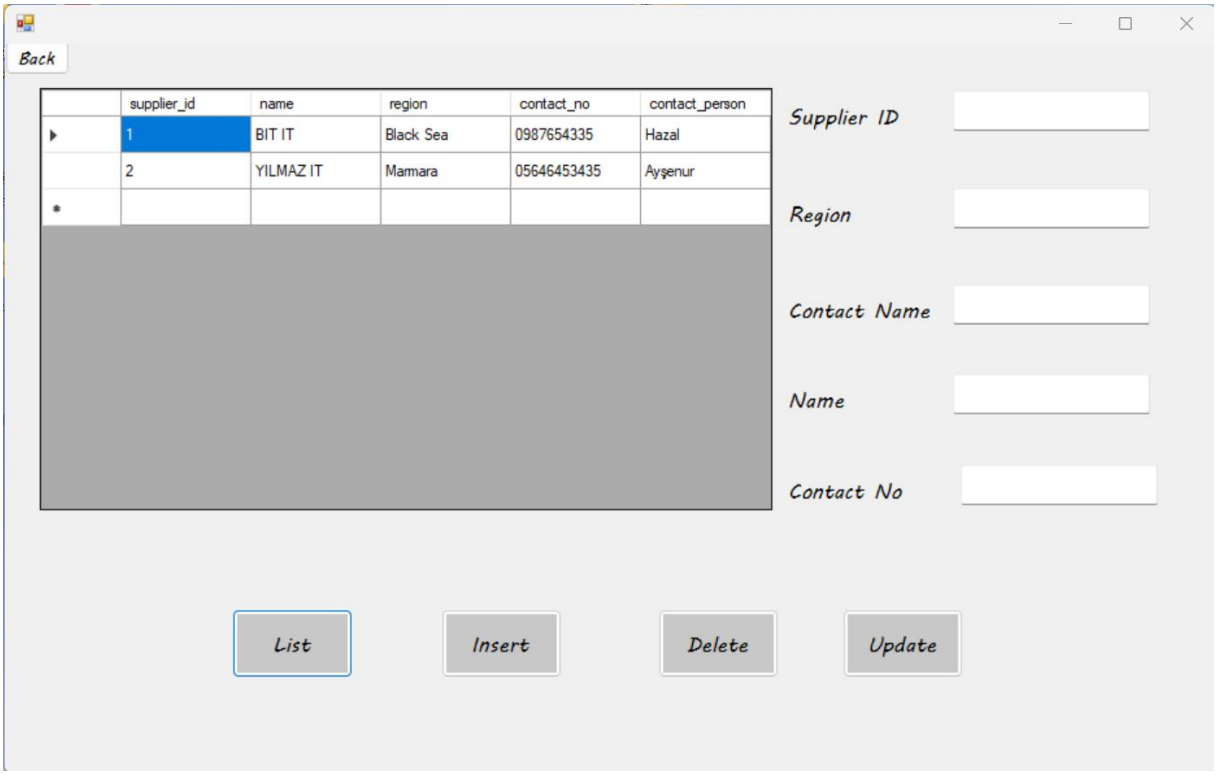
```
CREATE TABLE low_product
(
    productID integer ,
    "name" character varying,
    price integer,
    stock integer,
    CONSTRAINT lowproduct_pkey PRIMARY KEY (productID)
);

CREATE OR REPLACE FUNCTION public.create_low_stock_record()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
BEGIN
  IF NEW.stock < 15 THEN
    INSERT INTO low_product (product_id, name, stock, price)
    VALUES (NEW.product_id, NEW.name, NEW.stock, NEW.price);
  END IF;
  RETURN NULL;
END;
$function$

CREATE TRIGGER create_low_stock_record
AFTER INSERT ON product.product
FOR EACH ROW
EXECUTE PROCEDURE create_low_stock_record();
```

8-OPERATIONS

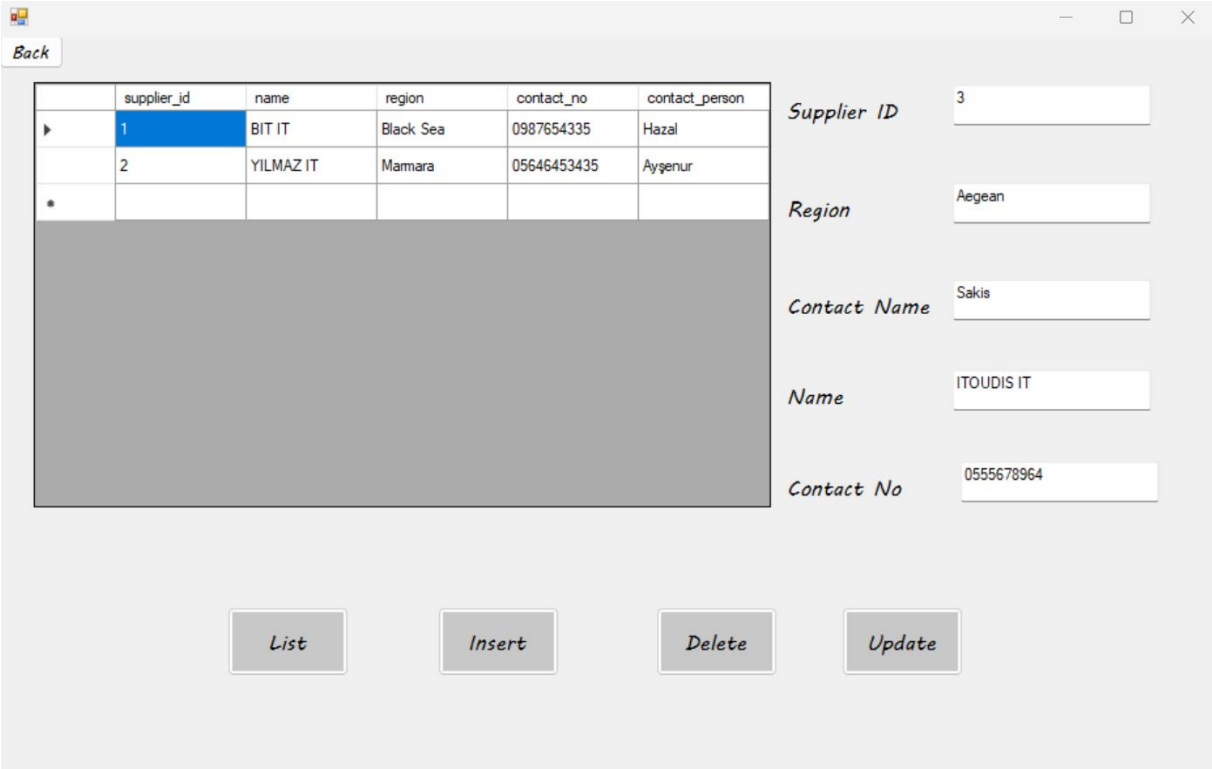
- After manager login the system, this page is shown.

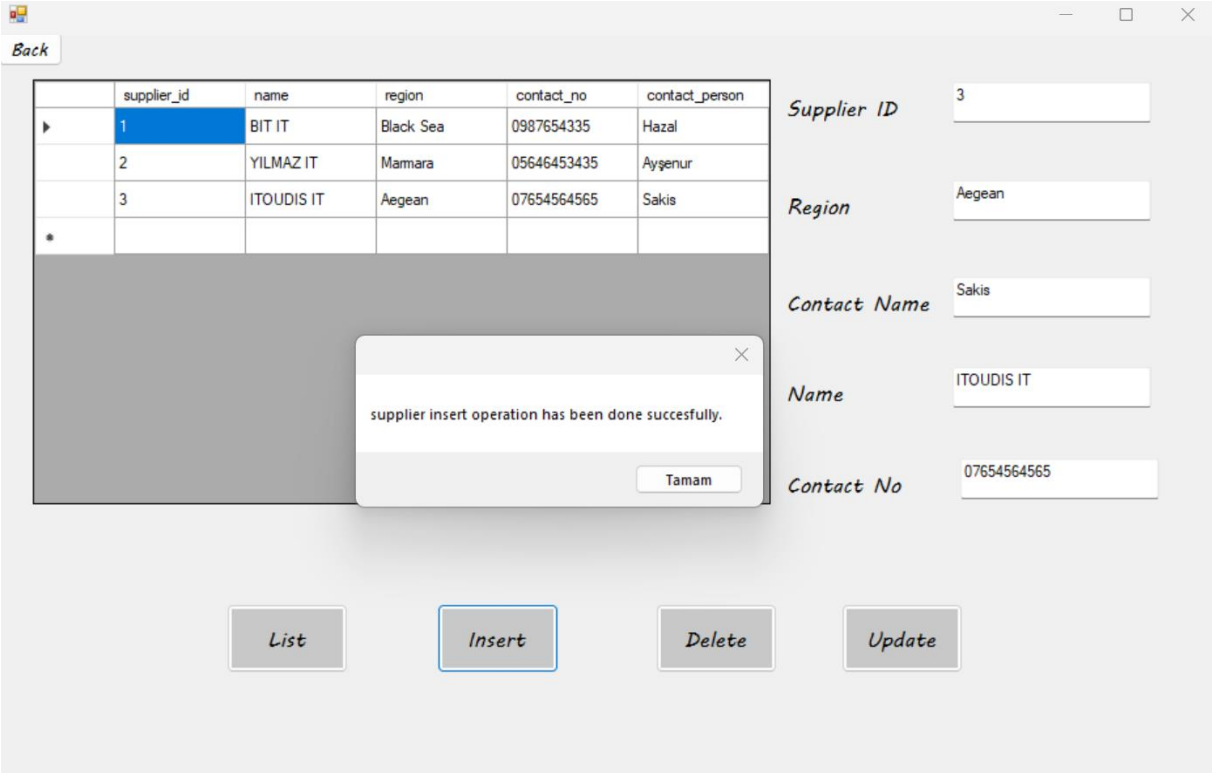


- List Operation for manager.

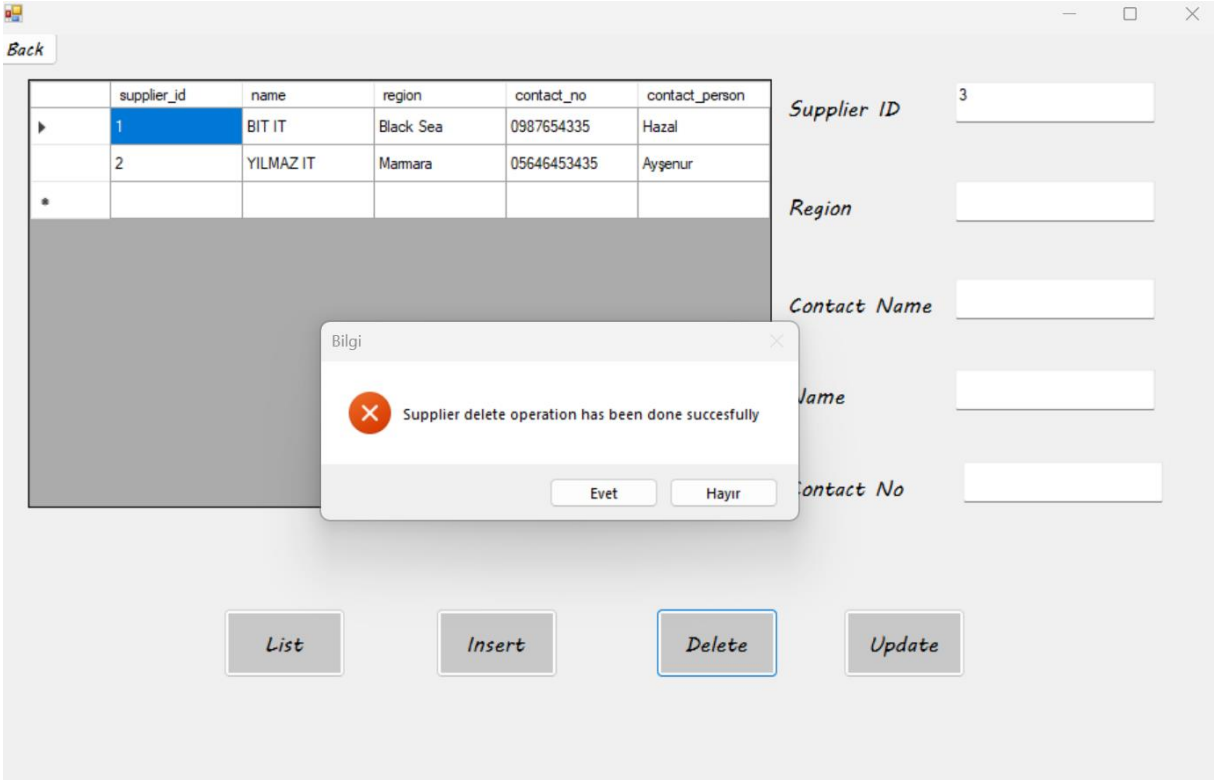


- Insert Operation

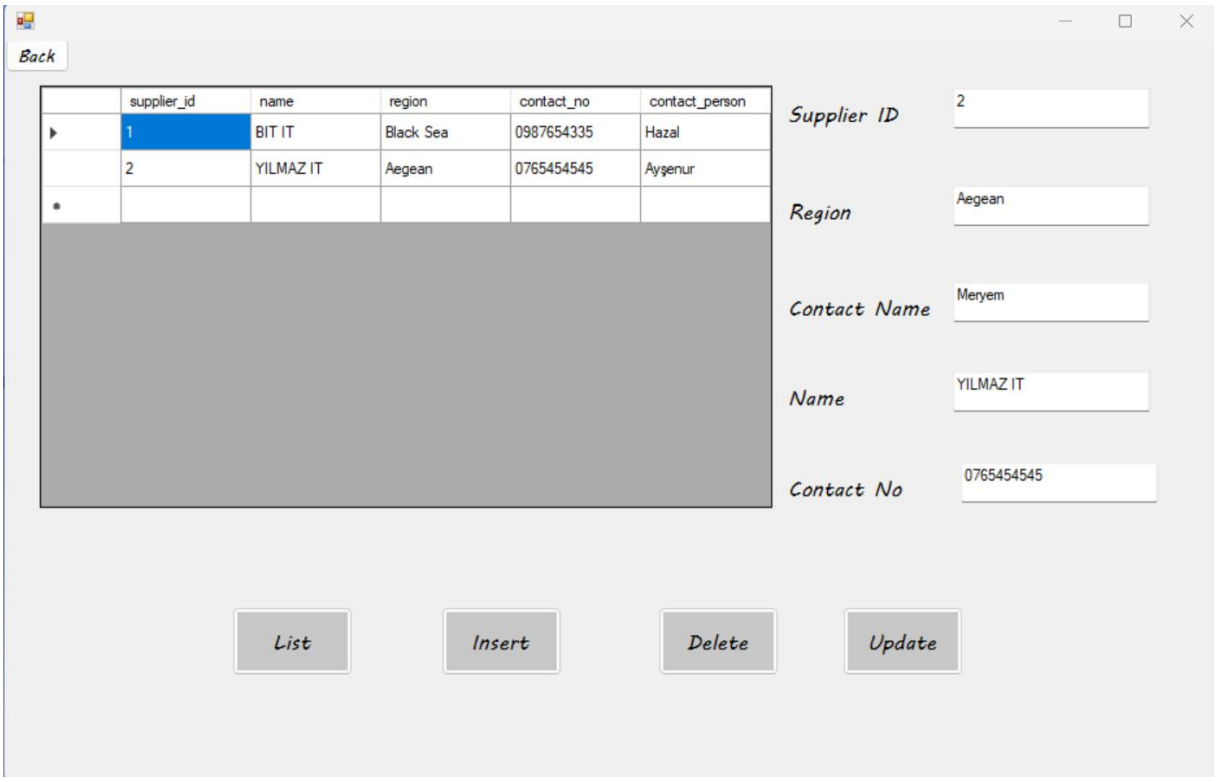


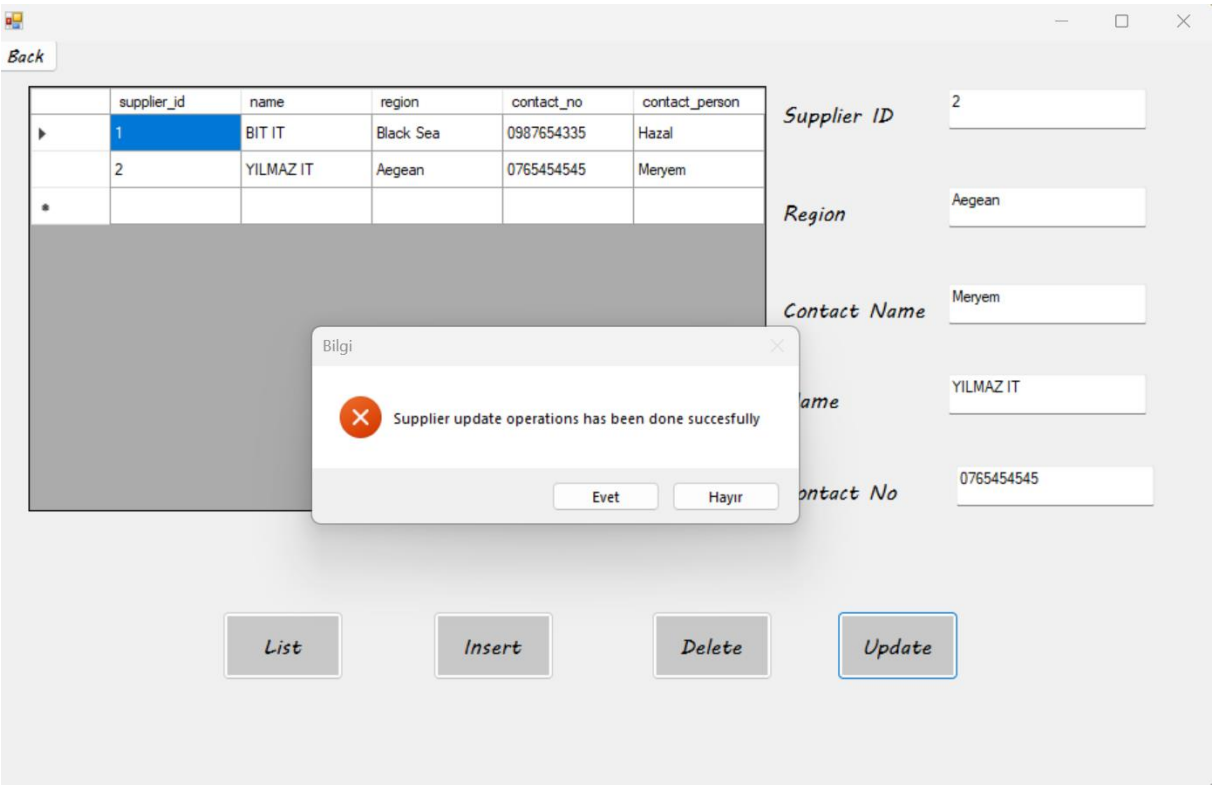


- Delete Operation

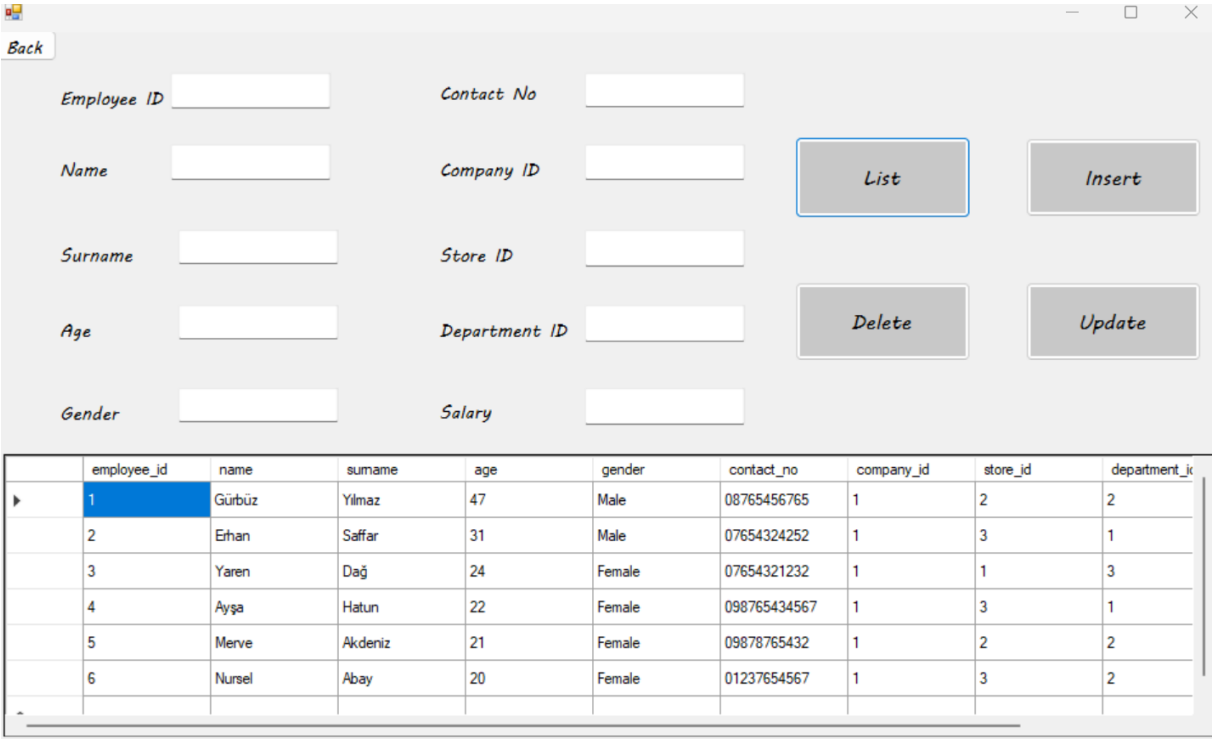


- Update Operation



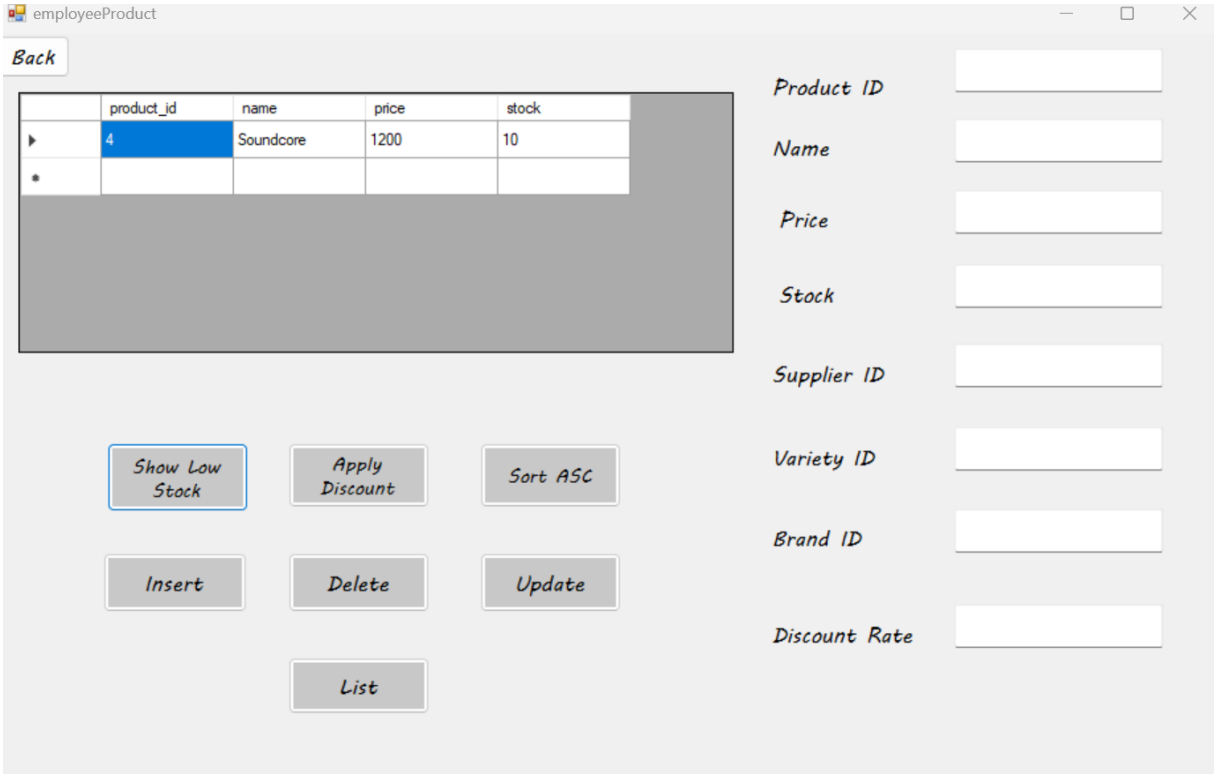


- List Operation for manager to manage the employee information.

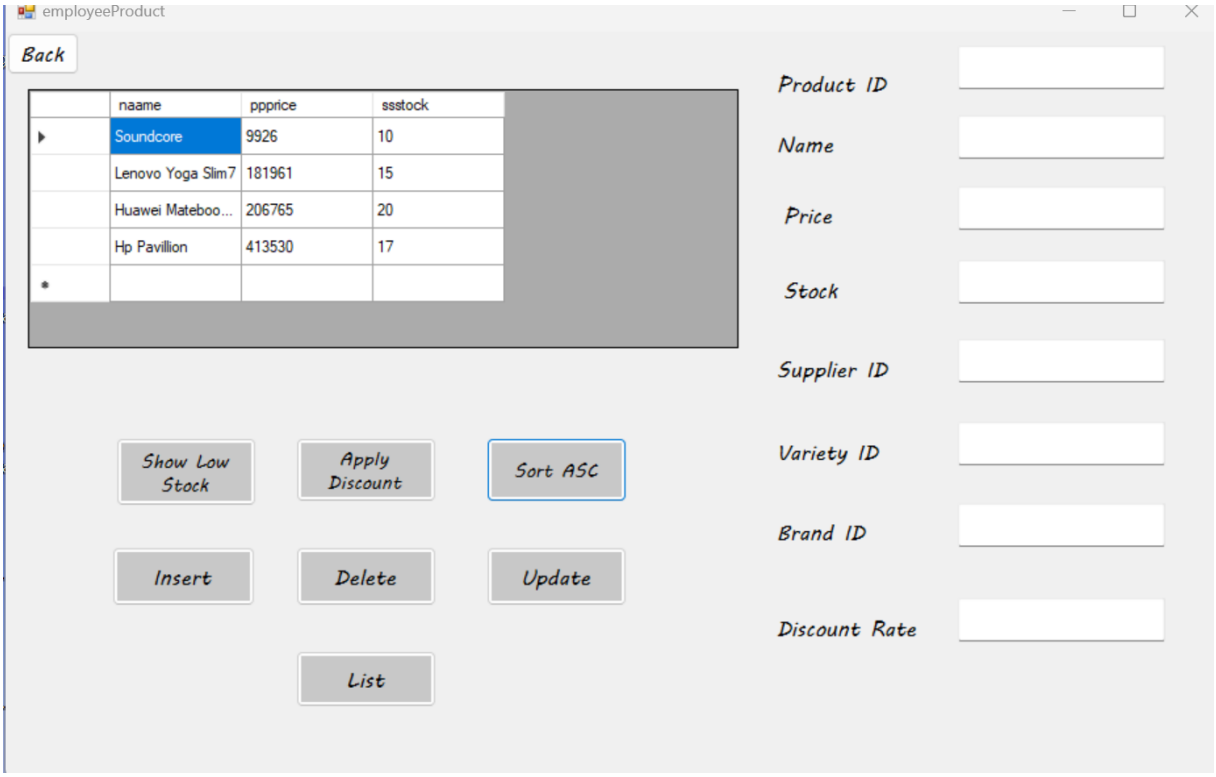


- After employee login this page is shown.

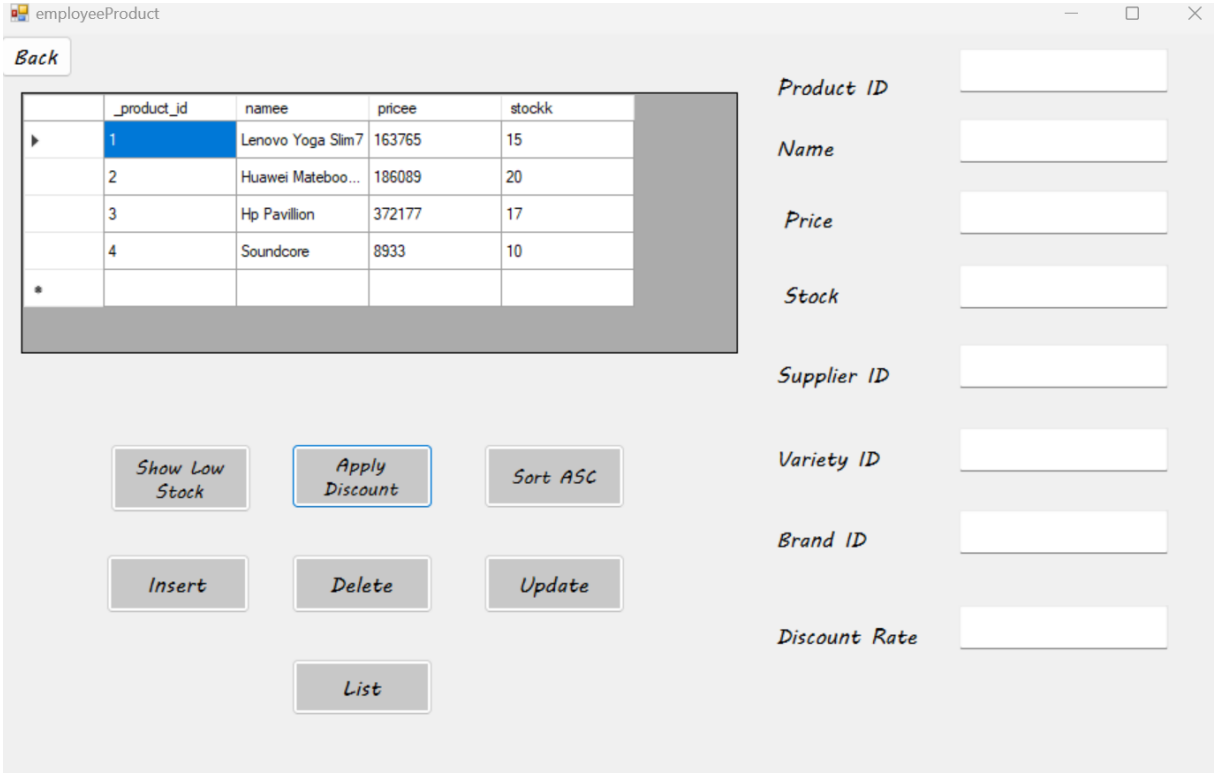
- If employee click the product section, this page is shown. Employee can display the low stock information of product.



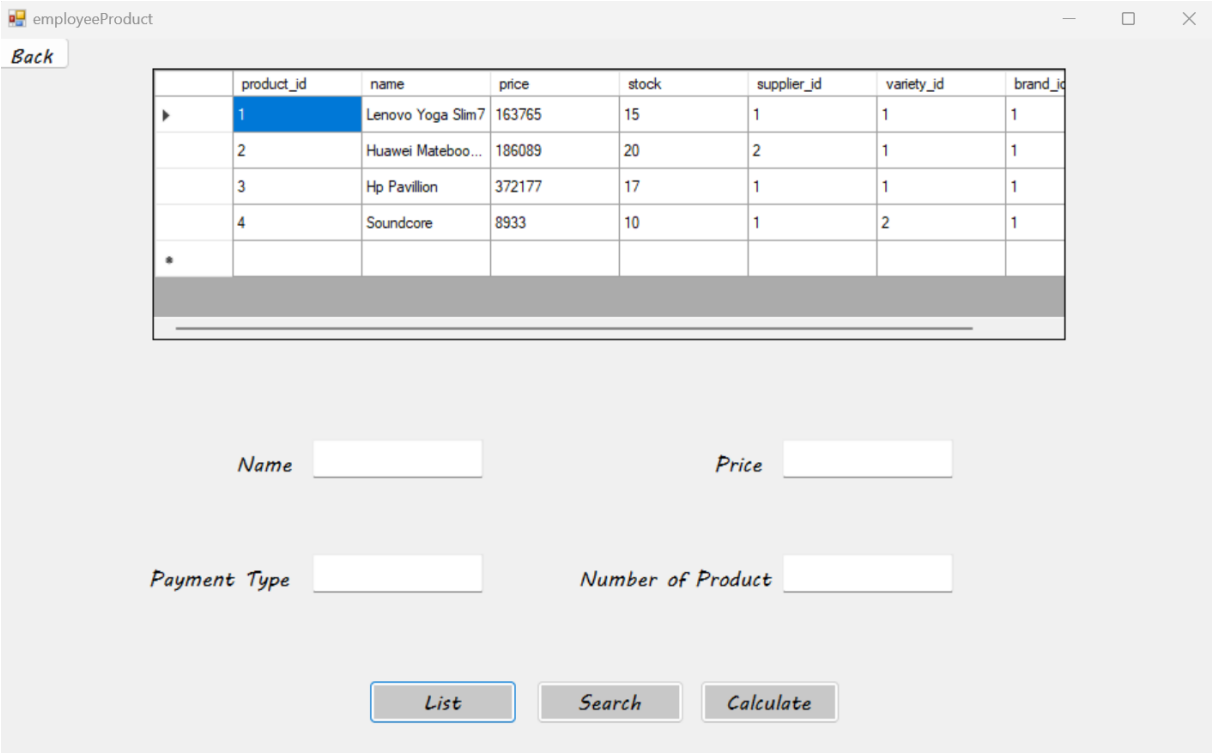
- Employee can sort the product by price ascending.



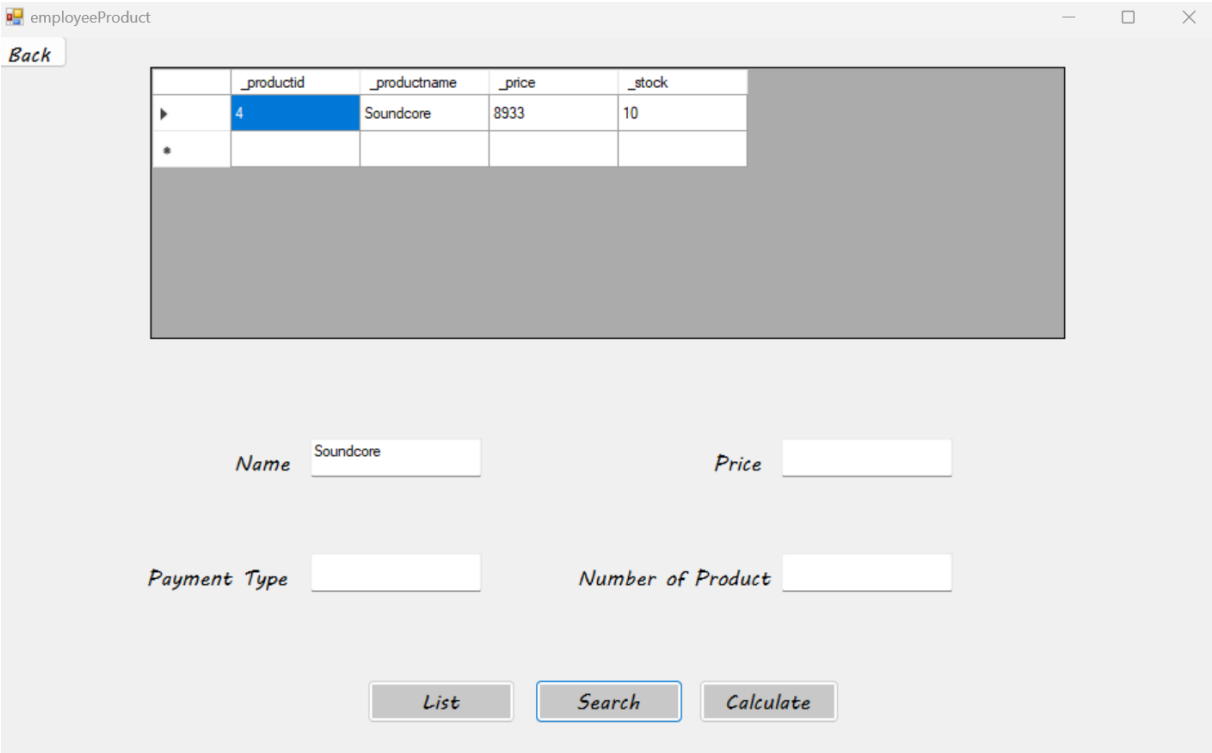
- Employee can apply discount on products.



- If employee click the payment section, this page is shown.



- Employee can search according to name of products.



- Employee can calculate the total price of products according to number of products and payment type. If customer prefers the payment with debit card, total price is calculated with %10 amount of tax.