FINAL PROJECT

Fresh Supermarket Management

IS431 - Visual Programming

Lecturer: Agus Sulaiman, S.Kom, M.M

Authored by Group 5 :

Christopher Abie Diaz Doviano	(NIM: 00000067692)
Ray Anthony Pranoto	(NIM: 00000066655)
Nicholas Alven	(NIM: 00000066511)
Yoga Saputra	(NIM: 00000065491)
Adryel Ethantyo	(NIM: 00000068165)

INFORMATION SYSTEMS STUDY PROGRAM

FACULTY OF ENGINEERING AND INFORMATICS

MULTIMEDIA NUSANTARA UNIVERSITY

TANGERANG

2023

CHAPTER I
INTRODUCTION

1.1 Background of the Research

In the retail industry, supermarkets are one form of business that plays a crucial role in meeting the daily needs of consumers. Supermarkets provide a wide range of products, from food, beverages, hygiene products, to other daily necessities. To maintain competitiveness in an increasingly competitive market, supermarket management must be able to efficiently manage inventory, monitor sales, and maintain operational efficiency. In recent decades, the development of information technology has had a significant impact on the retail industry, including supermarket management. Computer-based management systems are an effective solution to overcome the challenges of managing supermarkets. An integrated and efficient supermarket management application allows owners or managers to have better visibility of product stock, update product information quickly, manage sellers, and track and analyze sales.

One of the main objectives of supermarket management is to keep products in stock to meet customer demand. By using a supermarket management app, owners or managers can monitor and manage inventory more efficiently, thereby avoiding stock shortages that can lead to losses or customer disappointment. This application also allows the owner or manager to plan product purchases more accurately based on analysis of previous sales data In addition, the supermarket management application also provides convenience in updating product information in real-time. Owners or managers can easily change prices, update stock availability, and change product descriptions without having to make manual changes at the physical store. This helps in maintaining accurate product information and providing a better customer experience.

In their daily operations, supermarkets also employ salespeople or cashiers who are responsible for serving customers and conducting sales transactions. Supermarket management applications allow managers to manage vending data more efficiently, such as recording vending information, updating contact data, and managing user access rights. This helps in optimizing vendor management and ensuring a smooth transaction process in the supermarket. In the increasingly advanced digital era, sales data plays an important role in making smart business decisions. Supermarket management apps can record and track daily sales, provide detailed sales reports, and analyze sales trends. This sales data analysis can provide valuable insights to the owner or manager in identifying best-selling products, adjusting marketing strategies, and optimizing store performance.

1.2 Problem Identification

1. Often the inventory items to be distributed do not have an identification mark, so that the goods sent are difficult to track their whereabouts and are not guaranteed safety.
2. Problems experienced by warehouse workers when experiencing loss of goods
3. There are a lot of human errors when recording stock data when goods enter and exit (input output).
4. A lot of time is wasted just to record incoming and outgoing stock (connecting it with various divisions).

1.3 Objective of the Research

1. Create an application that facilitates business processes in trading companies
2. Increase the productivity of warehouse workers in handling their stock-related work
3. Minimize human errors during input and output for distribution activities
4. Increase the efficiency of employees' work
5. Make it easier for owners or managers to manage product inventory quickly and efficiently.
6. Update real-time product information, such as price, stock, and product description.
7. Improve operational efficiency by automating the sales and receipt generation process.
8. Optimize seller management with the ability to add, edit, and update seller information.
9. Provides the ability to manage product categories and group products according to relevant categories.
10. Record and track daily sales to analyze store performance, identify sales trends, and make better business decisions.

CHAPTER 2 PROCESS AND CONCEPTING

2.1 Concept of “Fresh Supermarket Management”

Supermarket management includes various activities to optimize supermarket operations. An important part of supermarket management is inventory management. This includes tracking and managing product inventory, purchasing, inventory counting, and inventory updates. Effective inventory management ensures sufficient product availability to meet customer demand, avoids inventory shortages that can lead to lost sales, and avoids overstocking that can lead to unnecessary storage costs.

Sales monitoring is also an important part of supermarket management. It involves collecting and analyzing sales data such as daily, weekly, or monthly sales volume, sales trends for specific products, and customer buying behavior. Sales monitoring helps understand customer preferences, identify the most requested products, and optimize sales strategies. In addition, sales management is also an important part of supermarket management. Sales management includes hiring, training, assigning, and evaluating sales performance. It includes salesperson information such as personal information, contact information, working hours and costs. Effective salesperson management helps to ensure adequate salesperson presence for customer service, monitor salesperson performance, and organize appropriate incentives and rewards.

Finally, product category management is also an important part of supermarket management. Product groups are groupings of products based on certain characteristics or types, such as: B. Food, fresh products, daily necessities, etc. Product category management includes inventory control, product placement on shelves, price management, campaigns, and marketing strategies for specific categories. Effective product category management helps increase product visibility, make it easier for customers to find products, and streamline sales based on customer preferences and needs.

2.2 Business Process

1. Login Process:
   - The user opens the application and sees the splash form.
   - After the splash form is displayed, the user is directed to the login page.
   - The user enters their ID and password.
   - The application validates the login information against the Sellers table in the database.
   - If the user is a Seller and the login information is valid, the user is directed to the Selling Form.

- If the user is an Admin and the login information is valid, the user is directed to the Admin Form.

2. Selling Process (Selling Form):
   - The Seller selects the products they want to sell from the available product list.
   - The Seller enters the quantity of the products to be sold.
   - The application calculates the subtotal based on the product price and quantity sold.
   - The Seller can add other products to the sales list.
   - After entering all the products, the Seller can print a bill that includes the product details, subtotal, and total.

3. Product Management Process (Admin Product Form):
   - The Admin can view the list of available products.
   - The Admin can add new products to the system by entering information such as product name, price, and stock.
   - The Admin can edit existing product information, such as changing the price or stock.
   - The Admin can delete products that are no longer available.

4. Seller Management Process (Admin Seller Management Form):
   - The Admin can view the list of registered Sellers.
   - The Admin can add new Sellers by entering information such as name, ID, and password.
   - The Admin can edit Seller information, such as changing the password.
   - The Admin can delete Sellers from the system if needed.

5. Category Management Process (Admin Category Management Form):
   - The Admin can view the list of categories.
   - The Admin can add new categories by entering the category name.
   - The Admin can edit category information, such as changing the category name.
   - The Admin can delete categories if they are no longer needed.

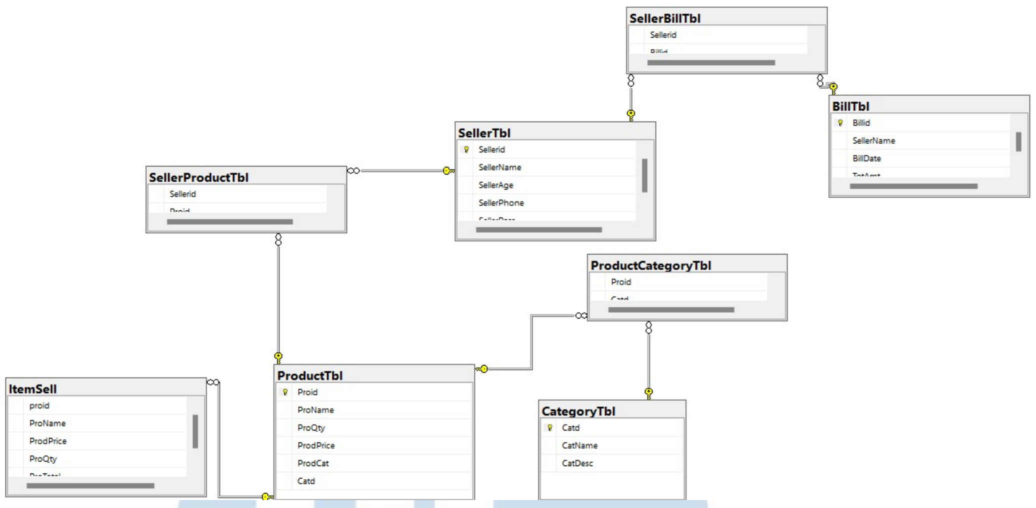
6. Sales Reporting Process (Admin Sales Report Form):
   - The Admin can select a specific date to view the sales report.
   - The application retrieves sales data from the Sales table based on the selected date.
   - The Admin can view sales details such as the list of products sold, quantities, and total sales.

2.3 Features

1. User Authentication: The application will provide a login feature that allows users to access the system according to their role and access rights, i.e. as Admin or Seller.
2. Product Management: Admins will have full access to add, edit, and delete products from the database. They can update product information such as price, stock, description and other attributes.
3. Product Sales: This feature will allow Sellers to conduct sales transactions quickly and efficiently. They can select products to sell, set quantities, and generate purchase receipts for customers.
4. Sales Recording: The application will record every sales transaction made by the Seller. Sales data will be stored in the database for further analysis.
5. Seller Management: Admins will be able to manage seller information, such as adding, editing, or deleting seller data. They can also set access rights and other settings related to sellers.
6. Category Management: Admins can manage product categories, add new categories, edit existing categories, or delete unnecessary categories. This feature will help in organizing and grouping products according to relevant categories.
7. Sales Report: The app will provide a feature to generate daily sales reports or sales reports for a specific time range. Admin can
8. Total Sold Report: The application provides a feature that can be used by sellers to print total sales per customer.

2.4 Database Design

2.4.1 Table Relation



Picture 2.1 Table Relation

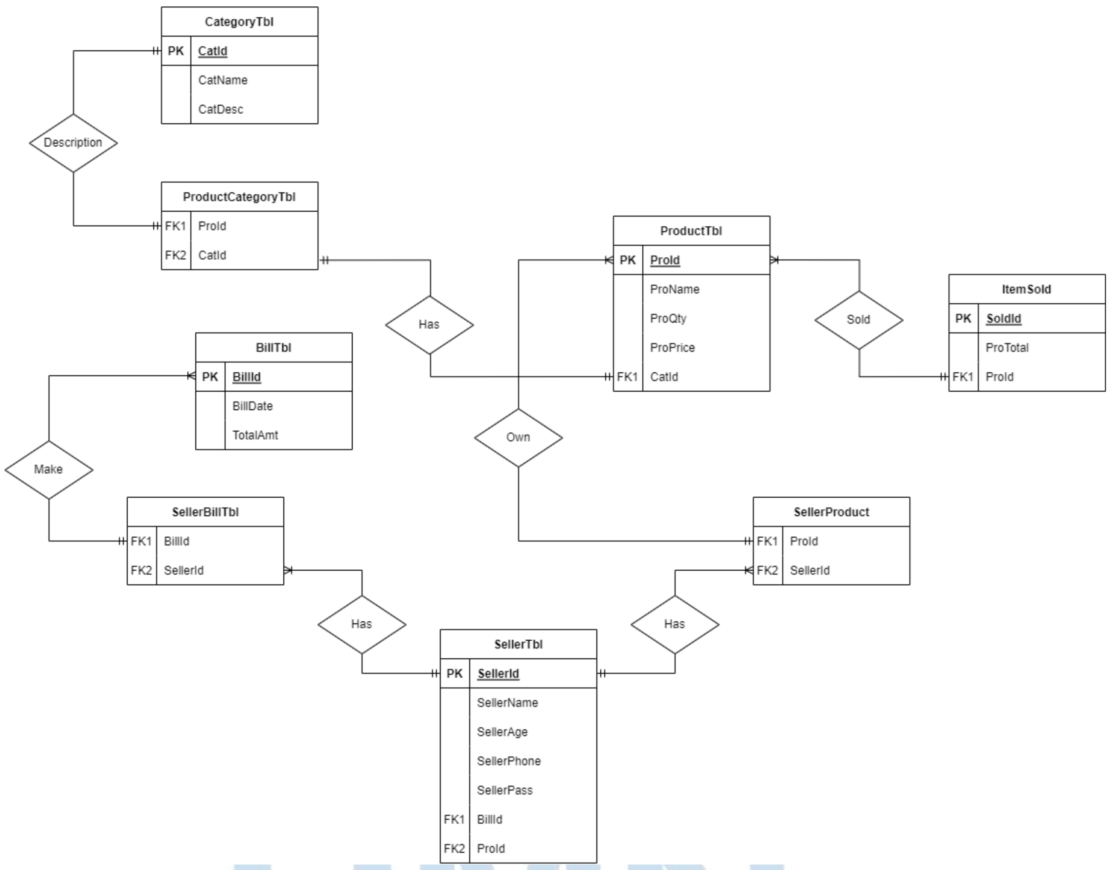
The picture above is a relationship table in the Fresh Shopping Management application which consists of 8 tables. In the main / parent table, there is a SellerTbl table that contains Sellerid, SellerName, SellerAge, SellerPhone, SellerPass data. Then, on the right side of the SellerTbl table has a relationship with SellerBillTbl which is useful as a merger of the SellerTbl table with BillTbl which contains a foreign key that references the primary key in SellerTbl and BillTbl. BillTbl is a table that serves to show the costs that must be paid by buyers / customers when buying goods, containing data such as the following; Billid, SellerName, BillDate, TotAmt.

On the left side of SellerTbl, there is SellerProductTbl which has the same function as SellerBillTbl which is useful for combining SelletTbl with ProductTbl. Furthermore, in the ProductTbl table, there are table attributes such as; Proid, ProName, ProQty, ProdPrice, ProdCat, and Catd which are foreign keys to the CategoryTbl table later.

On the right side of the ProductTbl table, there is a relationship with the ProductCategoryTbl table which is the same as other connecting tables to join the CategoryTbl. In the CategoryTbl table, there are attribute variables to determine the category of a product, such as Catd, CatName, CatDesc. Then, there is a table that is useful as a summary of the sales results of a shopping mart, the ItemSell table. In the itemSell table, there are variables; Proid which is the foreign key in the

ProductTbl table, ProName, ProdPrice, ProQty, and ProTotal as attributes that explain how many types of products have been sold.

2.4.2 Entity Relationship Diagram (ERD)

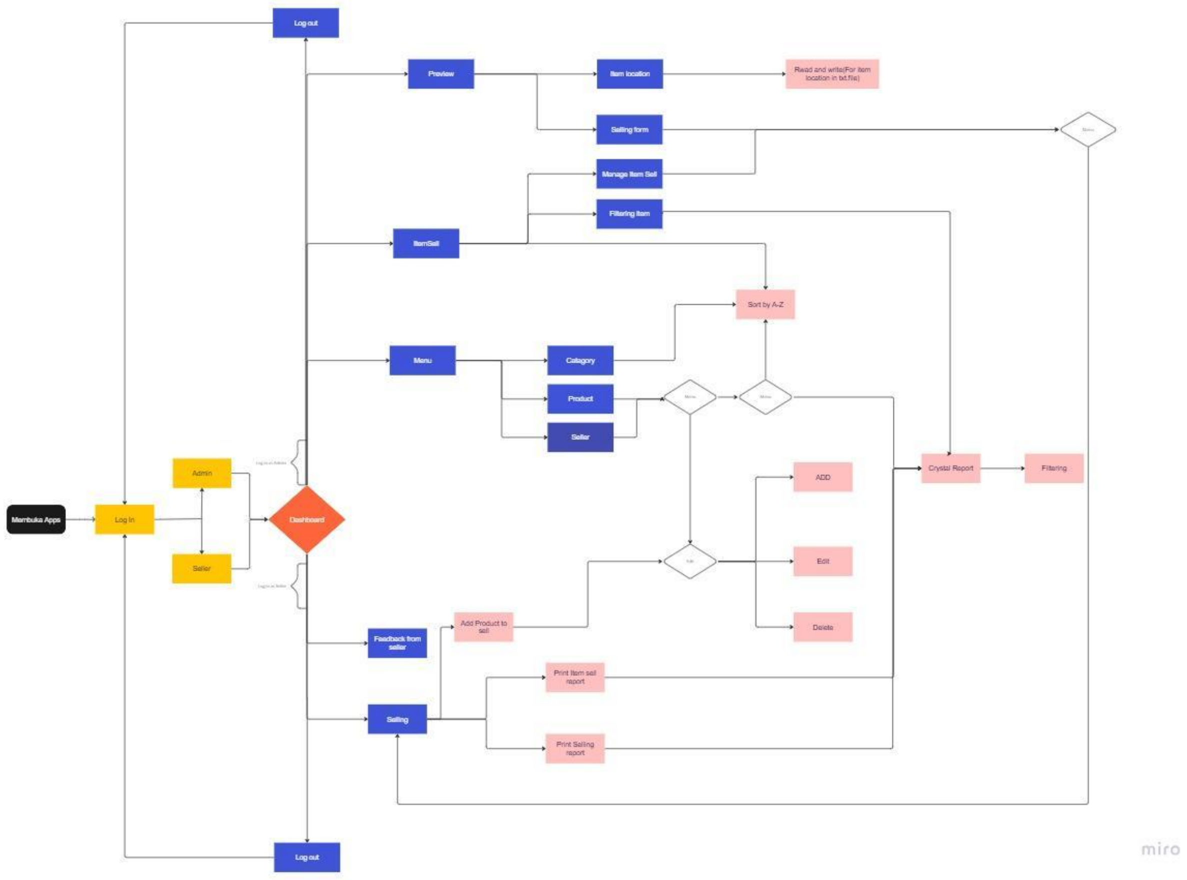


Picture 2.2 Entity Relationship Diagram (ERD)

So, here is the Entity Relationship Diagram we have for Fresh Supermarket, and we have a total of 8 tables: sellerTbl, sellerproduct, sellerbilltbl, billtbl, productTbl, itemSold, ProductCategoryTbl, and CategoryTbl. The main and parent table is SellerTbl. SellerTbl has a one-to-many relationship with SellerBillTbl, where SellerBillTbl can create multiple bills. SellerTbl also has a one-to-many relationship with sellerProduct, where each seller can have multiple products. Within sellerProduct, there is productTbl which also has a one-to-many relationship with products.

Additionally, productTbl has a many-to-one relationship with itemSold, indicating that multiple items can be sold under a single product. However, productTbl has a one-to-one relationship with ProductCategoryTbl, which means each product is associated with a single category. ProductCategoryTbl, in turn, provides a description of the categories using a one-to-one relationship with CategoryTbl.

2.5 Flowchart



Picture 3.3 Flowchart App

Here is a flowchart of the Fresh Supermarket application so the flow starts from opening the application, after you open the application there are 2 login options, namely as Admin and also as a Seller. We will start explaining starting from logging in as an admin first, so when you log in as an admin there will be a dashboard that contains 4 main menus, each of which is: Menu. Preview, ItemList, Log out.

Let's start from the Menu, here we are provided with the option to add products via the product window, then add an admin via the admin window. as well as adding categories for products in the category menu. The three menus above have one of the same functions, namely sort by A-Z, but different from the category menu, the product and seller menus can be filtered by category. Sellers will be filtered by age while products by category.

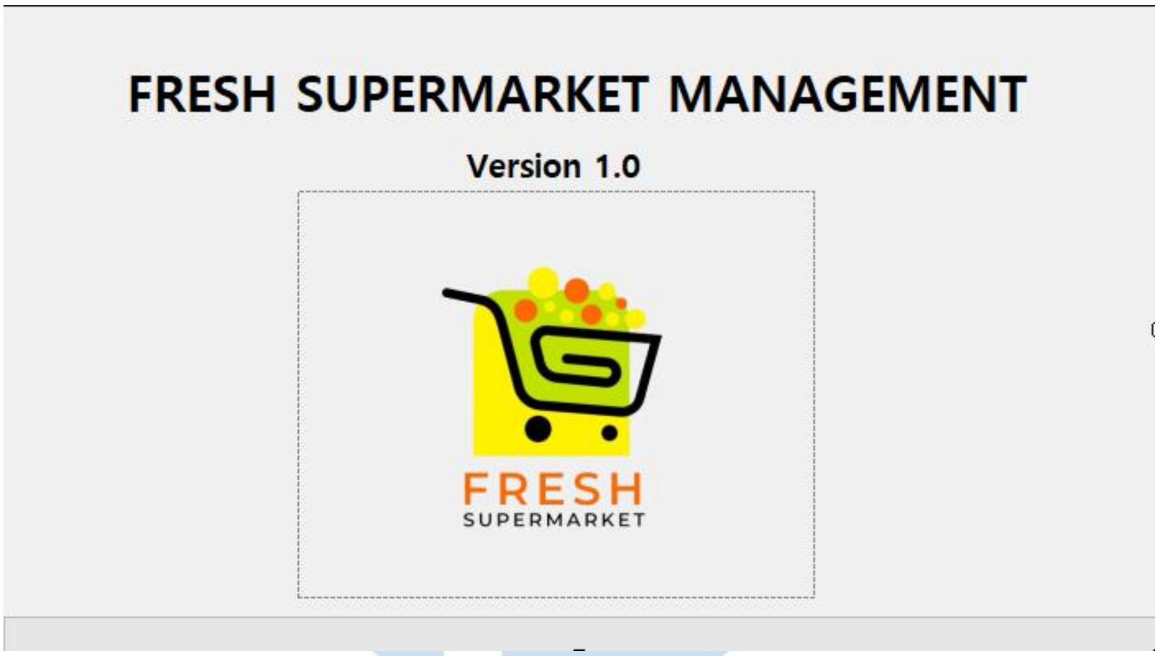
Continuing to the Preview menu here, you will be given 2 options for the selling form menu and item location. The selling menu here, you as the admin can enter the selling report made by the seller. While ItemLocation is a read and write function that will create a list of sample item locations (Item A : Shelf 1).

Continuing to the Itemlist menu, there are 3 main menus, namely Sort, Filtering items, Manage selling items. so before continuing, let's understand what itemsell is, so item sell is a menu that contains a bill for sellers per customer. Sorting in the item sell menu itself using Sort A-Z will sort the products starting with the letters A-Z. then filtering items, filtering here uses a crystal report so items will be filtered according to the name of the item sold. Then manage sell items, this menu is used if the seller experiences an error in recording the bill, then the admin can edit the contents of the bill besides that the seller can also access this menu too. Finally, the Logout menu, this menu aims to end the session and return to the login page.

Continuing to login as a seller, when you have successfully logged in, you will enter the home page. On this page, you can add products ordered by the customer, all selected products will be accommodated and will become a bill which can later be printed using a crystal report. then besides the home page there is an itemsell page the same as I explained in the itemsell admin menu here also has the same concept as itemsell in the admin. after you finish using this menu you can logout and return to Login

CHAPTER 3
APPLICATION LAYOUT

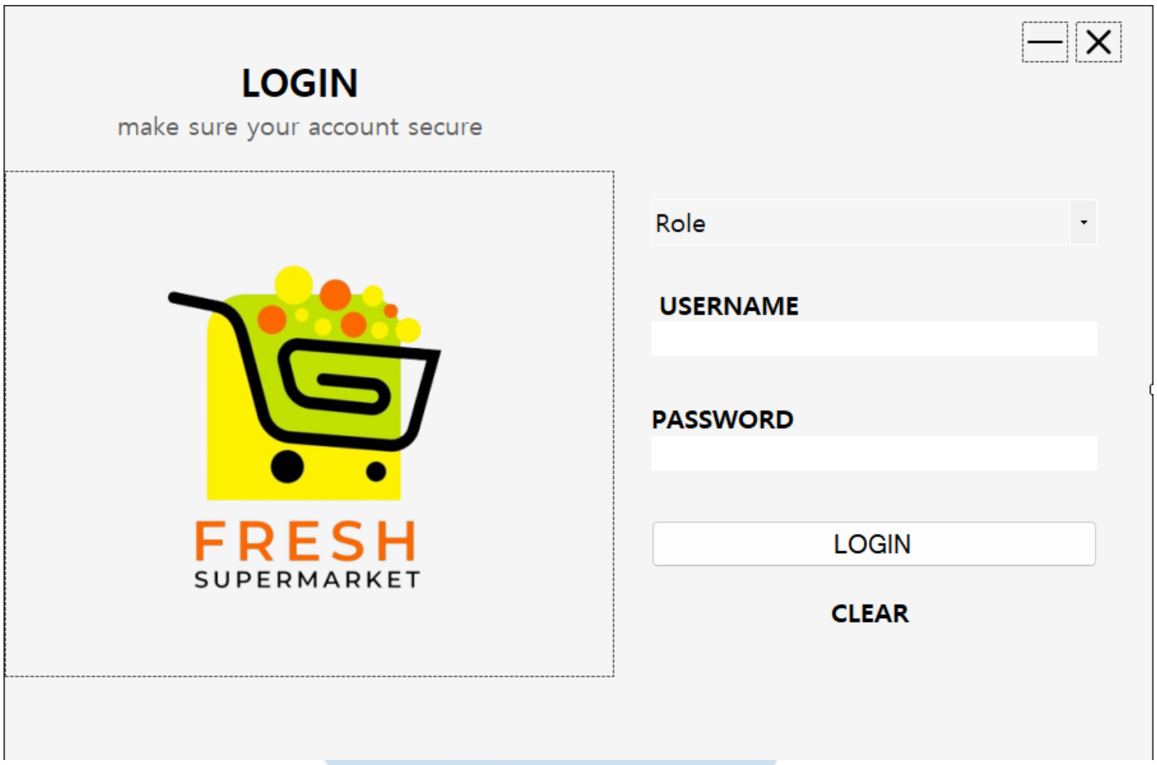
3.1 Loading Screen



Picture 3.1 Loading Screen
Supermarket Management
Application

So, our application, Fresh Supermarket Management, has a Loading Screen that serves as the opening before users enter the system, followed by the Login Page.

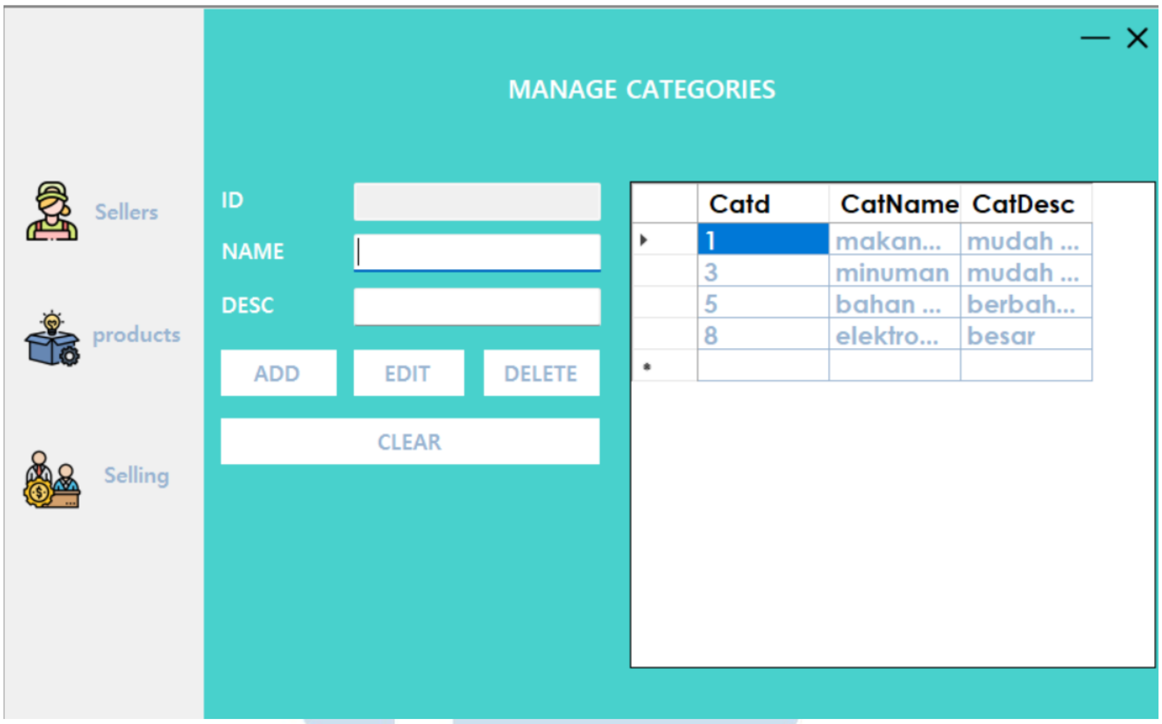
3.2 Login Page



Picture 3.2 Login Page Supermarket
Management Application

After passing through the Loading Screen, There is a button that allows users to select the role they want to log in as, either user or admin. After that, they need to enter their username and password to log in. Once they have entered the credentials, they can click on the login button. If they want to clear the entered username and password, they can click on the "Clear" button. Once the user or admin clicks the login button, they will be redirected to the category form.

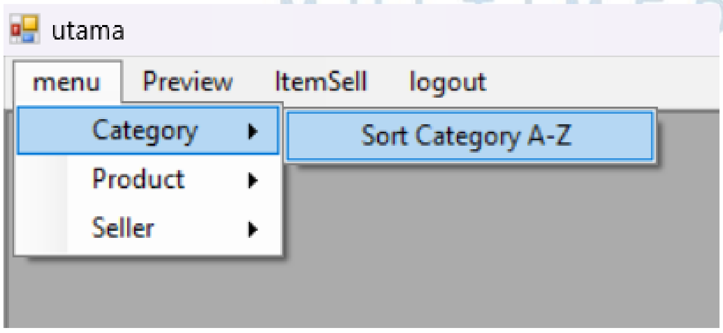
3.3 (Admin) Category Form



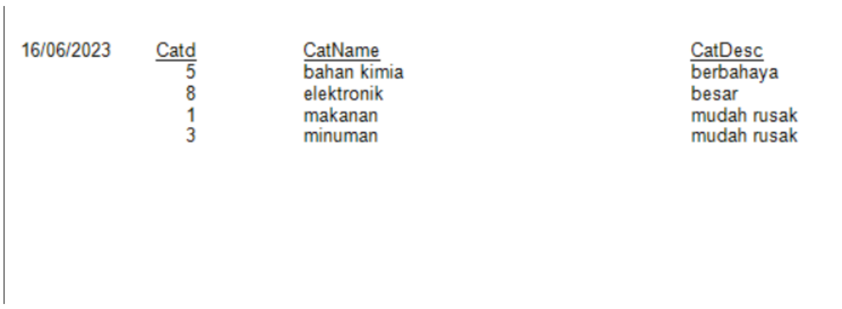
Catd	CatName	CatDesc
1	makan...	mudah ...
3	minuman	mudah ...
5	bahan ...	berbah...
8	elektro...	besar

Picture 3.3 (Admin) Category Form

Above is a page that contains a Category Form, where we need to enter the ID of the food first, for example, 1, 2, 3. Then, enter the name of the food, for example, "nasi goreng", and provide a description for it. For example, "nasi goreng is an instant food." Once the ID, name, and description are entered, we can click the "Add" button to add it. If we want to edit the ID or name, we can click the "Edit" button. To delete an entry, we can simply click the "Delete" button. If we want to clear all the entries, we can click the "Clear" button. Once all the information such as ID, name, and description are entered correctly, the result will be displayed in a box on the right side.



Picture 3.4 (Admin) Category Form, Sort Category A-Z



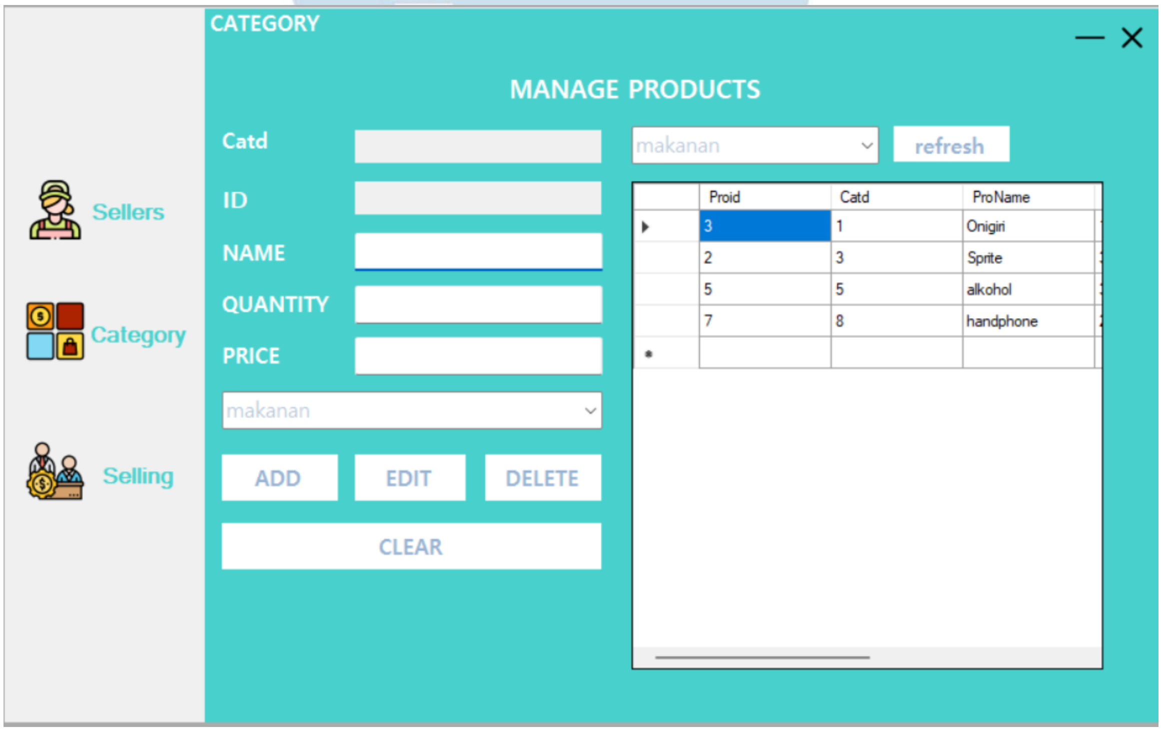
16/06/2023

Catd	CatName	CatDesc
5	bahan kimia	berbahaya
8	elektronik	besar
1	makanan	mudah rusak
3	minuman	mudah rusak

Picture 3.5 (Admin) Category Form, Sort
Category A-Z result

In order to sort categories in ascending and descending order, you need to click the sort category A-Z button, and you will see a crystal report appear that will show you the categories which have been sorted in ascending and descending order.

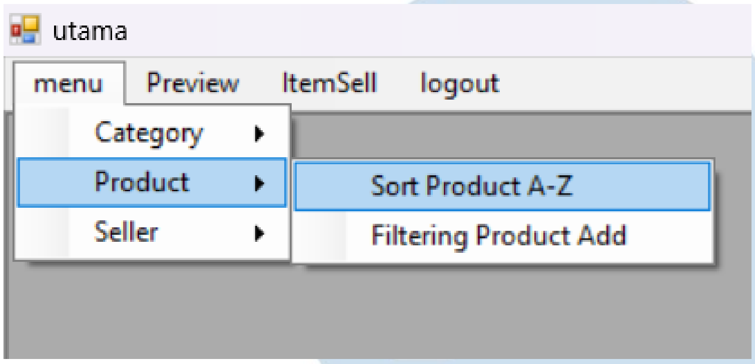
3.4 (Admin) Product Form



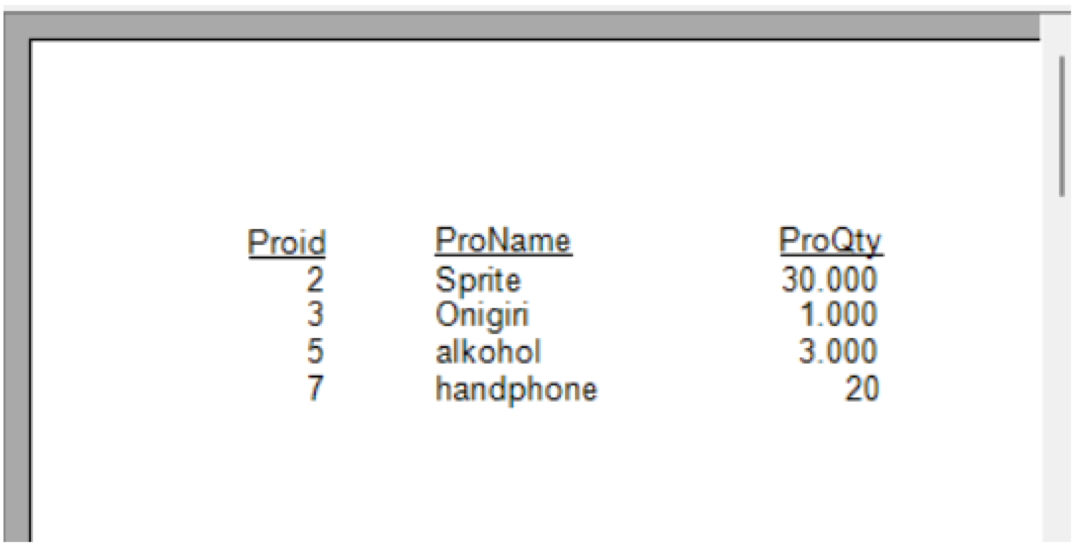
Proid	Catd	ProName
3	1	Onigiri
2	3	Sprite
5	5	alkohol
7	8	handphone

Picture 3.6 (Admin) Product Form

This is the page for adding products to the admin window. so after you fill in the categories on the previous page called categories, you can continue filling out this page. on this page you can add various types of products according to the categories that have been added. As an example above we use the food category so we can add food names such as bread, jam, rice, chicken or other food. After all inputs have been filled in and you click add, the product will be accommodated in the database. You can refresh it to see the list. besides the add window here there is also edit and delete. let's explain from edit so edit here will edit products that are already registered in the database. For example, you gave the wrong name or the quantity you typed is wrong, then you can edit it through this function. Continuing to the delete window, if you want to delete a product you can use the delete window. With this function, products in the database can be deleted by selecting them. besides that there is also a clear menu on this page, this menu clears all the text in the name, quantity and price textboxes.



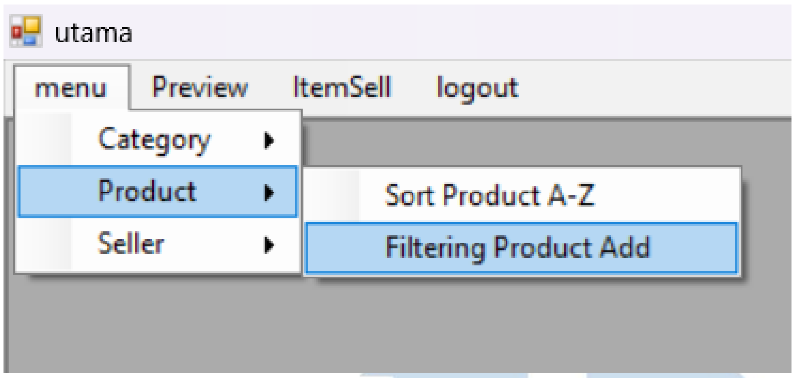
Picture 3.7 (Admin) Product Form, Sort Product A-Z



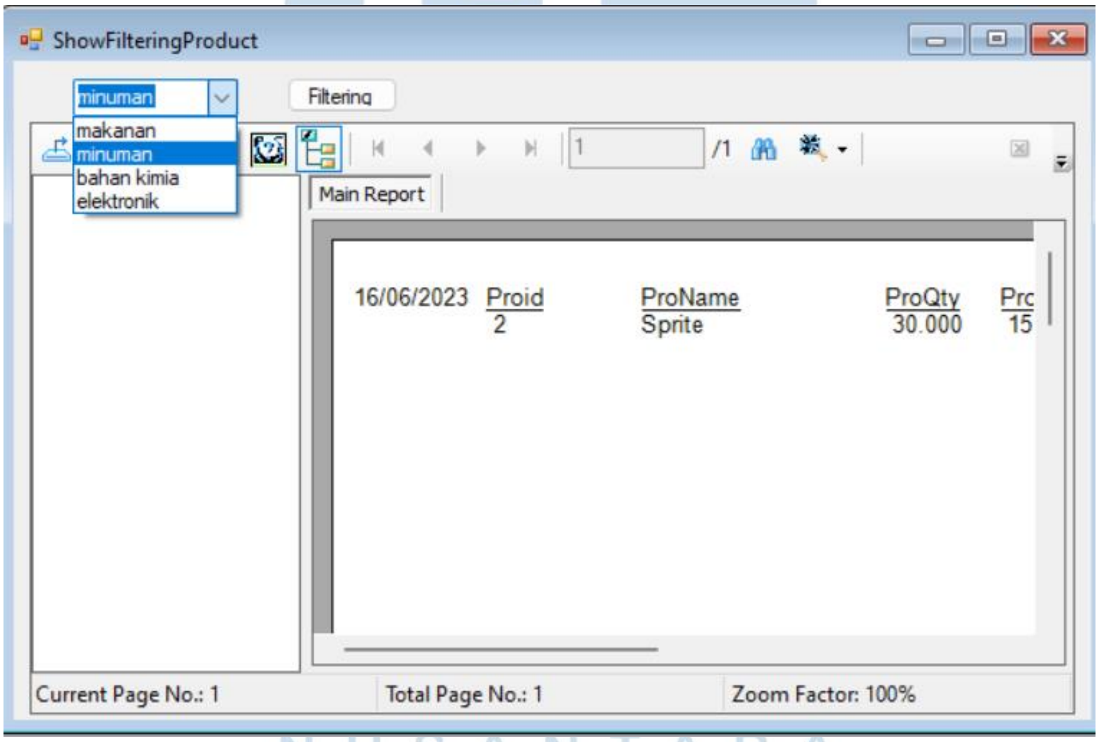
Proid	ProName	ProQty
2	Sprite	30.000
3	Onigiri	1.000
5	alkohol	3.000
7	handphone	20

Picture 3.8 (Admin) Product Form, Sort Product A-Z Result

The following screen will appear when you choose 'Sort Product A-Z'. A crystal report will display an ascending and descending product list. We have provided a table where you can sort products by price, name, and quantity to suit your preferences. One-click access to product details and in-depth analysis of specific products can be performed with crystal report.



Picture 3.9 (Admin) Product Form, Filtering Product Add



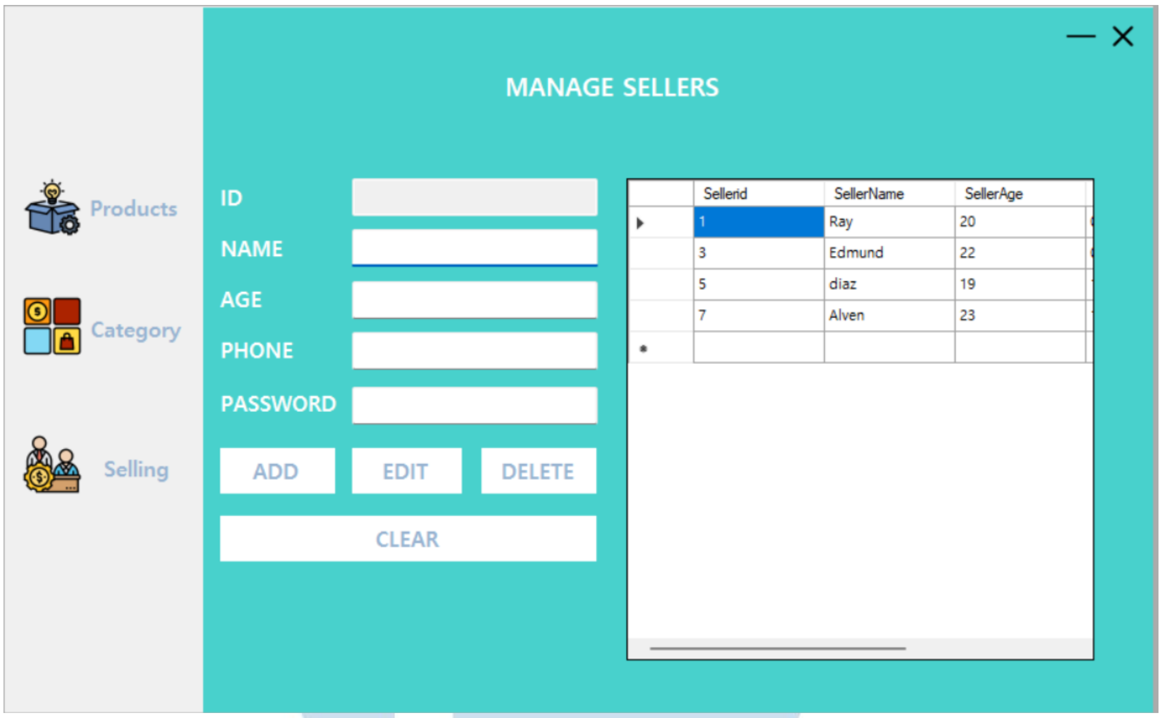
16/06/2023 Proid ProName ProQty Prc
2 Sprite 30.000 15

Current Page No.: 1 Total Page No.: 1 Zoom Factor: 100%

Picture 3.10 (Admin) Product Form, Filtering Product Add Crystal Report View

Here is the result of filtering the data named Filtering Product Add, such as food, beverages, chemicals, and electronics. If we want to search for beverages, for example, it will directly perform the filtering and display the results in the Crystal Report view on the right side.

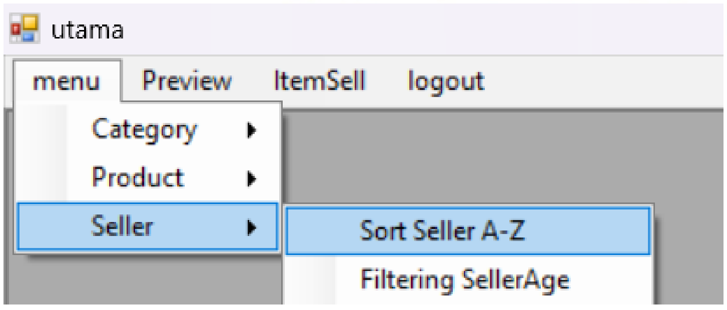
3.5 (Admin) Seller Form



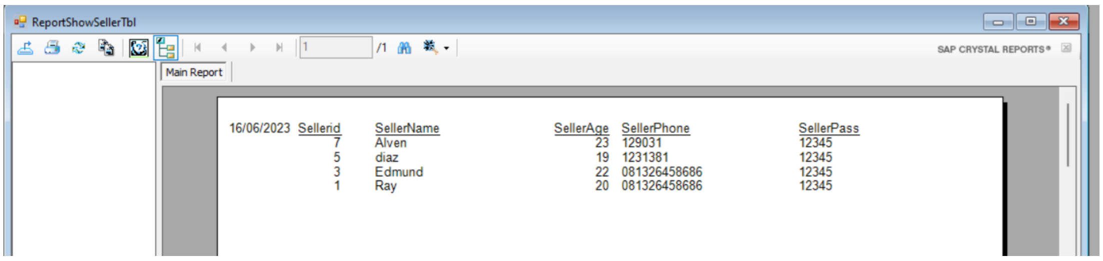
Sellerid	SellerName	SellerAge
1	Ray	20
3	Edmund	22
5	diaz	19
7	Alven	23

Picture 3.11 (Admin) Seller Form

Here is the Seller Form page, where the seller can enter the attributes/identity of the seller. They can enter the ID of the worker, for example, 1, 2, 3. Then, they can enter the name of the worker, for example, Ray. They also need to enter the age of the worker, for example, 20, and the contact phone number, for example, 08129391383. Additionally, they need to set a password for the seller. Once all the information is entered, they can click the "Add" button to add it. If they want to make changes, they need to click the "Edit" button. To delete an entry, they can click the "Delete" button. If they want to clear all the entries, they need to click the "Clear" button. Once everything is completed, the list of workers will be displayed on the right side.



Picture 3.12 (Admin) Seller Form, Sort Seller A-Z



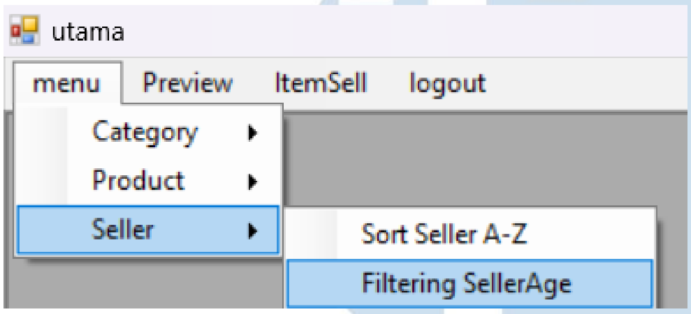
16/06/2023

Sellerid	SellerName	SellerAge	SellerPhone	SellerPass
7	Alven	23	129031	12345
5	diaz	19	1231381	12345
3	Edmund	22	081326458686	12345
1	Ray	20	081326458686	12345

Picture 3.13 (Admin) Seller Form, Sort Seller A-Z

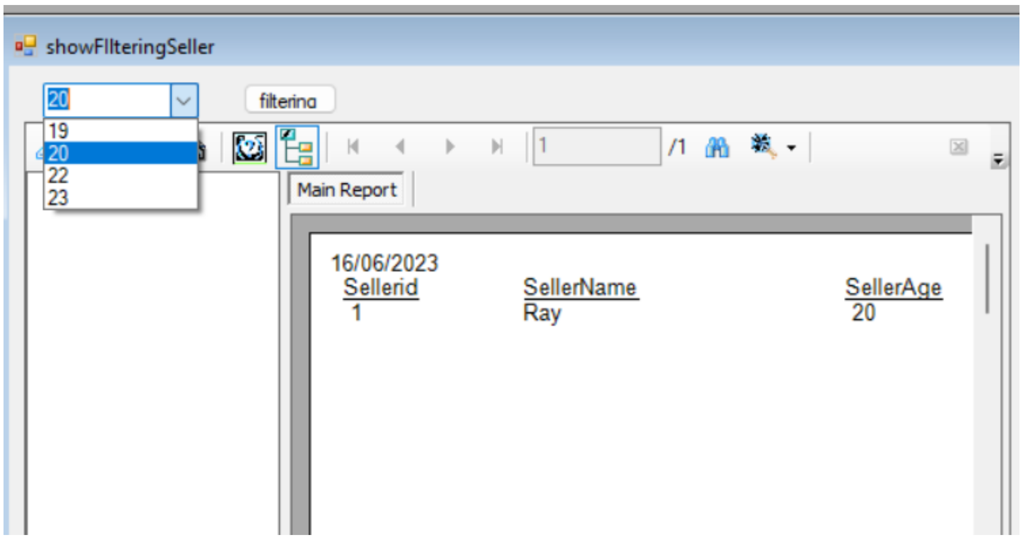
Result in Crystal Report View

The above features enable the sorting of seller in both ascending and descending order. After clicking on the menu a crystal report will appear as above



Picture 3.14 (Admin) Seller Form,

Filtering Seller Age



16/06/2023

Sellerid	SellerName	SellerAge
1	Ray	20

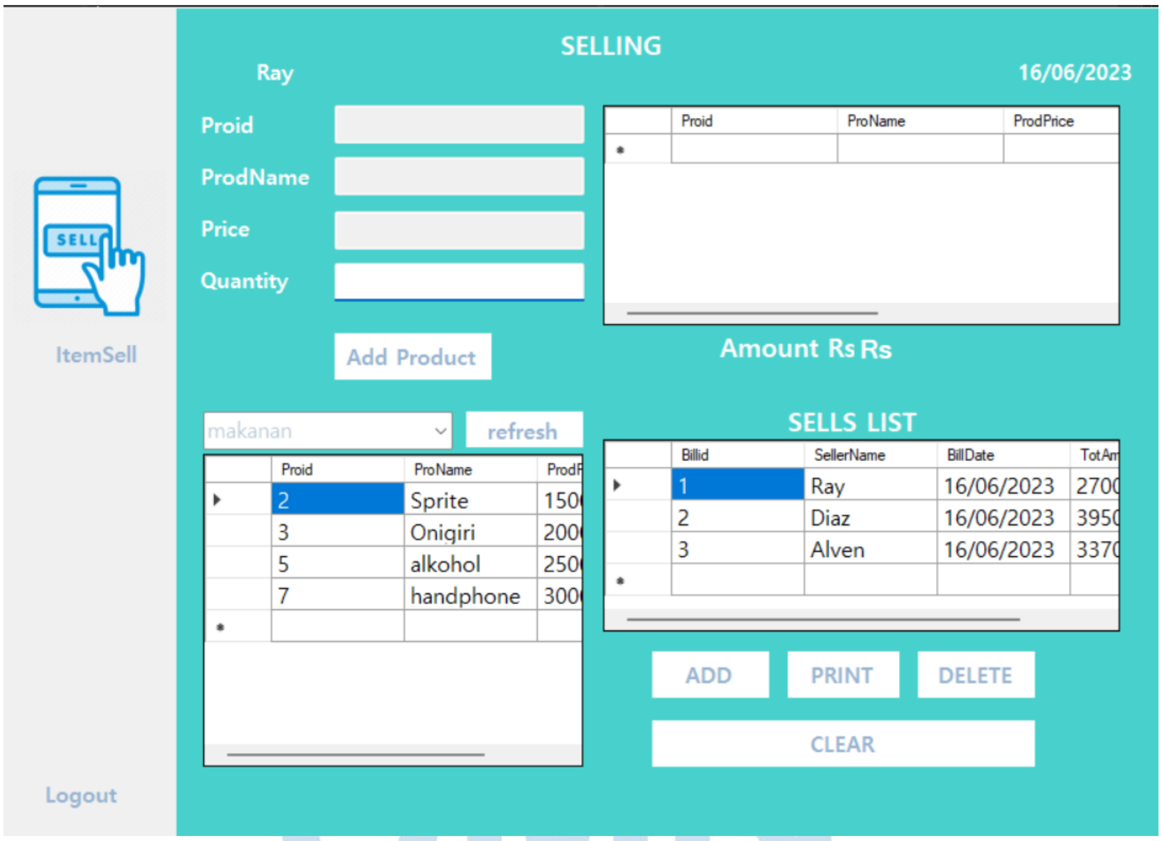
Picture 3.15 (Admin) Seller Form,

Filtering Seller Age Result in Crystal

Report View

Here is the data filtering named Filtering SellerAge based on the age of the workers. With the filtering, you can divide the age into categories such as 19, 20, 22, 23, and so on. For example, if you want to search for workers aged 20, it will directly display the workers who are 20 years old in the Crystal Report View.

3.6 (Seller) Selling Form



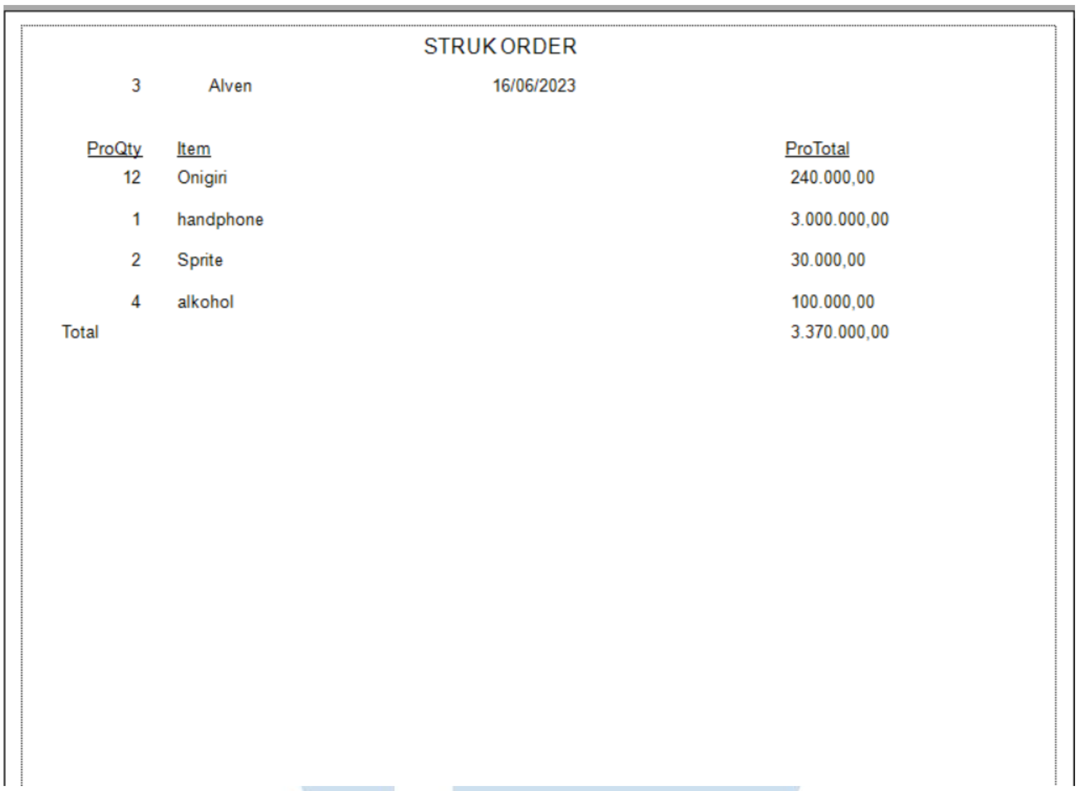
	Proid	ProName	ProdPrice
*			

	Proid	ProName	ProdF
▸	2	Sprite	150
	3	Onigiri	200
	5	alkohol	250
	7	handphone	300
*			

	BillId	SellerName	BillDate	TotAm
▸	1	Ray	16/06/2023	270
	2	Diaz	16/06/2023	395
	3	Alven	16/06/2023	337
*				

Picture 3.16 (Seller) Selling Form

Above is the (Admin) Selling Form where the seller can make sales to customers. For example, the seller enters the ID of the item, such as 1, 2, 3. Then, they enter the name of the item purchased by the customer, for example, Onigiri. They also enter the price of the item purchased by the customer, for example, 20000. Next, they enter the quantity of the item purchased. All of this can be automatically entered into the Textbox to speed up the process. If the customer's purchased items are entered correctly, they will be displayed on the right side. The seller can add the item by clicking the "Add" button. They can also print a receipt of the purchase, delete an item, or clear the inputted items.


STRUK ORDER

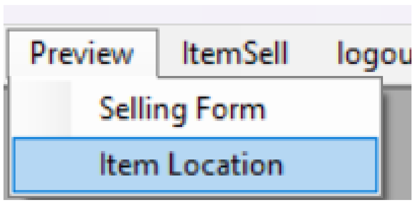
3	Alven	16/06/2023

ProQty	Item	ProTotal
12	Onigiri	240.000,00
1	handphone	3.000.000,00
2	Sprite	30.000,00
4	alkohol	100.000,00
Total		3.370.000,00

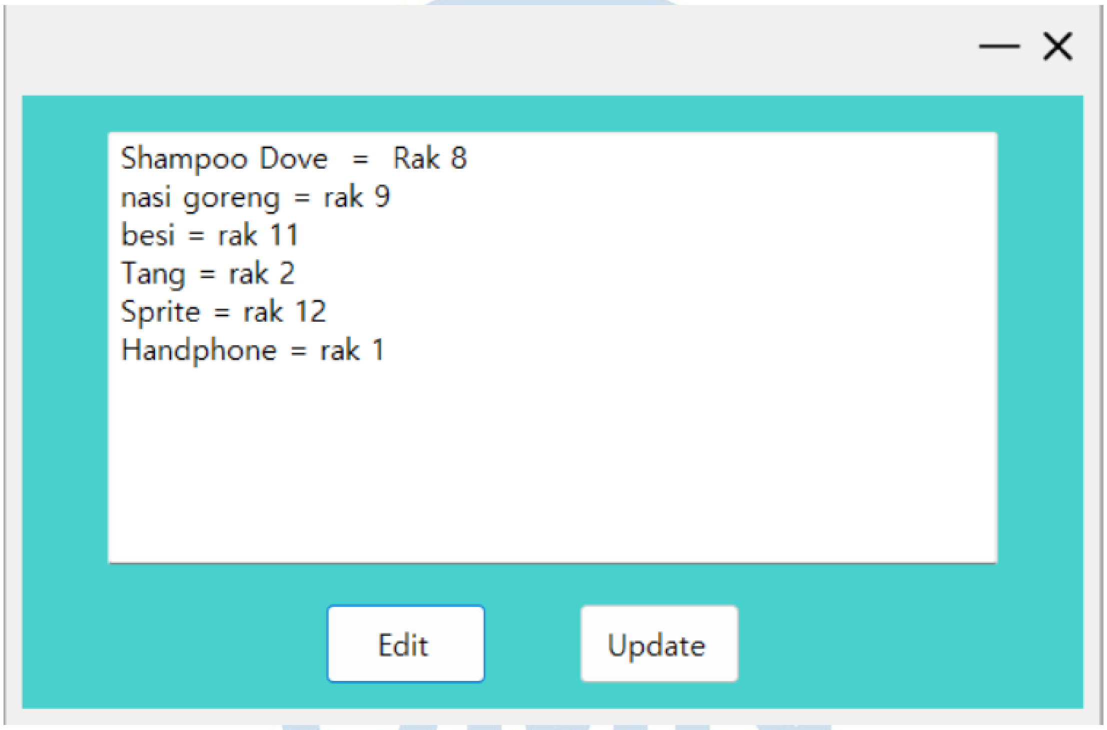
Picture 3.17 (Seller) Selling Form, Print

The following is an example of an order using crystal report (in Admin Login) structure resulting from product sales per customer or purchase. you can see that here we display the seller's name, date, and also the product. the explanation is the proqty which is the quantity product, then the items and also the sub total per product. all sub-totals are totaled and will generate a Nominal Total

3.7 (Admin) Item Location

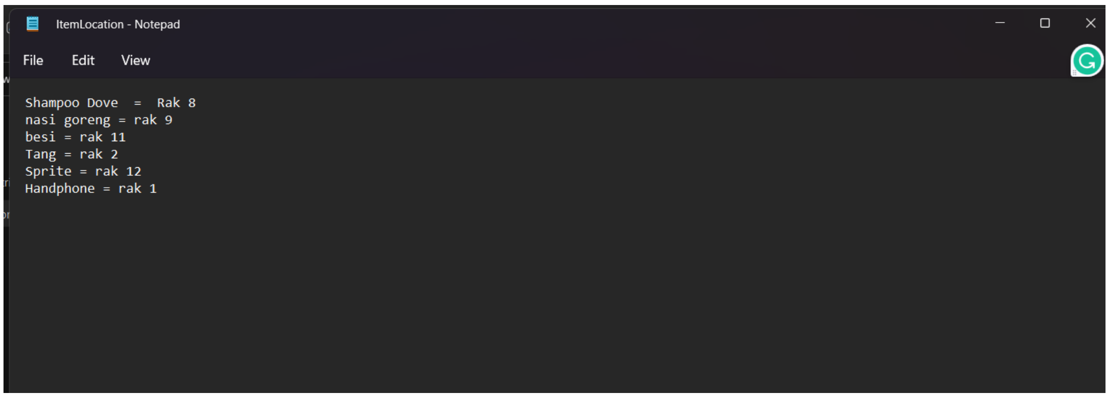


Picture 3.18 (Admin) Item Location



```
Shampoo Dove  =  Rak 8
nasi goreng = rak 9
besi = rak 11
Tang = rak 2
Sprite = rak 12
Handphone = rak 1
```

Picture 3.19 (Admin) Item Location, Formating .txt

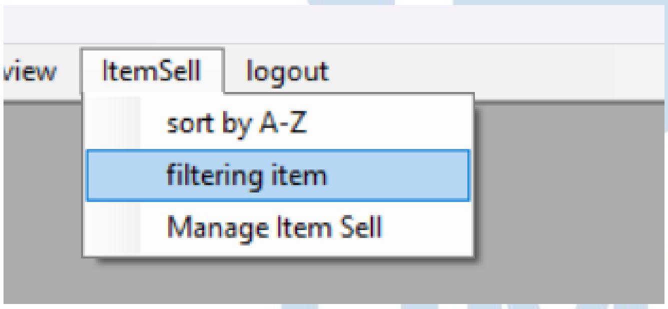

ItemLocation - Notepad

```
Shampoo Dove  =  Rak 8
nasi goreng = rak 9
besi = rak 11
Tang = rak 2
Sprite = rak 12
Handphone = rak 1
```

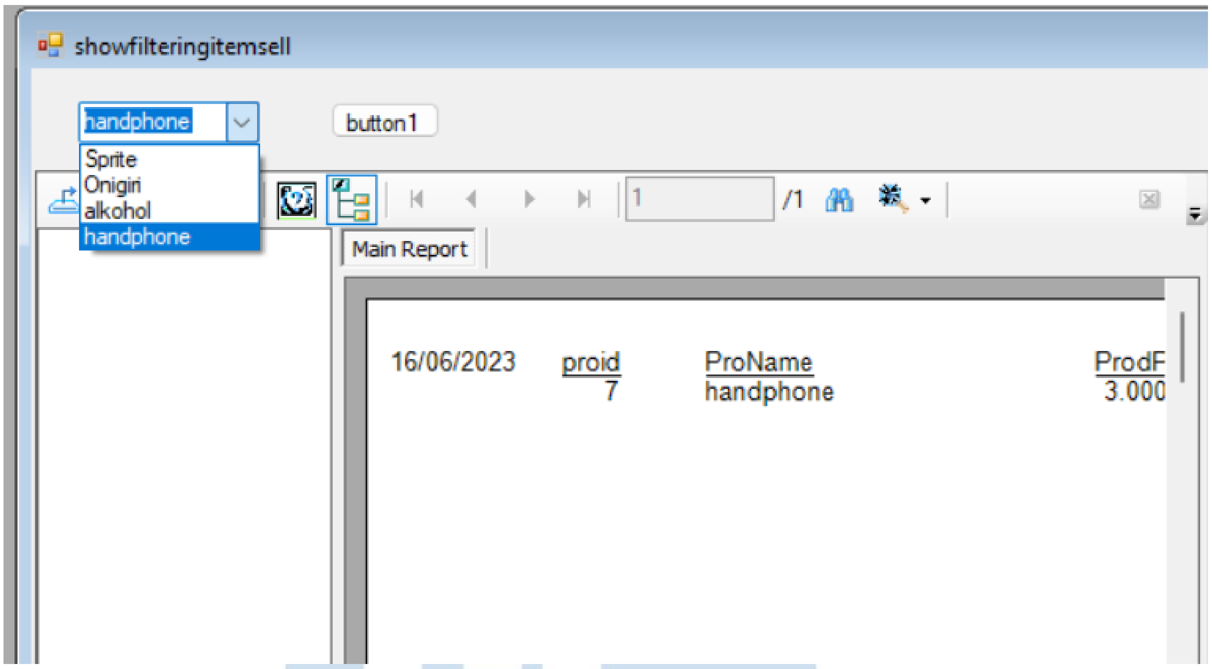
Picture 3.20 (Admin) Item Location, Result file.txt

This feature will allow the admin to determine and see directly the location of the product in the supermarket, this feature itself uses the read and write principle to an available txt file.

3.8 (Admin) ItemSell Filtering



Picture 3.21 (Admin) Item Sell, Filtering Item

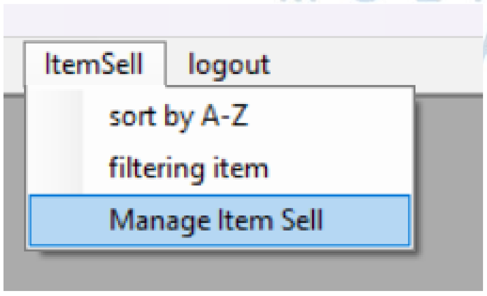


16/06/2023 proid ProName ProdF
7 handphone 3.000

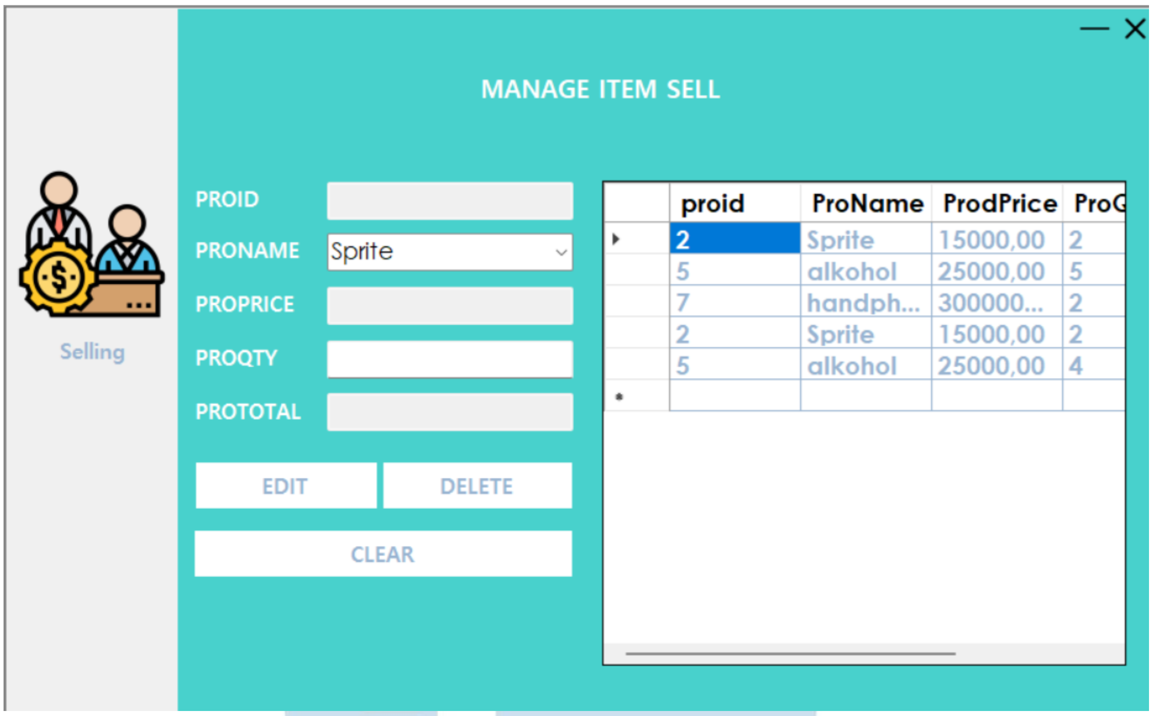
Picture 3.22 (Admin) Item Sell, Filtering Result

Next is the Item Filtering. This filtering is used to separate item categories such as Sprite, Onigiri, Alcohol, and Mobile Phones. If you want to search for mobile phones, you can simply click on the "Mobile Phones" category, and the data will be filtered and displayed in the Crystal Report view on the right side.

3.9 (Admin) ItemSell Manage Item Sell



Picture 3.23 (Admin) Item Sell, Manage Item Sell



PROID	
PRONAME	Sprite
PROPRICE	
PROQTY	
PROTOTAL	

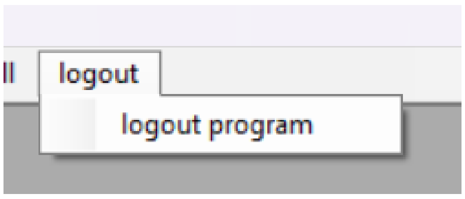
proid	ProName	ProdPrice	ProQ
2	Sprite	15000,00	2
5	alkohol	25000,00	5
7	handph...	300000...	2
2	Sprite	15000,00	2
5	alkohol	25000,00	4

Picture 3.24 (Admin) Item Sell, Manage Item Sell

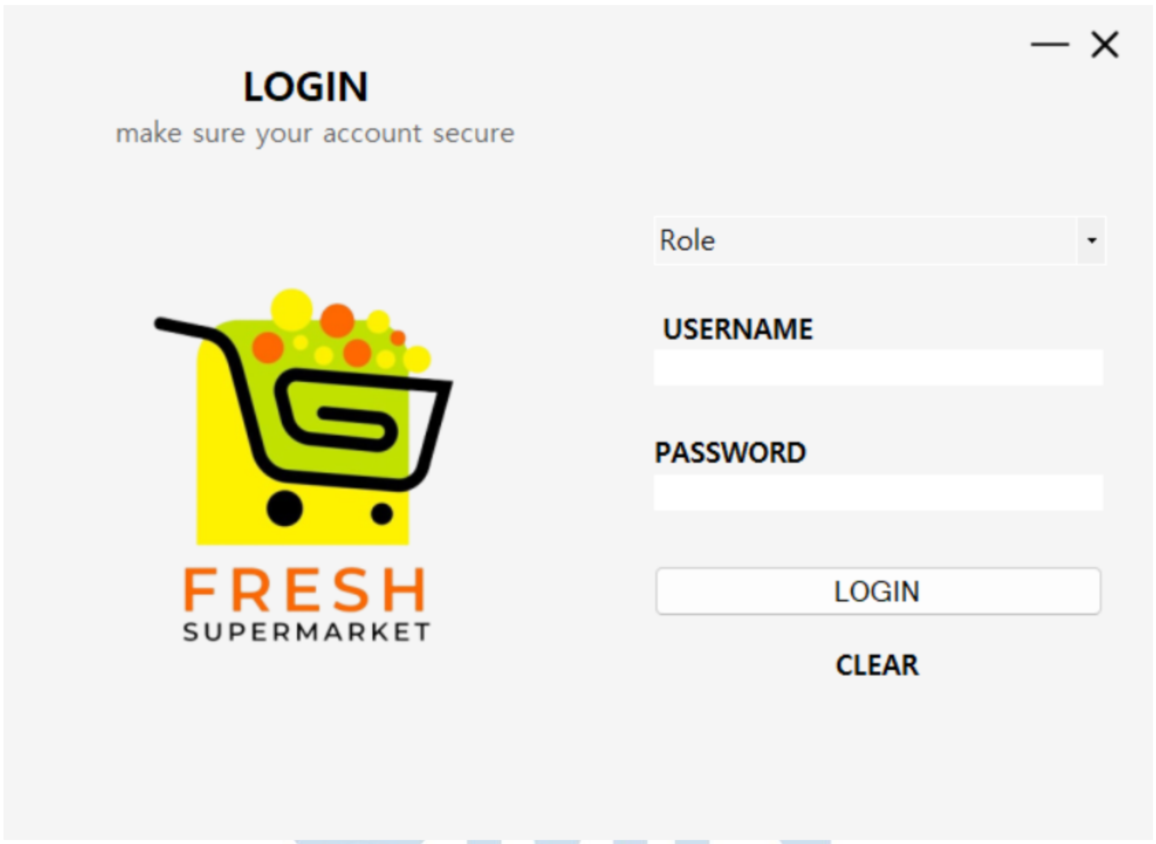
Information

Itemsell is used to edit items that have been sold, for example a customer doesn't buy, you can delete it or if your seller input it incorrectly then you can use the edit feature, the way this page works is very easy for you as a user to understand, namely by clicking on a product that you want to change then select your choice between edit or delete. if not so then click clear

3.10 Logout



Picture 3.25 Logout Menu Strip

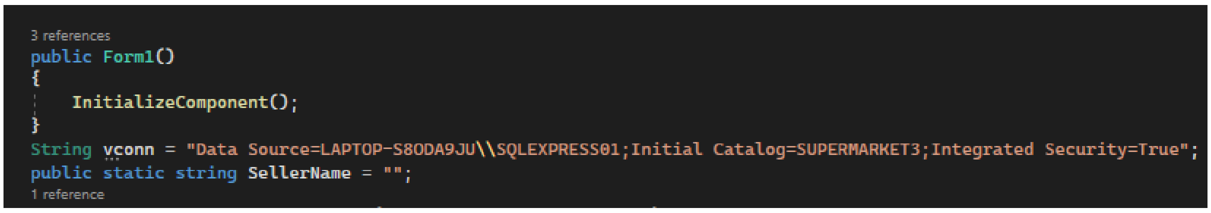


Picture 3.26 Logout Page

This button is used to end the login session and return to the initial login page.

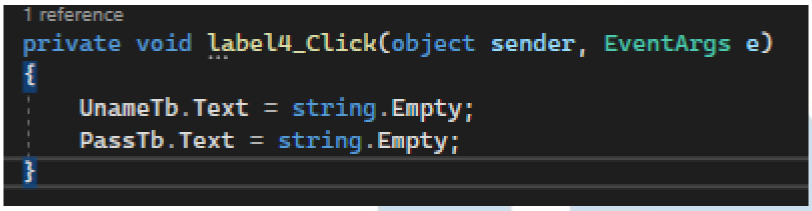
CHAPTER 4
CODING

LOGIN FORM



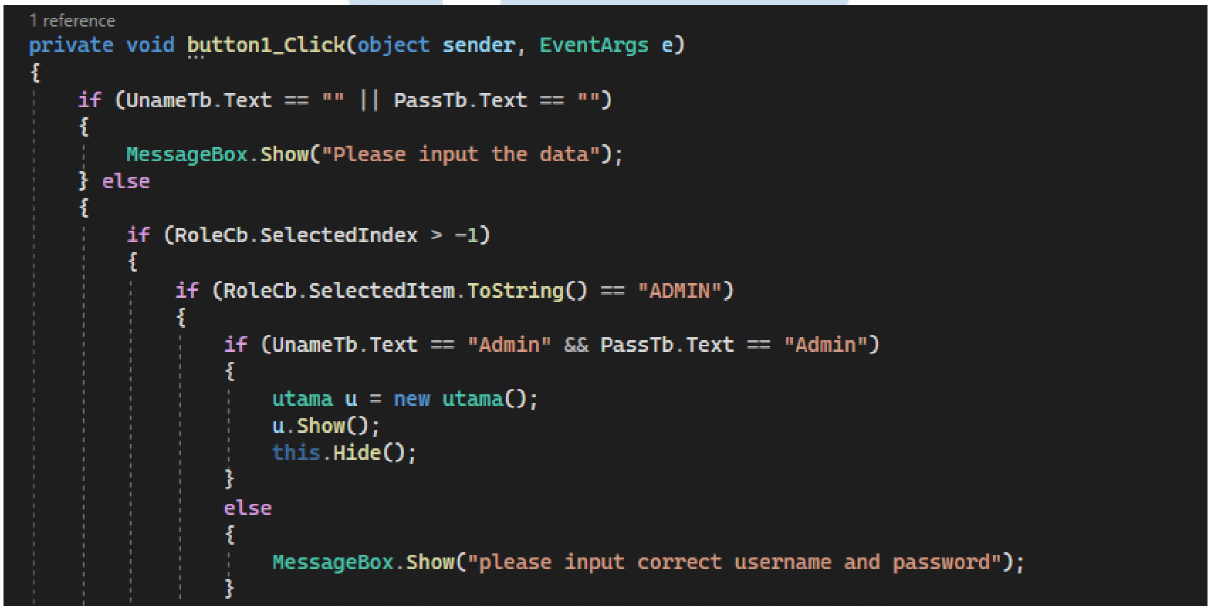
```
3 references
public Form1()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
public static string SellerName = "";
1 reference
```

The code above is the code to call the database that we use. In the code above, the database we use is named "SUPERMARKET3". Then on the next line, I created a String with public access called SellerName to create a naming on the login.



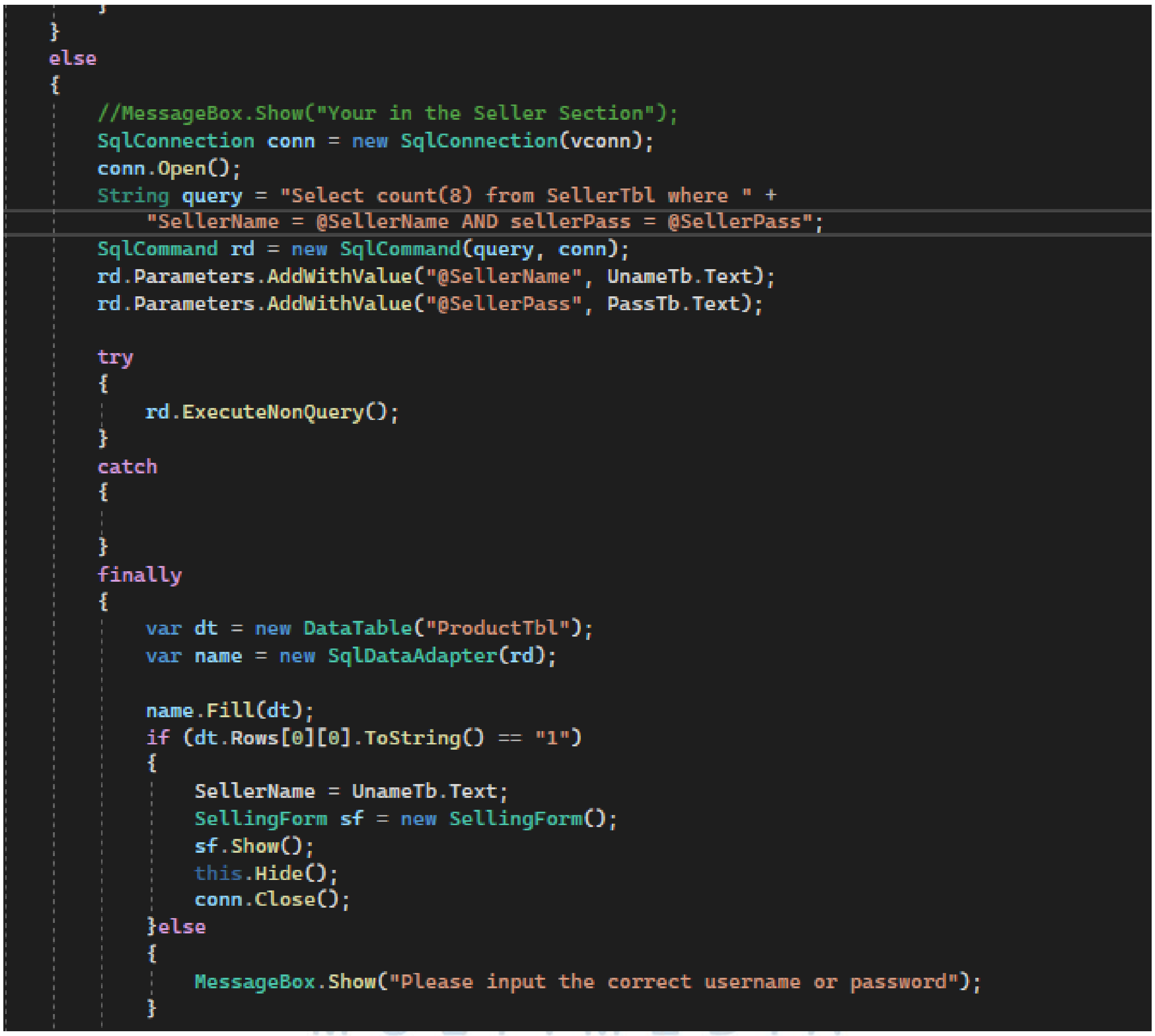
```
1 reference
private void label4_Click(object sender, EventArgs e)
{
    UnameTb.Text = string.Empty;
    PassTb.Text = string.Empty;
}
```

The code above is a code to empty the value that has been filled in the textbox, making it easier for admins and sellers to delete data if they experience difficulties.



```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    if (UnameTb.Text == "" || PassTb.Text == "")
    {
        MessageBox.Show("Please input the data");
    } else
    {
        if (RoleCb.SelectedIndex > -1)
        {
            if (RoleCb.SelectedItem.ToString() == "ADMIN")
            {
                if (UnameTb.Text == "Admin" && PassTb.Text == "Admin")
                {
                    utama u = new utama();
                    u.Show();
                    this.Hide();
                }
                else
                {
                    MessageBox.Show("please input correct username and password");
                }
```

The code above is an explanation of the code on the login button. In the first statement, the first if means that if the username and password textboxes are not filled in by the admin or seller, it will issue a "Please input the data" messagebox. However, if the textbox is filled in, it will enter the second if statement. The second if statement checks whether there is an item selected in the role combobox. If there is a selected item then the code in the second if statement will be executed. After all the conditions have been met, it will enter the third if statement. In the third if is a code to check whether the item in the role combobox is "ADMIN". If what is selected is "ADMIN" then the system will check the username textbox and password textbox. when the third if statement has fulfilled what is ordered in the code then it will enter the fourth if statement. The fourth if statement code contains if the username textbox and password textbox are the same as "Admin", then we successfully enter the login and enter the "main" display form and the login form is hidden. However, if the username and password do not match, it will display a messageBox "please input correct username and password".

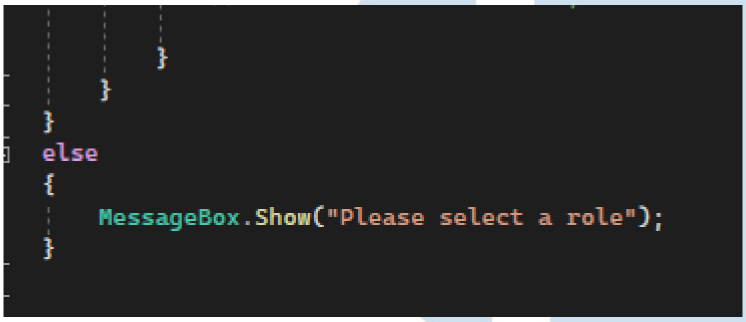


```
    }
    else
    {
        //MessageBox.Show("Your in the Seller Section");
        SqlConnection conn = new SqlConnection(vconn);
        conn.Open();
        String query = "Select count(8) from SellerTbl where " +
            "SellerName = @SellerName AND sellerPass = @SellerPass";
        SqlCommand rd = new SqlCommand(query, conn);
        rd.Parameters.AddWithValue("@SellerName", UnameTb.Text);
        rd.Parameters.AddWithValue("@SellerPass", PassTb.Text);

        try
        {
            rd.ExecuteNonQuery();
        }
        catch
        {

        }
        finally
        {
            var dt = new DataTable("ProductTbl");
            var name = new SqlDataAdapter(rd);

            name.Fill(dt);
            if (dt.Rows[0][0].ToString() == "1")
            {
                SellerName = UnameTb.Text;
                SellingForm sf = new SellingForm();
                sf.Show();
                this.Hide();
                conn.Close();
            }else
            {
                MessageBox.Show("Please input the correct username or password");
            }
```

Then we move on to else in the fourth if statement. The code in else is executed when the user does not choose the "ADMIN" role but chooses the "SELLER" role. In the code in else we first connect to the database using SqlConnection.

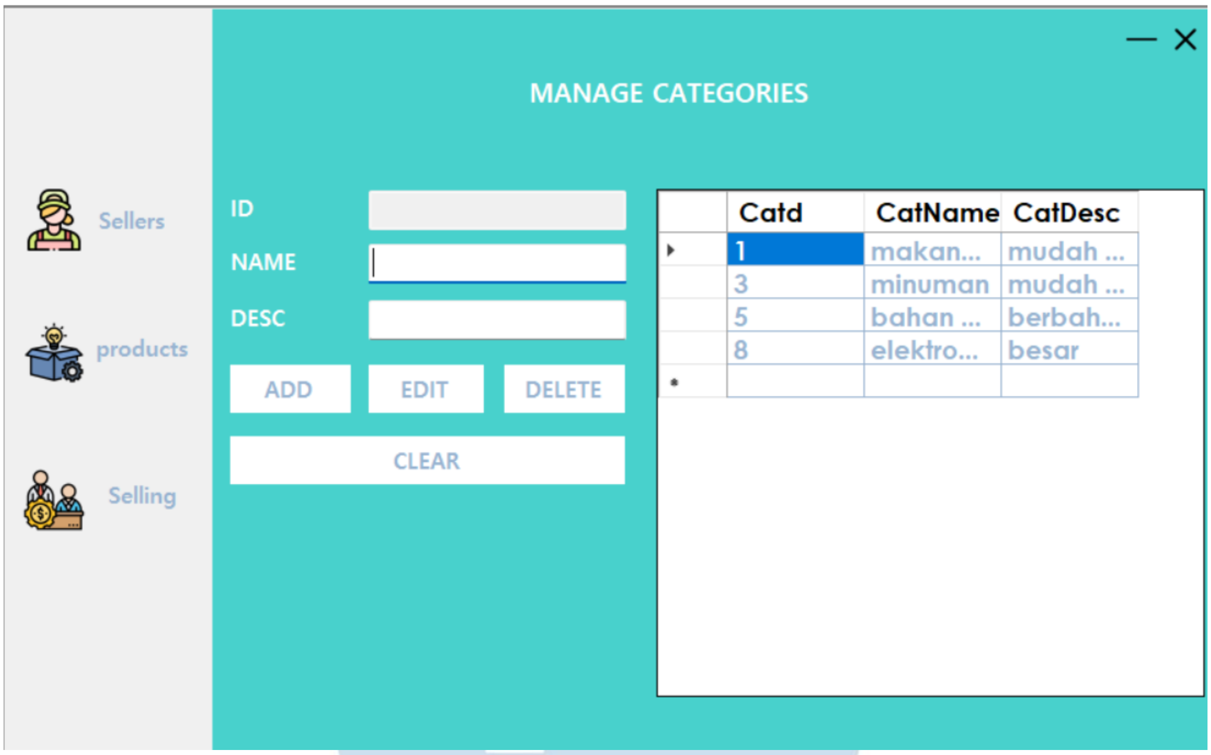
Then SQL Query is used to count the number of rows that match the "Seller Name" and "Seller Pass" entered by the user. Then the parameters "@sellername" and "@sellerpass" are adjusted to the values in the username and password textboxes and are also adjusted to the Seller Tbl database table in the "username" and "password" sections.

Then "rd.executeNonQuery()" is used to run the query and return the results of the row count calculation. Then in the "Finally" section we create a new datatable object with the name "SellerTbl".name.Fill(dt) means that the data from rd (SqlDataAdapter) is filled into the DataTable dt. if (dt.Rows[0][0].ToString() == "1") this line checks whether the value in the first row, first column of DataTable dt is "1". If it is, then the username and password entered by the user are valid. The Seller Name variable is set with the value of TextBox UnameTb. When the username and password match the value in the "SelletTbl" table, the selling form is displayed and the login form is hidden and the connection to the database is closed. If the username or password does not match a value not "1", a warning message will be displayed asking the user to enter the correct username or password.

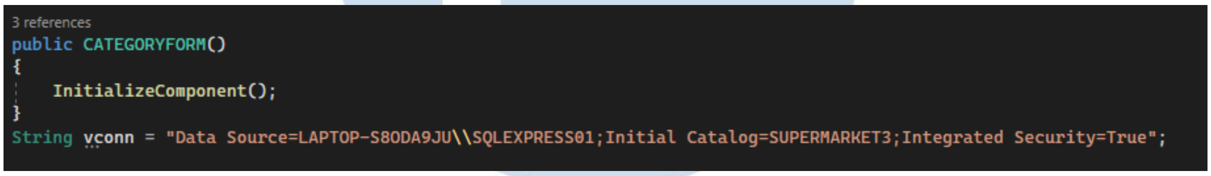

```
            }
        }
    }
    else
    {
        MessageBox.Show("Please select a role");
    }
```

If the user has not selected a role in the combobox or no item is selected in the ComboBox "RoleCb", it will display a MessageBox "Please select a role".

CATEGORY FORM

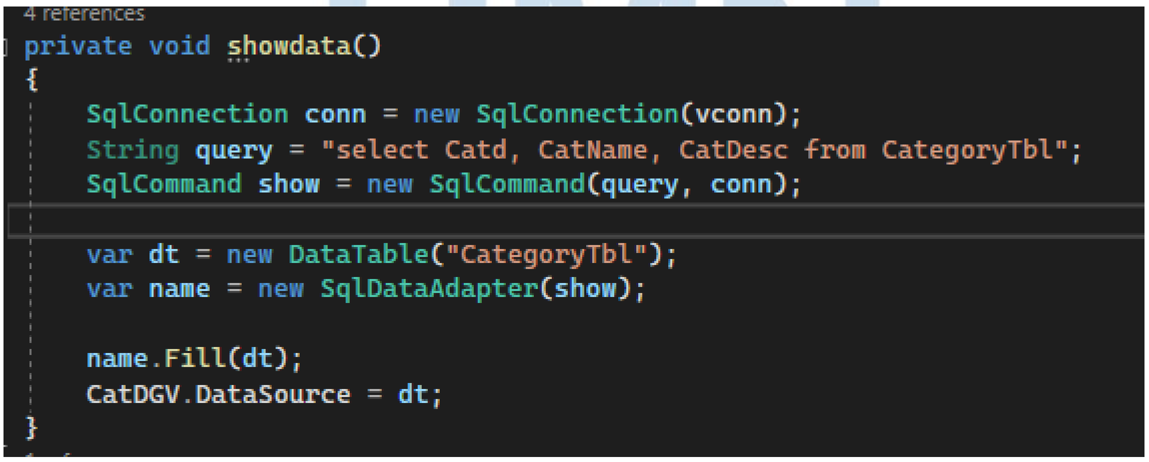


Catd	CatName	CatDesc
1	makan...	mudah ...
3	minuman	mudah ...
5	bahan ...	berbah...
8	elektro...	besar



```
3 references
public CATEGORYFORM()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
```

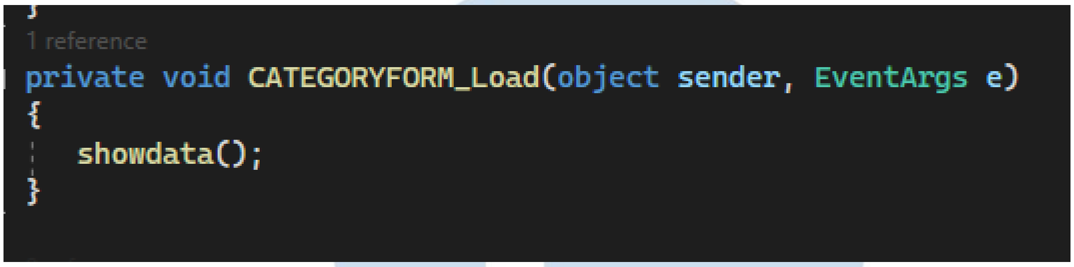
The code above is the code to call the database that we use. In the code above the database we use is named "SUPERMARKET3".



```
4 references
private void showdata()
{
    SqlConnection conn = new SqlConnection(vconn);
    String query = "select Catd, CatName, CatDesc from CategoryTbl";
    SqlCommand show = new SqlCommand(query, conn);

    var dt = new DataTable("CategoryTbl");
    var name = new SqlDataAdapter(show);

    name.Fill(dt);
    CatDGV.DataSource = dt;
}
```

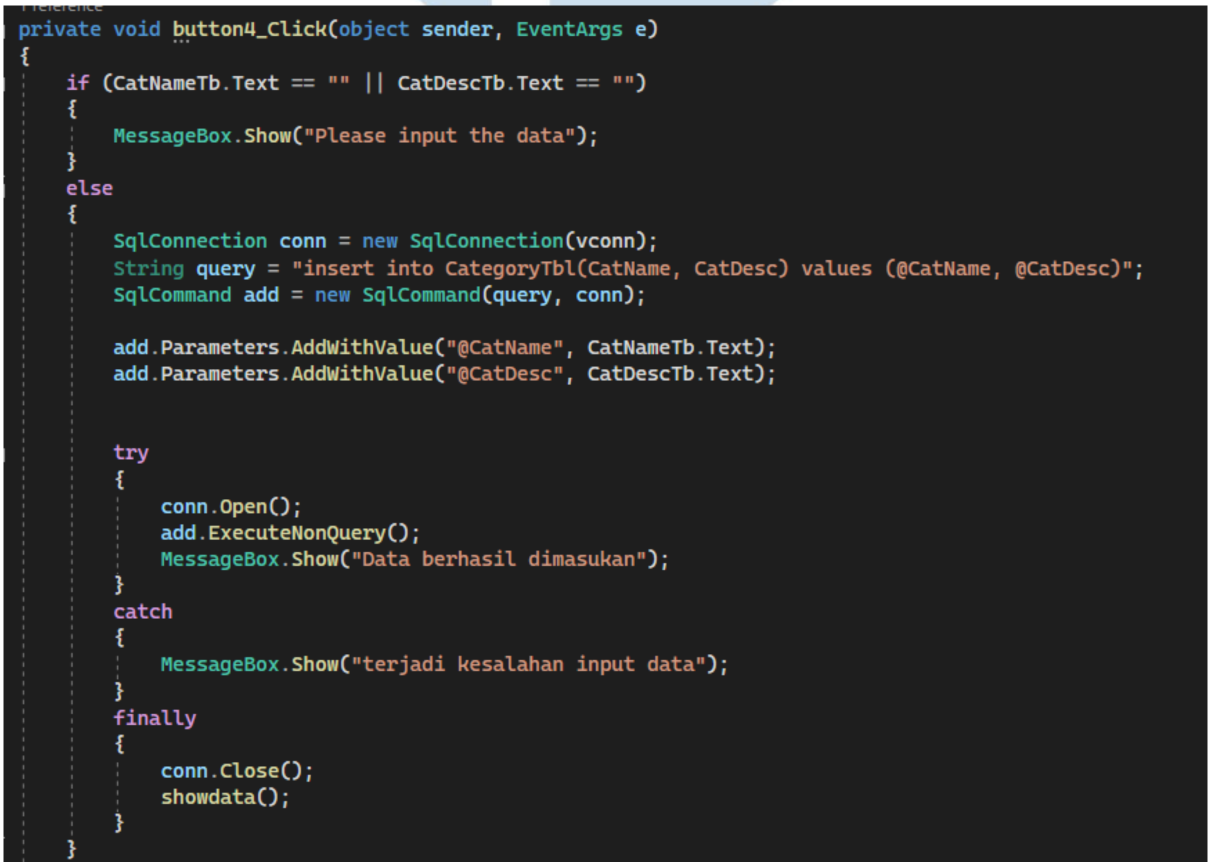
The code above is a class function that our group created to display data, namely "DataGridView" on "CategoryForm". We rename the "DataGridView" with the name "CatDGV". Then the query above is used to retrieve data from the "CategoryTbl" table. This query will retrieve the columns "Catd", "CatName", and "CatDesc". Then the next line creates a SqlCommand object using the SQL query and SqlConnection object created earlier. Then on

"New DataTable" to create a new DataTable object with the name "CategoryTbl". This will be used to store the query results. then Create a SqlDataAdapter object using the SqlCommand object that was created earlier. Then Fills the DataTable object (dt) with the data obtained through the SqlDataAdapter (name). The data will be retrieved from the database using the SqlCommand object and finally Set the DataSource data source of the DataGridView "CatDGV" with the DataTable (dt) object. Thus, DataGridView will display the data in DataTable. However, the function above is still only a class function that has been created but not implemented yet.



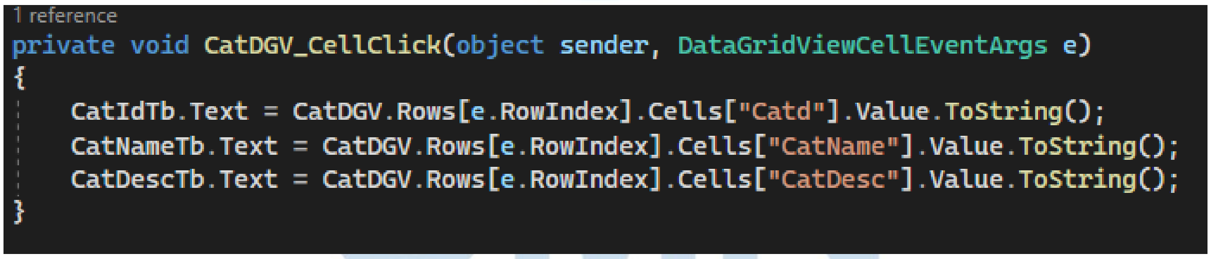
```
}
1 reference
private void CATEGORYFORM_Load(object sender, EventArgs e)
{
    showdata();
}
```

To implement, we need to name the class "CATEGORYFORM_load" so that the data can be shown in the "DataGridView".



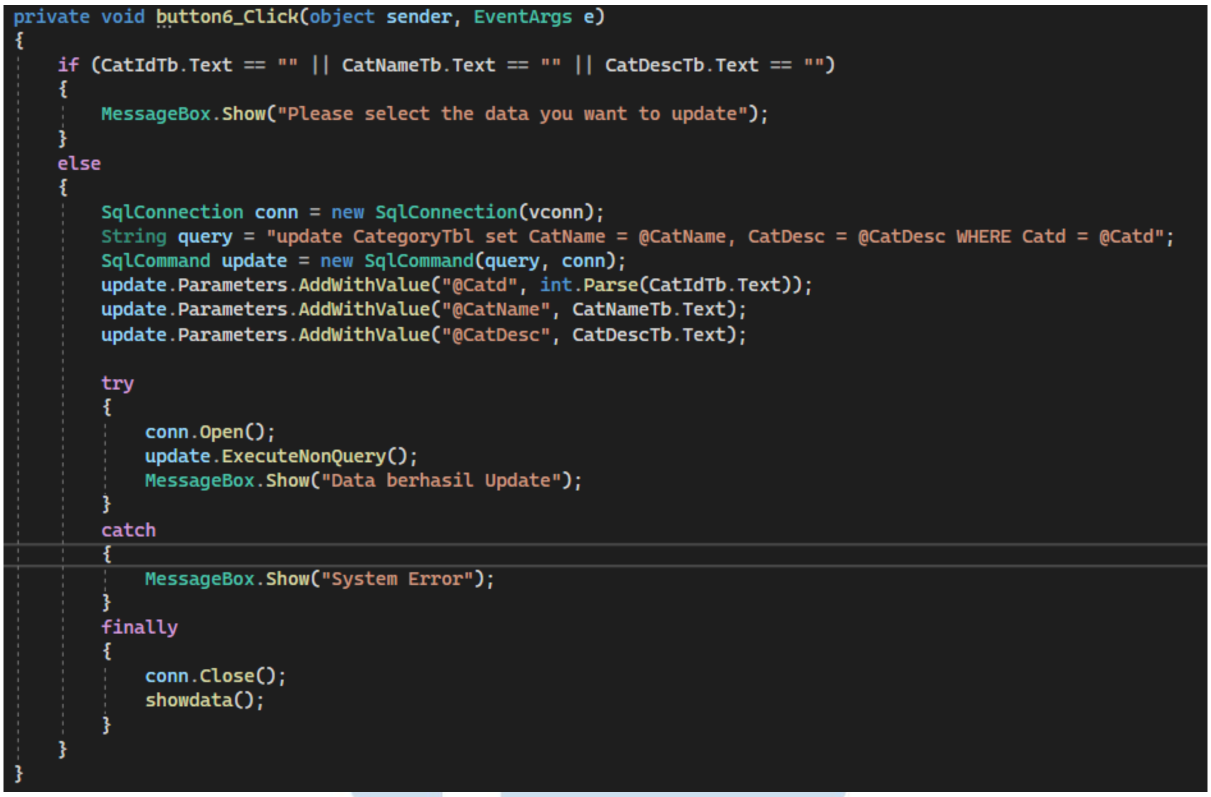
```
1 reference
private void button4_Click(object sender, EventArgs e)
{
    if (CatNameTb.Text == "" || CatDescTb.Text == "")
    {
        MessageBox.Show("Please input the data");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "insert into CategoryTbl(CatName, CatDesc) values (@CatName, @CatDesc)";
        SqlCommand add = new SqlCommand(query, conn);

        add.Parameters.AddWithValue("@CatName", CatNameTb.Text);
        add.Parameters.AddWithValue("@CatDesc", CatDescTb.Text);


        try
        {
            conn.Open();
            add.ExecuteNonQuery();
            MessageBox.Show("Data berhasil dimasukan");
        }
        catch
        {
            MessageBox.Show("terjadi kesalahan input data");
        }
        finally
        {
            conn.Close();
            showdata();
        }
    }
```

The code above is a code that has an "add" button. The "add" button above has a function to enter values into the data table in the database then the data is shown in the "DataGridView" that we have. The first if statement has a function to check whether the textboxes "CatNameTb" and "CatDescTb" have a value or not. If not then a messagebox will come out "Pleasse input the data". But if so, the process will continue in the else statement. Sql Connection as usual connects to the database. Then it stores the SQL query that will be used to insert the insert data into the "CategoryTbl" table. This query will insert the value of CatNameTb.Text into the "CatName" column and the value of CatDescTb.Text into the "CatDesc" column. Then SqlCommand has the same function as the class that our group created, namely "showdata". Then add parameters to "@catName" and "@CatDesc", ensuring that the values of CatNameTb.Text and "CatDesc.Text" will be used as the values to be entered into the query. Then we open the database with con.open and we run the query. When a data is successfully entered with the function that our group has created, it will display a messageBox "Data successfully entered" and the database will be closed. However, if there is an error in the process of adding data to the database, it will issue a messageBox "there was a data input error". Then in the "Finally" Statement, if the data is successfully entered, the data in the "DataGridView" will be updated with the "ShowData" function.



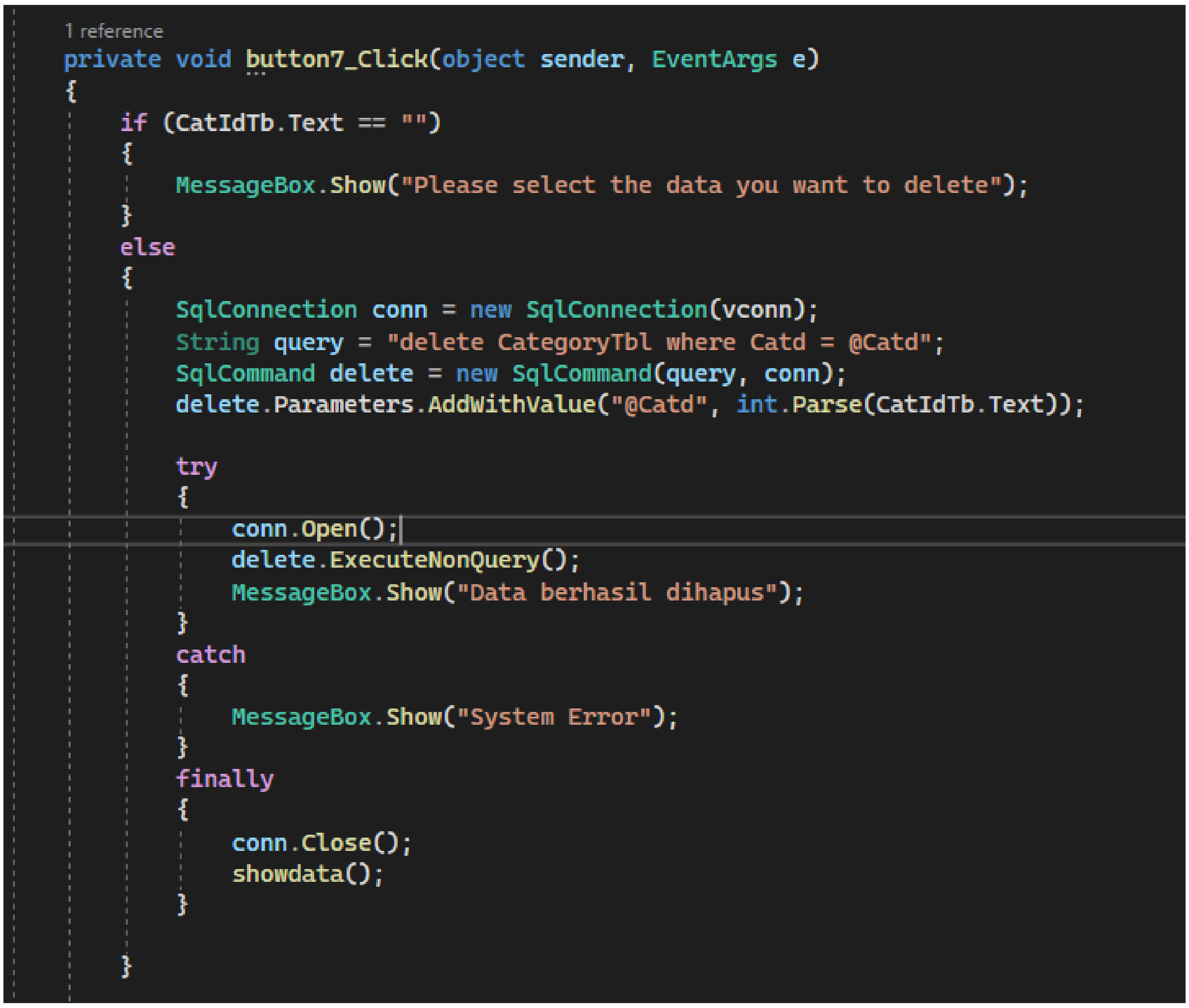
```
1 reference
private void CatDGV_CellClick(object sender, DataGridViewCellEventArgs e)
{
    CatIdTb.Text = CatDGV.Rows[e.RowIndex].Cells["Catd"].Value.ToString();
    CatNameTb.Text = CatDGV.Rows[e.RowIndex].Cells["CatName"].Value.ToString();
    CatDescTb.Text = CatDGV.Rows[e.RowIndex].Cells["CatDesc"].Value.ToString();
}
```

Then the function above has a function for when we click one row to the right on the "DataGridView" then the data in the table will appear in the Textbox according to the directions in the code.

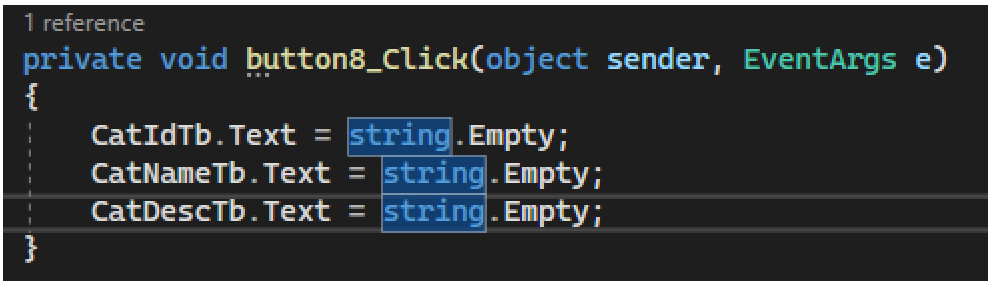


```
private void button6_Click(object sender, EventArgs e)
{
    if (CatIdTb.Text == "" || CatNameTb.Text == "" || CatDescTb.Text == "")
    {
        MessageBox.Show("Please select the data you want to update");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "update CategoryTbl set CatName = @CatName, CatDesc = @CatDesc WHERE Catd = @Catd";
        SqlCommand update = new SqlCommand(query, conn);
        update.Parameters.AddWithValue("@Catd", int.Parse(CatIdTb.Text));
        update.Parameters.AddWithValue("@CatName", CatNameTb.Text);
        update.Parameters.AddWithValue("@CatDesc", CatDescTb.Text);

        try
        {
            conn.Open();
            update.ExecuteNonQuery();
            MessageBox.Show("Data berhasil Update");
        }
        catch
        {
            MessageBox.Show("System Error");
        }
        finally
        {
            conn.Close();
            showdata();
        }
    }
}
```

The code above is the code on the edit button which functions to update data if there is an input error. The first if statement serves to check the textbox "CatIdTb", "CatNameTb", "CatDescTb" if the textbox is empty then a messagebox will come out "Please select the data you want to update". SqlConnection is the start of a query to the database. Stores the SQL query that will be used to update the data in the "CategoryTbl" table. The String Query above will change the value of the "CatName" column to the value of CatNameTb.Text, the value of the "CatDesc" column to the value of CatDescTb.Text, and update the row that has the value of the "Catd" column according to the value of CatIdTb.Text. SqlCommand has the function in the previous form. Then in the "update parameters" section, add the @Catd parameter to the SqlCommand object with the value of CatIdTb.Text. This ensures that the value of CatIdTb.Text will be used as the value to be updated in the query, Adds the @CatName parameter to the SqlCommand object with the value of CatNameTb.Text. This ensures that the value of CatNameTb.Text will be used as the value to be updated in the query, and Adds the @CatDesc parameter to the SqlCommand object with the value of CatDescTb.Text. This ensures that the value of CatDescTb.Text will be used as the value to be updated in the query. Then when the data is successfully updated, it will issue a messageBox "Data updated successfully". But if it doesn't work then "System error". When the code above is successfully executed, the data in the "DataGridView" will be updated directly through the "ShowData" function.

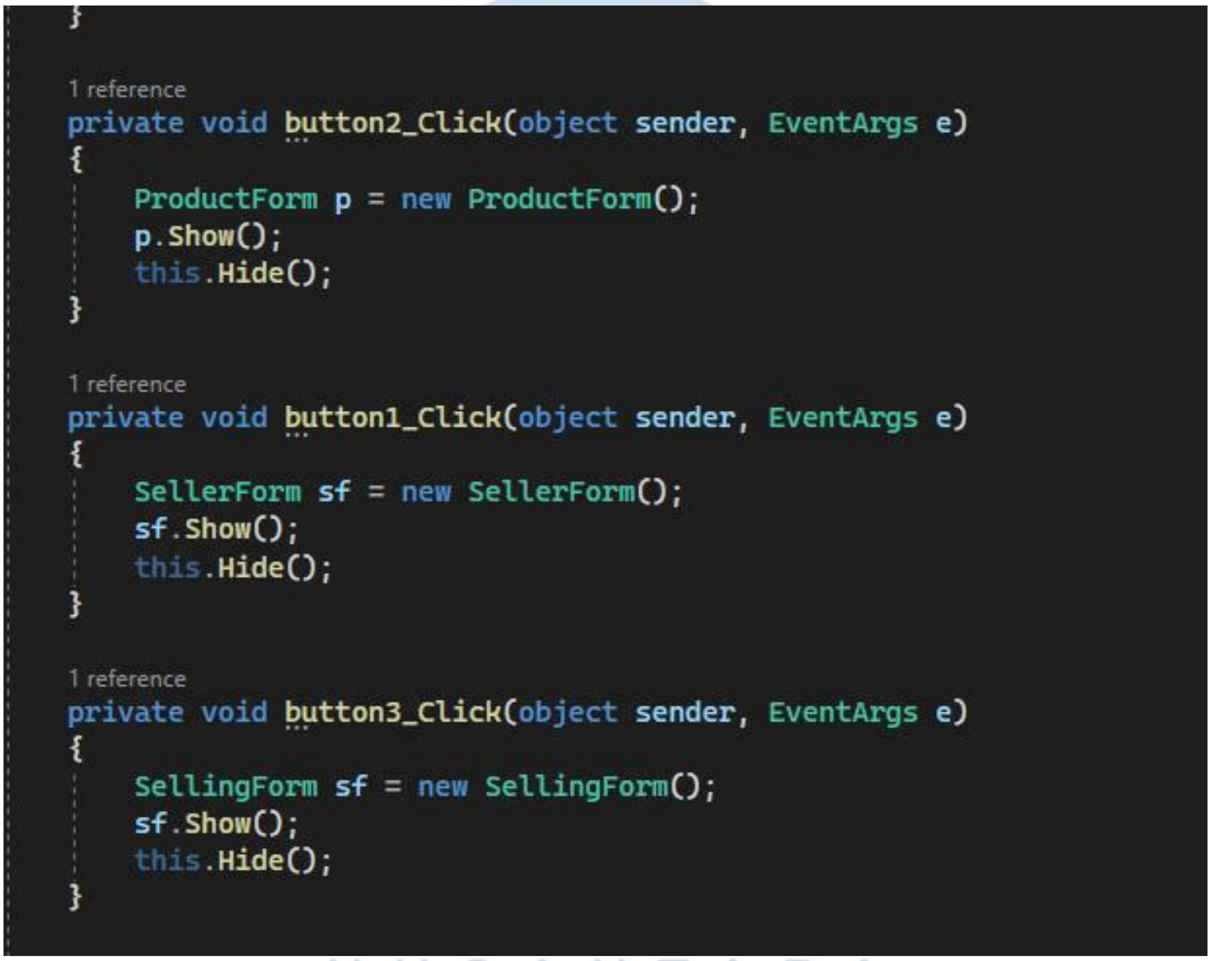

```
1 reference
private void button7_Click(object sender, EventArgs e)
{
    if (CatIdTb.Text == "")
    {
        MessageBox.Show("Please select the data you want to delete");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "delete CategoryTbl where Catd = @Catd";
        SqlCommand delete = new SqlCommand(query, conn);
        delete.Parameters.AddWithValue("@Catd", int.Parse(CatIdTb.Text));

        try
        {
            conn.Open();
            delete.ExecuteNonQuery();
            MessageBox.Show("Data berhasil dihapus");
        }
        catch
        {
            MessageBox.Show("System Error");
        }
        finally
        {
            conn.Close();
            showdata();
        }

    }
```

In the code above, the delete button has a function to delete the data in the "DataGridView". The first if statement checks whether the value of CatIdTb is empty. If it is empty, it will issue a messageBox "Please select the data you want to delete". Then the Query above has a function to delete data from the "CategoryTbl" table. This query will delete rows that have the value of the "Catd" column according to the value of CatIdTb.Text. Then in the "Delete parameters" section has a function to add the @Catd parameter to the SqlCommand object with the value of CatIdTb.Text. This ensures that the value of CatIdTb.Text will be used as the value to be deleted in the query. Then when a data is successfully deleted it will issue a messageBox "Data successfully deleted". However, if it is unsuccessful, it will issue a "System Error" messageBox. When the data is successfully deleted, the "DataGridView" will issue a new update data through the "showdata" function.



```
1 reference
private void button8_Click(object sender, EventArgs e)
{
    CatIdTb.Text = string.Empty;
    CatNameTb.Text = string.Empty;
    CatDescTb.Text = string.Empty;
}
```

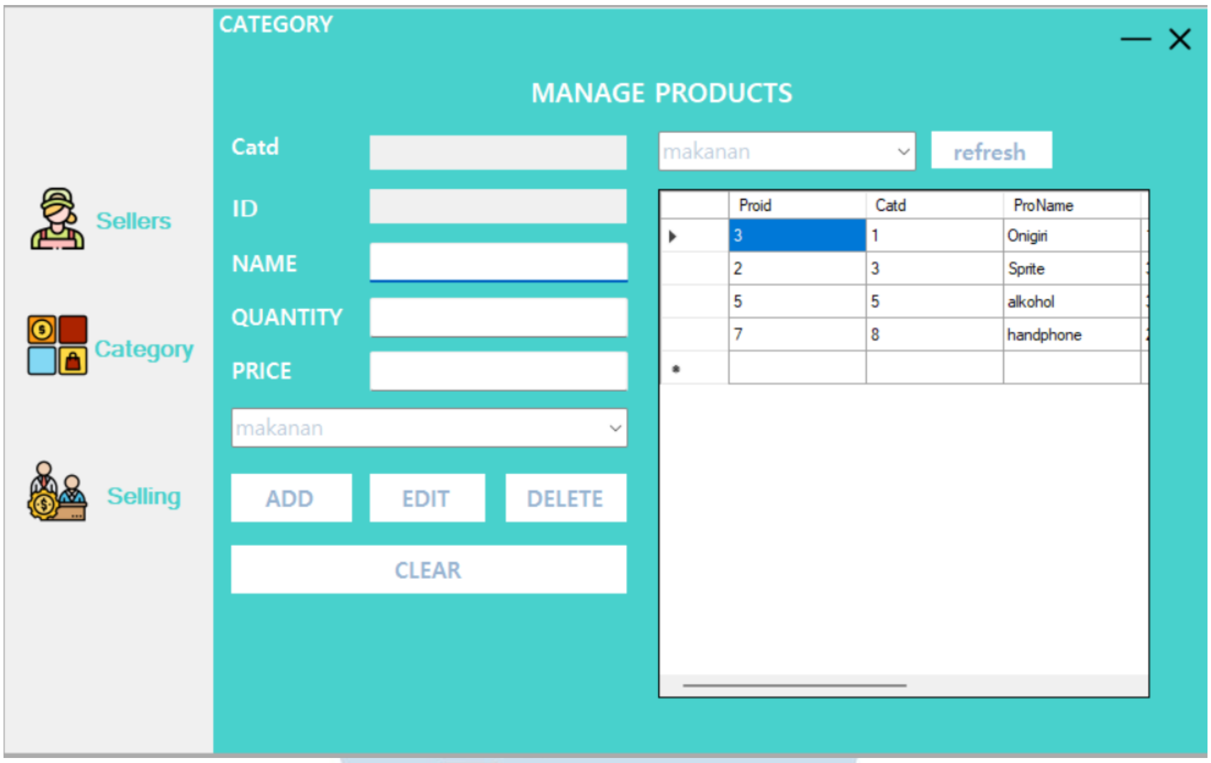
The code above is the code on the "Clear" button which functions to delete all the values in the textbox above.



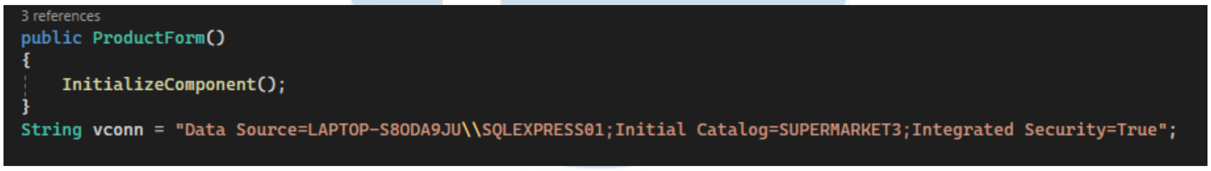
```
}

1 reference
private void button2_Click(object sender, EventArgs e)
{
    ProductForm p = new ProductForm();
    p.Show();
    this.Hide();
}

1 reference
private void button1_Click(object sender, EventArgs e)
{
    SellerForm sf = new SellerForm();
    sf.Show();
    this.Hide();
}

1 reference
private void button3_Click(object sender, EventArgs e)
{
    SellingForm sf = new SellingForm();
    sf.Show();
    this.Hide();
}
```

Then the 3 buttons above will lead to three different forms. button 1 will then go to ProductForm, button 2 goes to SellerForm, and button 3 goes to SellingForm.

PRODUCT FORM

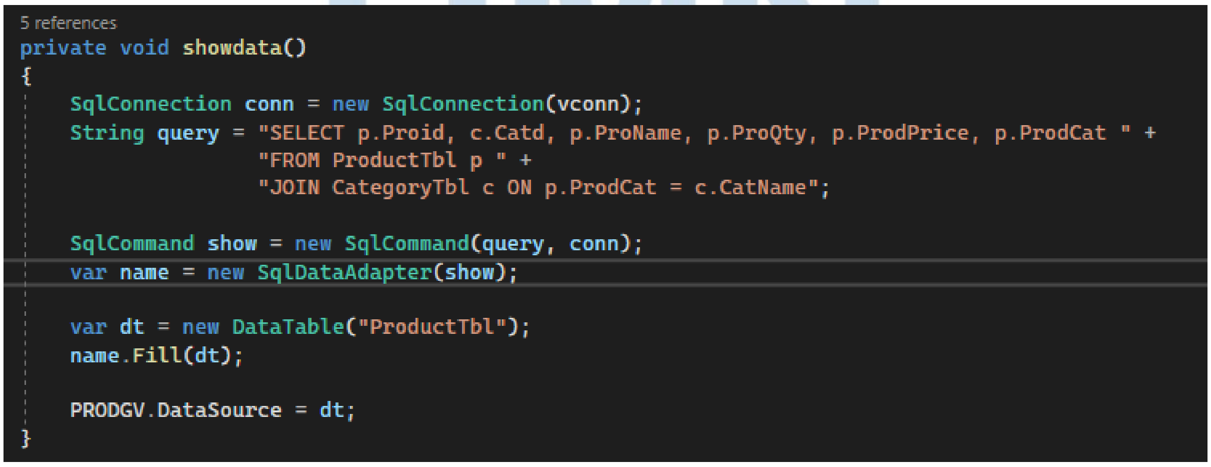


	Proid	Catd	ProName
▶	3	1	Onigiri
	2	3	Sprite
	5	5	alkohol
	7	8	handphone
*			



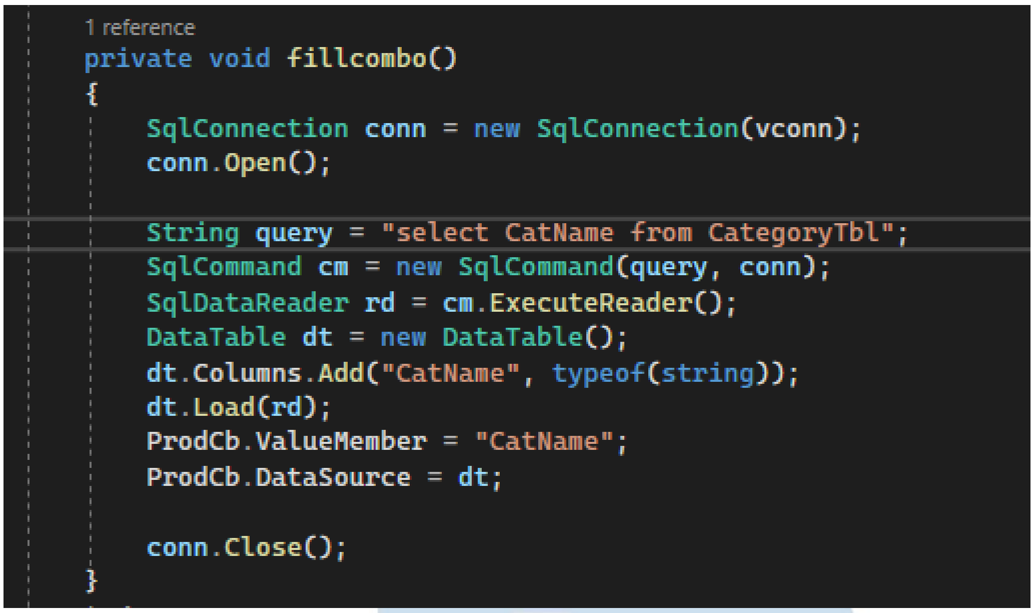
```
3 references
public ProductForm()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
```

The code above is the code to call the database that we use. In the code above the database we use is named "SUPERMARKET3".



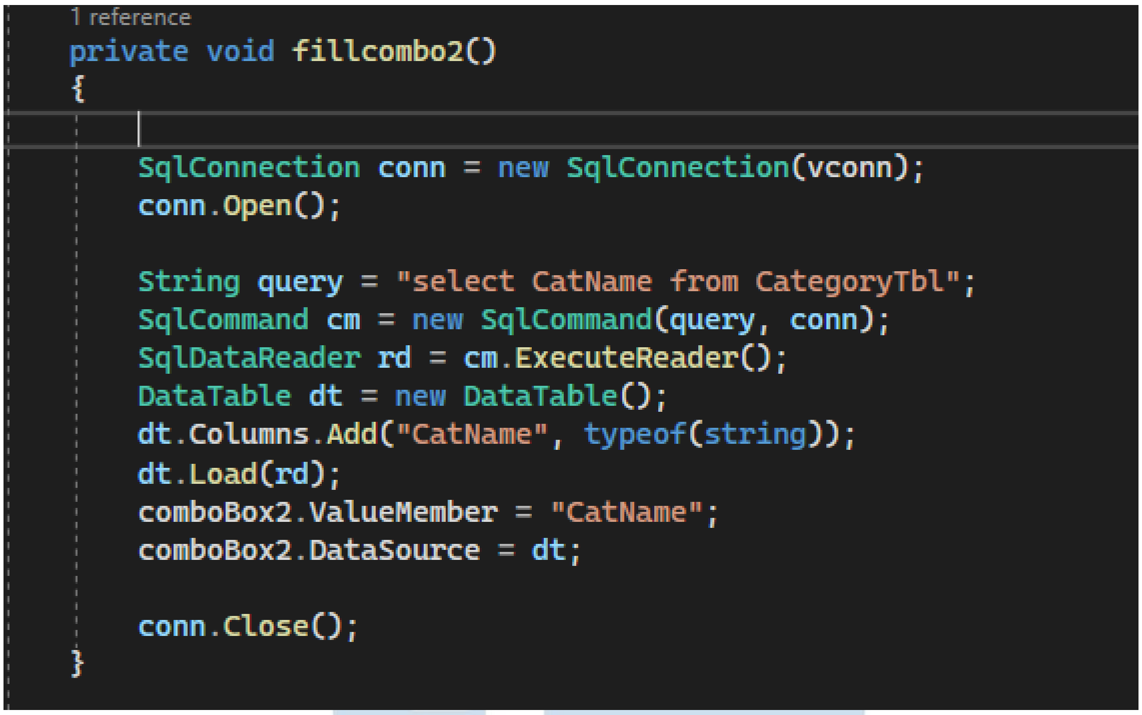
```
5 references
private void showdata()
{
    SqlConnection conn = new SqlConnection(vconn);
    String query = "SELECT p.Proid, c.Catd, p.ProName, p.ProQty, p.ProdPrice, p.ProdCat " +
                   "FROM ProductTbl p " +
                   "JOIN CategoryTbl c ON p.ProdCat = c.CatName";

    SqlCommand show = new SqlCommand(query, conn);
    var name = new SqlDataAdapter(show);

    var dt = new DataTable("ProductTbl");
    name.Fill(dt);

    PRODGV.DataSource = dt;
}
```

The code above has the same function as the class function in categoryform. The difference with CategoryForm is that the data shown in "DataGridView" is data with tables "ProductTbl" (proid, proname, proQty, ProdPrice) and "CategoryTbl" (catd, ProdCat).



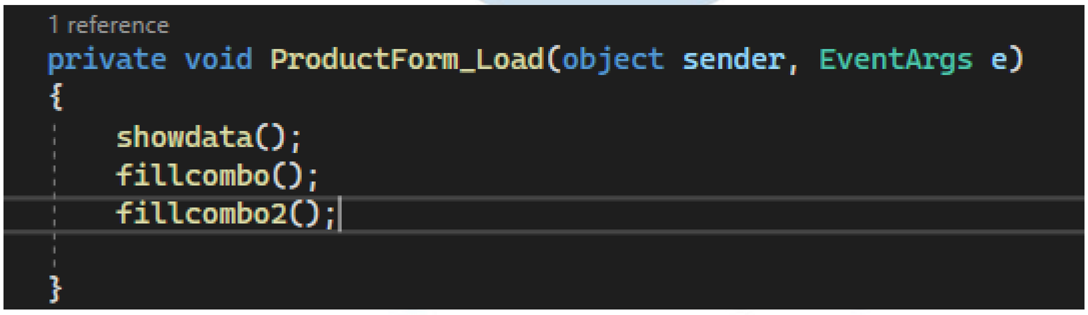
```
1 reference
private void fillcombo()
{
    SqlConnection conn = new SqlConnection(vconn);
    conn.Open();

    String query = "select CatName from CategoryTbl";
    SqlCommand cm = new SqlCommand(query, conn);
    SqlDataReader rd = cm.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Columns.Add("CatName", typeof(string));
    dt.Load(rd);
    ProdCb.ValueMember = "CatName";
    ProdCb.DataSource = dt;

    conn.Close();
}
```

The class above named "fillcombo" has a function to insert a value into a combobox with the name "ProdCb". The query above proves that the value entered into the combobox is "CatName" in the "CategoryTbl" table.SqlDataReader rd = cm.ExecuteReader()" Reads data generated by SQL commands using the SqlCommand object. "DataTable dt = new DataTable();" is useful for creating a DataTable object to store the data that will be bound to the combobox. "dt.Columns.Add("CatName", typeof(string));" useful for adding the "CatName" column to the DataTable object with the string data type. "dt.Load(rd);" is used to load data from the SqlDataReader object into the DataTable object. "ProdCb.ValueMember = "CatName";" is useful for setting the "CatName" column as the value to be selected from the "ProdCb" combobox. "ProdCb.DataSource = dt;" is useful for setting the DataTable object as the data source for the "ProdCb" combobox.



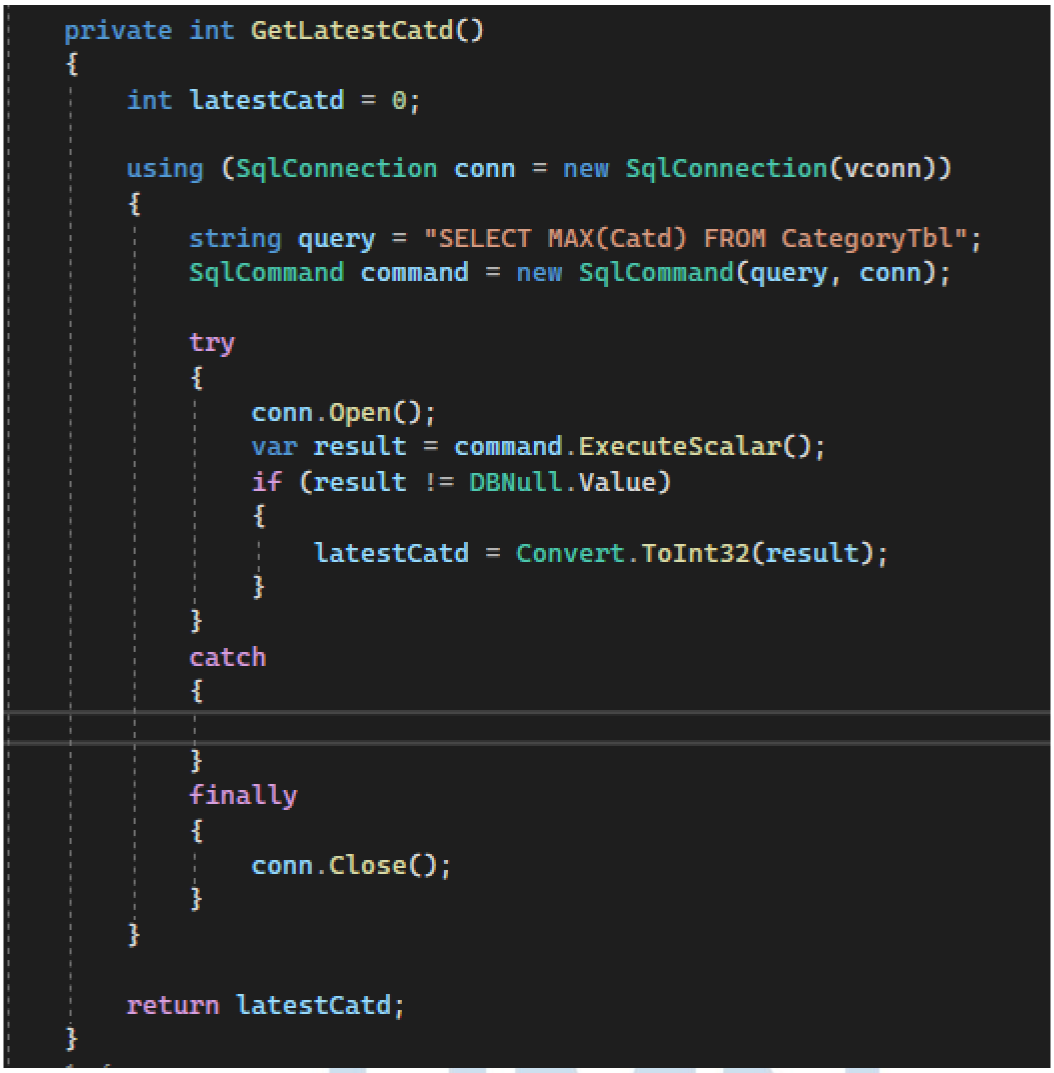
```
1 reference
private void fillcombo2()
{

    SqlConnection conn = new SqlConnection(vconn);
    conn.Open();

    String query = "select CatName from CategoryTbl";
    SqlCommand cm = new SqlCommand(query, conn);
    SqlDataReader rd = cm.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Columns.Add("CatName", typeof(string));
    dt.Load(rd);
    comboBox2.ValueMember = "CatName";
    comboBox2.DataSource = dt;

    conn.Close();
}
```

The code above has the same function as the code in "fillcombo". But what distinguishes it from other classes is that the value of the "CategoryTbl" table is directed to ComboBox2. In the code above, the value is directed to ComboBox 1.



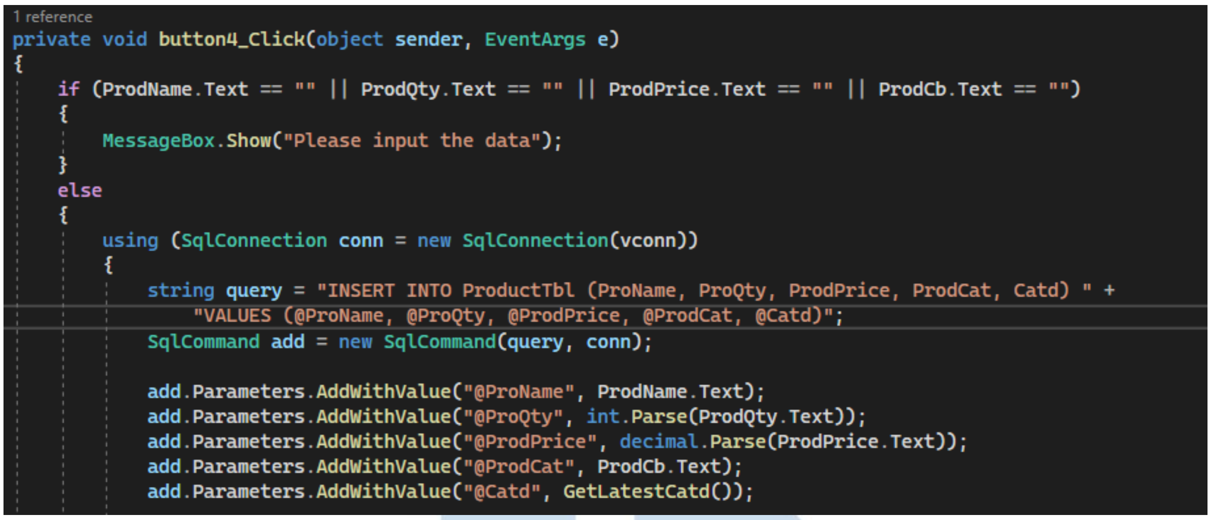
```
1 reference
private void ProductForm_Load(object sender, EventArgs e)
{
    showdata();
    fillcombo();
    fillcombo2();

}
```

The code above is a way for the data to be implemented into comboBox1, ComboBox2, and DataGridView. All classes that we have created are called into ProductForm

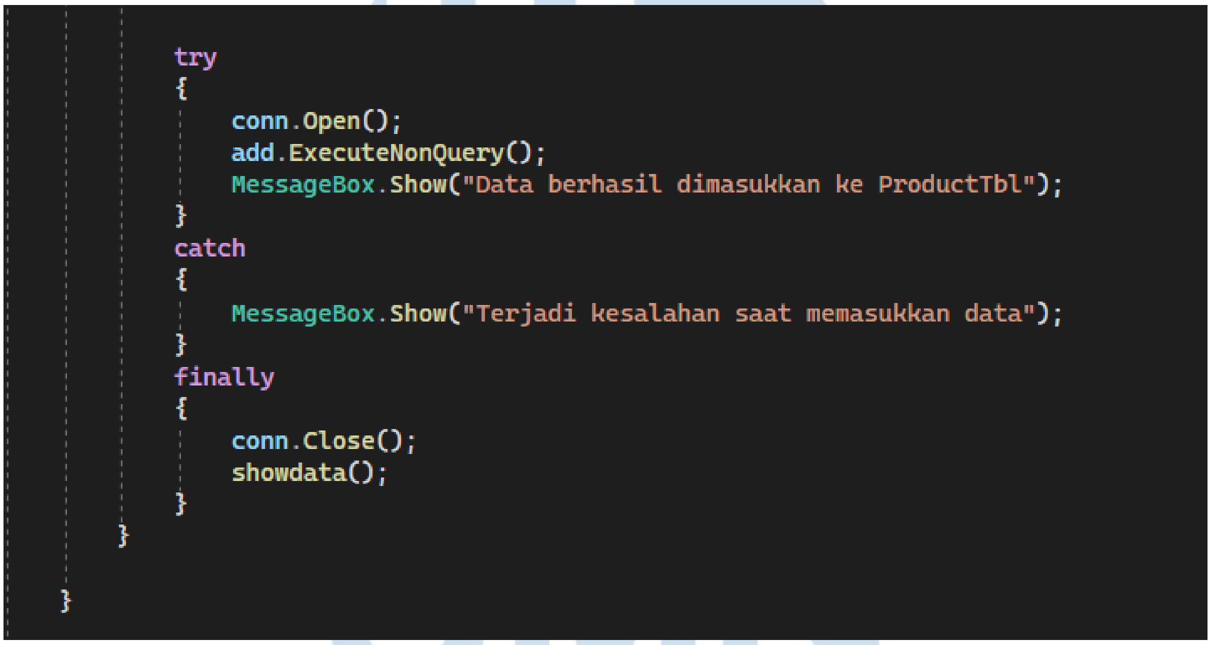


```
private int GetLatestCatd()
{
    int latestCatd = 0;

    using (SqlConnection conn = new SqlConnection(vconn))
    {
        string query = "SELECT MAX(Catd) FROM CategoryTbl";
        SqlCommand command = new SqlCommand(query, conn);

        try
        {
            conn.Open();
            var result = command.ExecuteScalar();
            if (result != DBNull.Value)
            {
                latestCatd = Convert.ToInt32(result);
            }
        }
        catch
        {

        }
        finally
        {
            conn.Close();
        }
    }

    return latestCatd;
}
```

The code above has a class function to retrieve the value in the CategoryTbl table in the "Catd" column so that the data can be foreign keyed into the "ProductForm" table data. So that the overall function above is to take the value in the CategoryTbl table so that it can enter ProductForm. "var result = command.ExecuteScalar();" serves to execute the SQL command and get a single result from the MAX aggregate function using the ExecuteScalar() method. This result will be stored in the result variable. "if (result != DBNull.Value)" serves to check whether the result is not DBNull.Value. If it is not, it means that there is a largest value found. "latestCatd = Convert.ToInt32(result);" serves to convert the result into an integer data type and store it in the latestCatd variable. Because of the data type itself, a Catd is of type int. "return latestCatd;" serves to return the largest value from the "Catd" column that has been obtained.



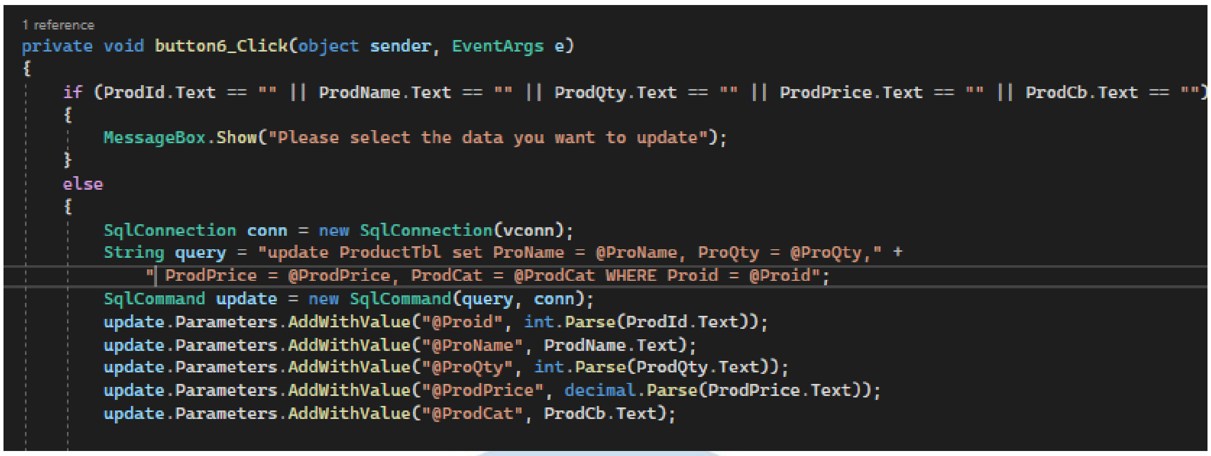
```
1 reference
private void button4_Click(object sender, EventArgs e)
{
    if (ProdName.Text == "" || ProdQty.Text == "" || ProdPrice.Text == "" || ProdCb.Text == "")
    {
        MessageBox.Show("Please input the data");
    }
    else
    {
        using (SqlConnection conn = new SqlConnection(vconn))
        {
            string query = "INSERT INTO ProductTbl (ProName, ProQty, ProdPrice, ProdCat, Catd) " +
                "VALUES (@ProName, @ProQty, @ProdPrice, @ProdCat, @Catd)";
            SqlCommand add = new SqlCommand(query, conn);

            add.Parameters.AddWithValue("@ProName", ProdName.Text);
            add.Parameters.AddWithValue("@ProQty", int.Parse(ProdQty.Text));
            add.Parameters.AddWithValue("@ProdPrice", decimal.Parse(ProdPrice.Text));
            add.Parameters.AddWithValue("@ProdCat", ProdCb.Text);
            add.Parameters.AddWithValue("@Catd", GetLatestCatd());
```



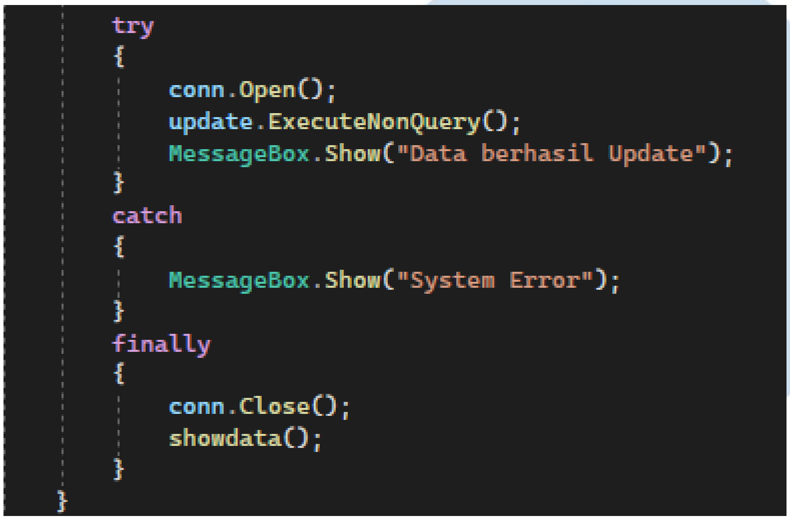
```
            try
            {
                conn.Open();
                add.ExecuteNonQuery();
                MessageBox.Show("Data berhasil dimasukkan ke ProductTbl");
            }
            catch
            {
                MessageBox.Show("Terjadi kesalahan saat memasukkan data");
            }
            finally
            {
                conn.Close();
                showdata();
            }
        }

    }
```

The code above is the code in the "add" button which has the same function as the add button in CategoryForm. The difference with CategoryForm is :

1. The value entered into the table is the ProductTbl table with columns ProName, ProQty, ProdPrice, ProdCat, Catd. Then the Catd column contains the value in the CategoryTbl table column Catd.
2. The data can enter the ProductTbl table because we have created the "GetLatestCatd" class function in the code above. So we insert the code into the Catd column.

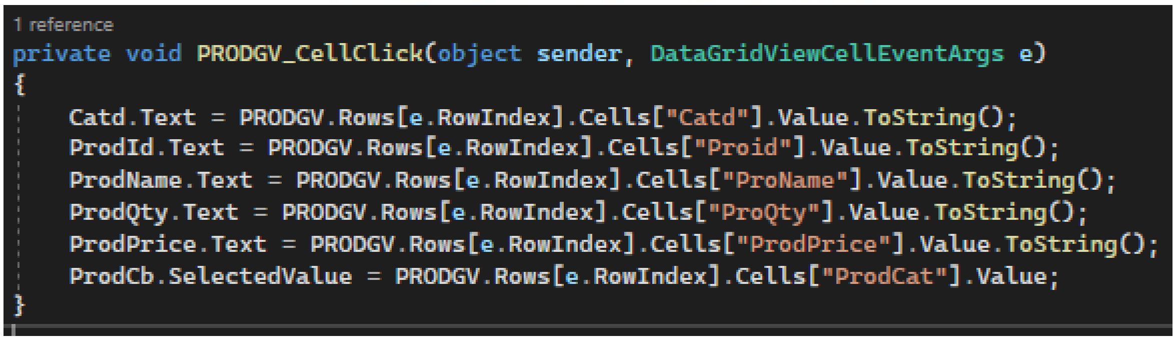


```
1 reference
private void button6_Click(object sender, EventArgs e)
{
    if (ProdId.Text == "" || ProdName.Text == "" || ProdQty.Text == "" || ProdPrice.Text == "" || ProdCb.Text == "")
    {
        MessageBox.Show("Please select the data you want to update");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "update ProductTbl set ProName = @ProName, ProQty = @ProQty," +
            "| ProdPrice = @ProdPrice, ProdCat = @ProdCat WHERE Proid = @Proid";
        SqlCommand update = new SqlCommand(query, conn);
        update.Parameters.AddWithValue("@Proid", int.Parse(ProdId.Text));
        update.Parameters.AddWithValue("@ProName", ProdName.Text);
        update.Parameters.AddWithValue("@ProQty", int.Parse(ProdQty.Text));
        update.Parameters.AddWithValue("@ProdPrice", decimal.Parse(ProdPrice.Text));
        update.Parameters.AddWithValue("@ProdCat", ProdCb.Text);
```



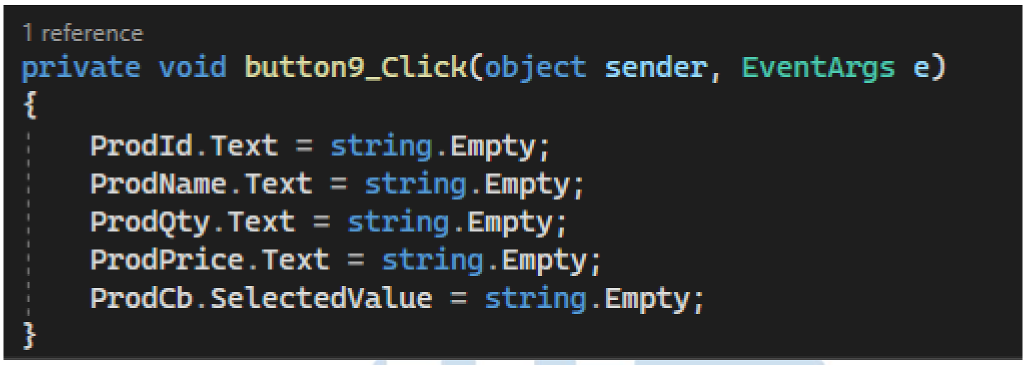
```
        try
        {
            conn.Open();
            update.ExecuteNonQuery();
            MessageBox.Show("Data berhasil Update");
        }
        catch
        {
            MessageBox.Show("System Error");
        }
        finally
        {
            conn.Close();
            showdata();
        }
    }
```

The code above is the code in the "Edit" button which has the same function as the edit button in CategoryForm. The difference with CategoryForm is that the value that is updated into the table is the ProductTbl table with columns ProName, ProQty, ProdPrice, ProdCat, Catd. If a data is successfully updated, it will issue a messageBox "Data Successfully updated" and if it fails, it will issue a messageBox "System Error". If a data is successfully updated, the data in "DataGridView" will immediately show the updated data with the class in "showdata".



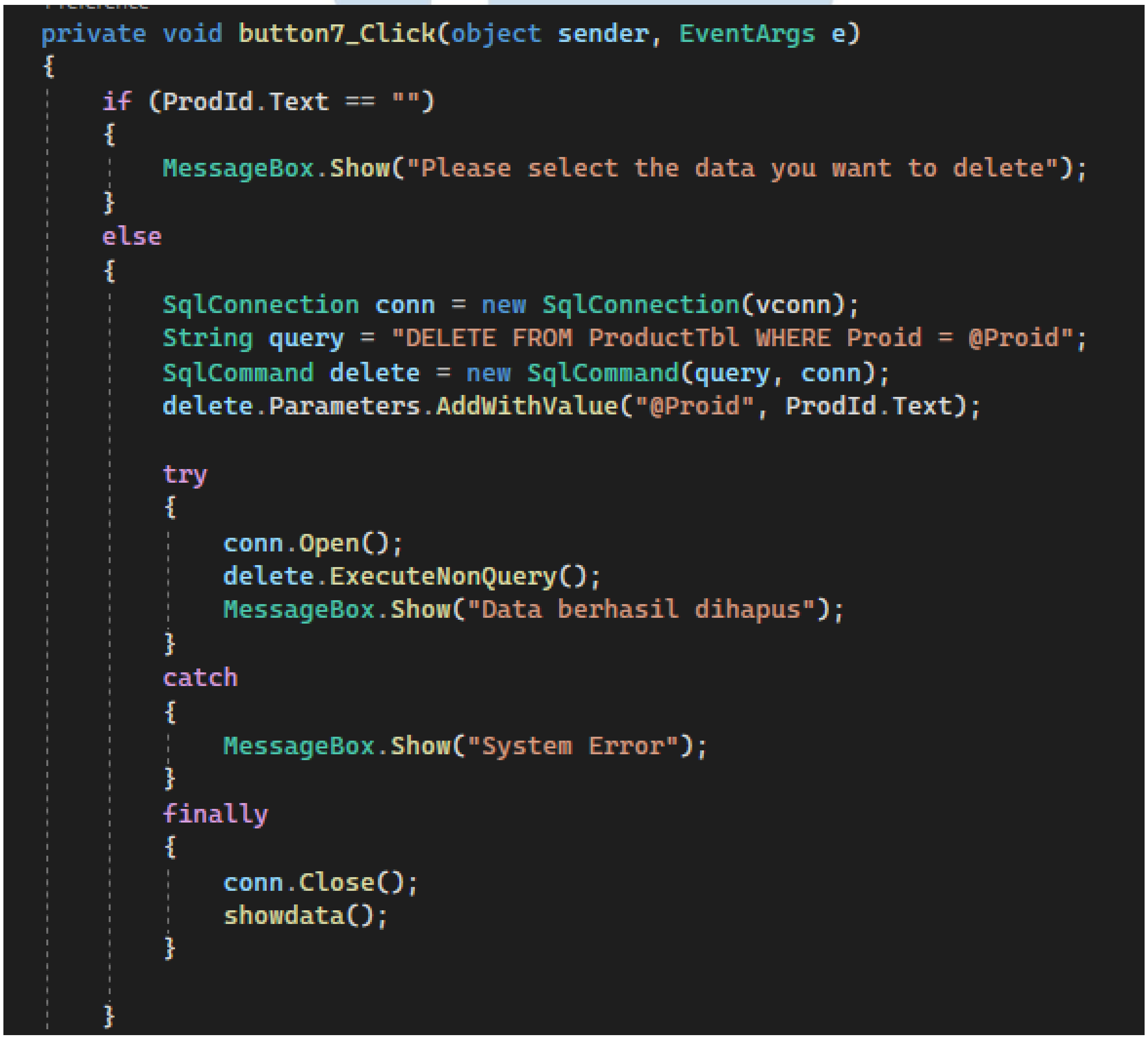
```
1 reference
private void PRODGV_CellClick(object sender, DataGridViewCellEventArgs e)
{
    Catd.Text = PRODGV.Rows[e.RowIndex].Cells["Catd"].Value.ToString();
    ProdId.Text = PRODGV.Rows[e.RowIndex].Cells["Proid"].Value.ToString();
    ProdName.Text = PRODGV.Rows[e.RowIndex].Cells["ProName"].Value.ToString();
    ProdQty.Text = PRODGV.Rows[e.RowIndex].Cells["ProQty"].Value.ToString();
    ProdPrice.Text = PRODGV.Rows[e.RowIndex].Cells["ProdPrice"].Value.ToString();
    ProdCb.SelectedValue = PRODGV.Rows[e.RowIndex].Cells["ProdCat"].Value;
}
```

The code above is something in the "DataGridView" in the above code. When we click the right row, the data in the row and table will appear in the textbox. The textbox matches the code above. Where there are 6 textboxes that will output data from the "DataGridView".



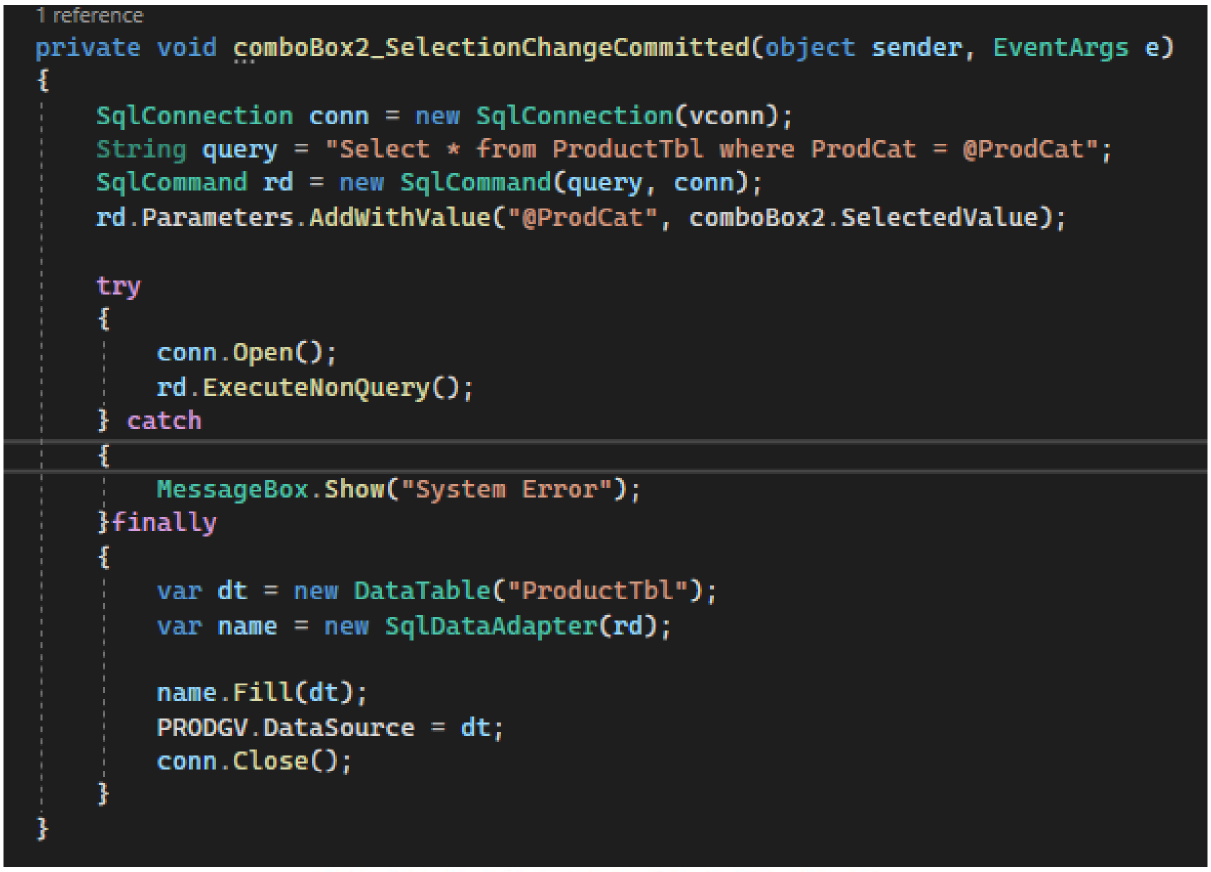
```
1 reference
private void button9_Click(object sender, EventArgs e)
{
    ProdId.Text = string.Empty;
    ProdName.Text = string.Empty;
    ProdQty.Text = string.Empty;
    ProdPrice.Text = string.Empty;
    ProdCb.SelectedValue = string.Empty;
}
```

The code above has a function to empty every value in the textbox. Making it easier for users to update data and want to fill in new data because once the button is clicked, the value in the textbox will be immediately deleted.



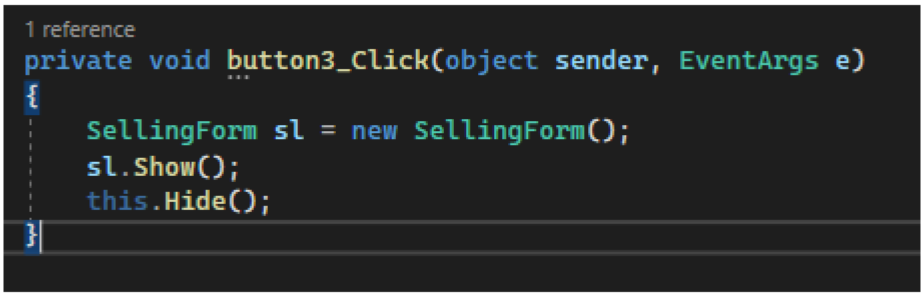
```
private void button7_Click(object sender, EventArgs e)
{
    if (ProdId.Text == "")
    {
        MessageBox.Show("Please select the data you want to delete");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "DELETE FROM ProductTbl WHERE Proid = @Proid";
        SqlCommand delete = new SqlCommand(query, conn);
        delete.Parameters.AddWithValue("@Proid", ProdId.Text);

        try
        {
            conn.Open();
            delete.ExecuteNonQuery();
            MessageBox.Show("Data berhasil dihapus");
        }
        catch
        {
            MessageBox.Show("System Error");
        }
        finally
        {
            conn.Close();
            showdata();
        }

    }
```

The code above is the code in the "Delete" button which has the same function as the Delete button in CategoryForm. What distinguishes it from CategoryForm is that the value that is deleted into the table is the ProductTbl table with columns ProName, ProQty, ProdPrice, ProdCat, Catd. If a data is successfully kdeleted it will issue a messageBox "Data Deleted Successfully" and if it fails it will issue a messageBox "System Error". If a data is successfully deleted then the data in "DataGridView" will immediately show updated data with the class in "showdata".

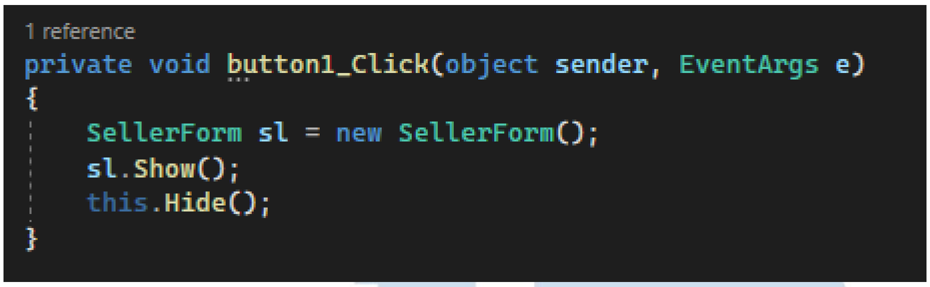


```
1 reference
private void comboBox2_SelectionChangeCommitted(object sender, EventArgs e)
{
    SqlConnection conn = new SqlConnection(vconn);
    String query = "Select * from ProductTbl where ProdCat = @ProdCat";
    SqlCommand rd = new SqlCommand(query, conn);
    rd.Parameters.AddWithValue("@ProdCat", comboBox2.SelectedValue);

    try
    {
        conn.Open();
        rd.ExecuteNonQuery();
    } catch
    {
        MessageBox.Show("System Error");
    }finally
    {
        var dt = new DataTable("ProductTbl");
        var name = new SqlDataAdapter(rd);

        name.Fill(dt);
        PRODGV.DataSource = dt;
        conn.Close();
    }
}
```

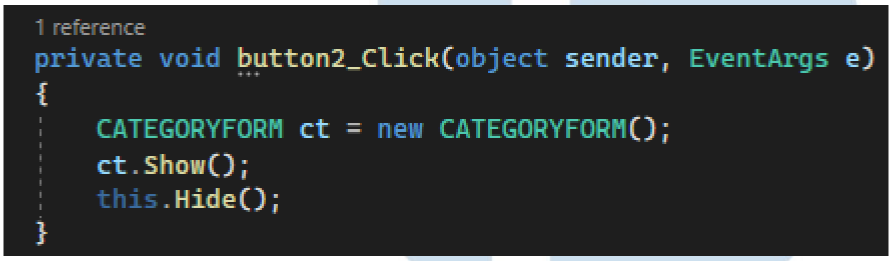
The comboBox2_SelectionChangeCommitted method will be executed when the selection in comboBox2 changes. this method connects to the database, executes an SQL query with parameters derived from comboBox2's selected values, and populates a DataTable object with the resulting data. The data is then displayed in the PRODGV control grid. Then the data in the ComboBox will always be updated according to what is input by the admin.



```
1 reference
private void button3_Click(object sender, EventArgs e)
{
    SellingForm sl = new SellingForm();
    sl.Show();
    this.Hide();
}
```



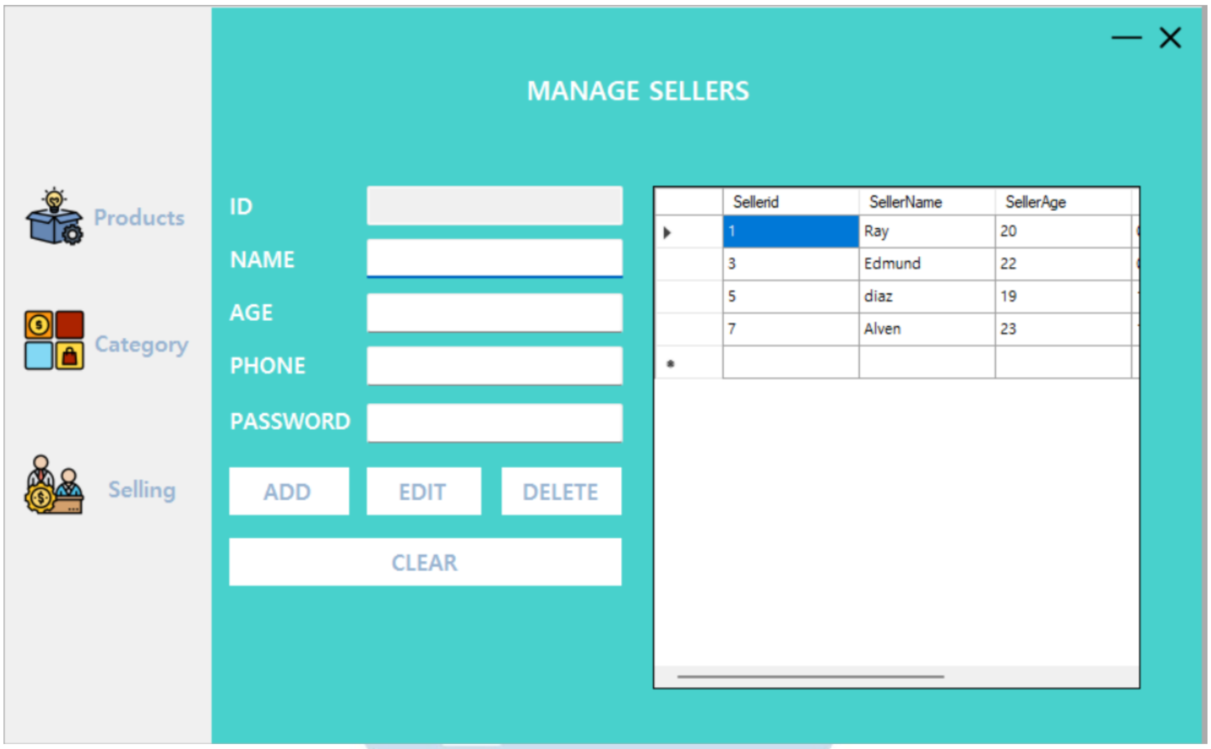
```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    SellerForm sl = new SellerForm();
    sl.Show();
    this.Hide();
}
```



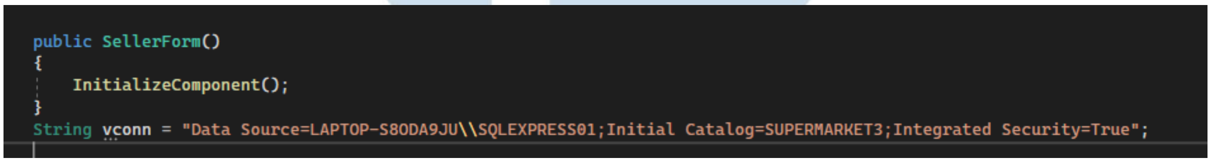
```
1 reference
private void button2_Click(object sender, EventArgs e)
{
    CATEGORYFORM ct = new CATEGORYFORM();
    ct.Show();
    this.Hide();
}
```

In the code above there are 3 buttons that lead to different forms according to the code. Button 1 will lead to the "SellingForm" form, the second button will lead to the "SellerForm" form, and the third button will lead to the "CategoryForm" form.

SELLER FORM

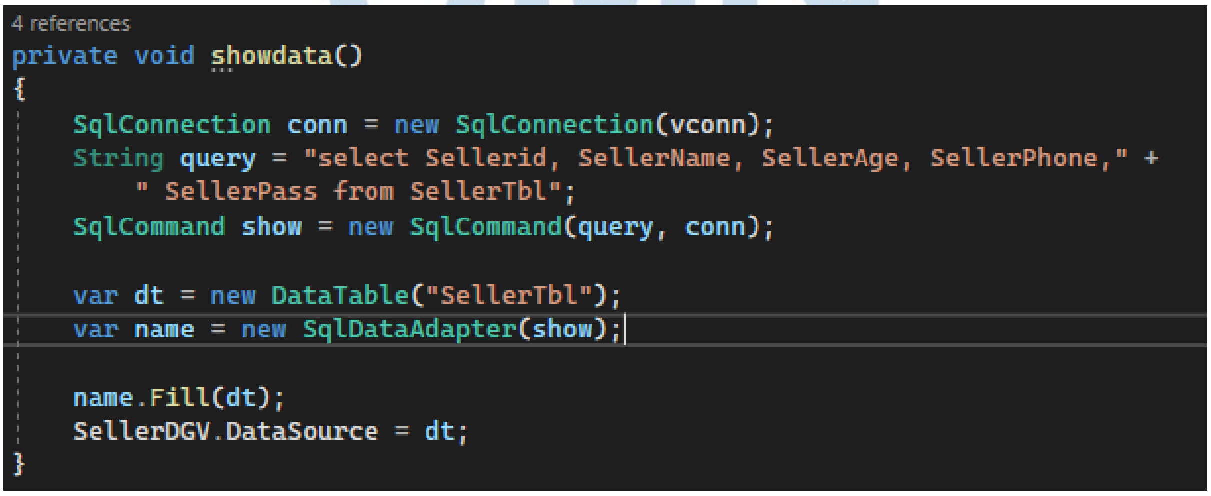


	Sellerid	SellerName	SellerAge
▸	1	Ray	20
	3	Edmund	22
	5	diaz	19
	7	Alven	23
*			



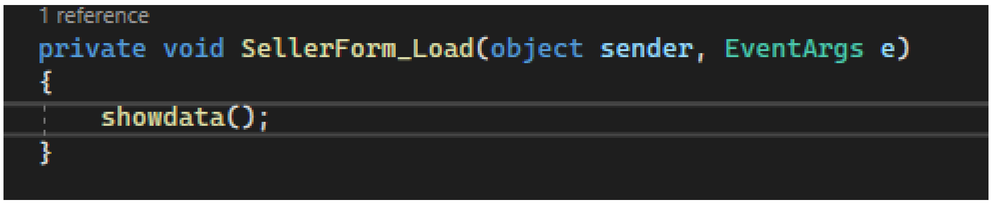
```
public SellerForm()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
```

The code above is the code to call the database that we use. In the code above the database we use is named "SUPERMARKET3".



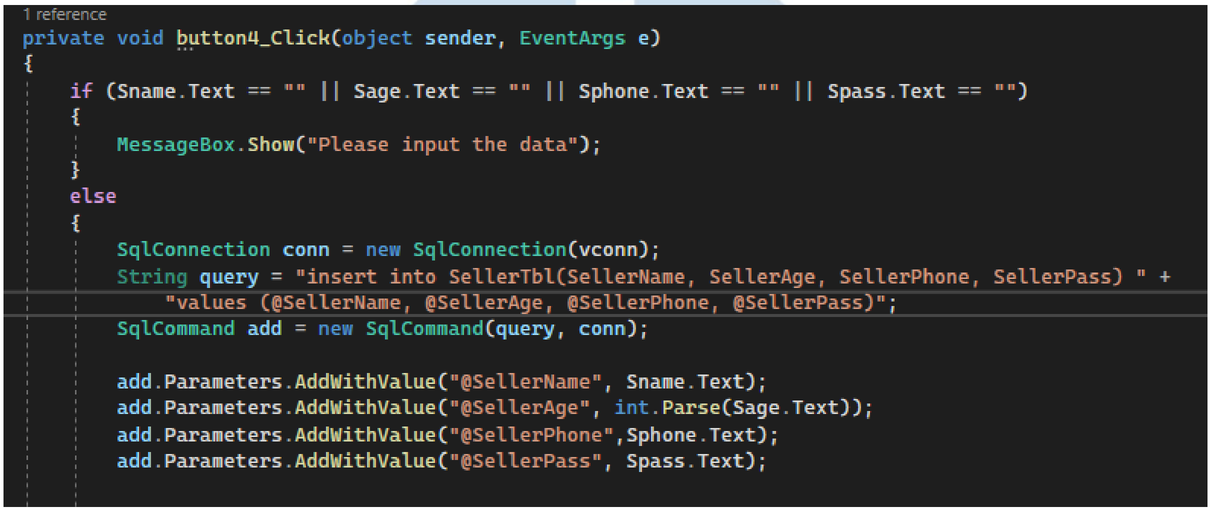
```
4 references
private void showdata()
{
    SqlConnection conn = new SqlConnection(vconn);
    String query = "select Sellerid, SellerName, SellerAge, SellerPhone," +
        " SellerPass from SellerTbl";
    SqlCommand show = new SqlCommand(query, conn);

    var dt = new DataTable("SellerTbl");
    var name = new SqlDataAdapter(show);

    name.Fill(dt);
    SellerDGV.DataSource = dt;
}
```

The showdata function above has the same function as the functions in the Category and Product forms. However, what distinguishes it from other forms is that the data displayed in the "DataGridView" is the data in the SellerTbl table (Sellerid, SellerName, SellerAge, SellerPhone).

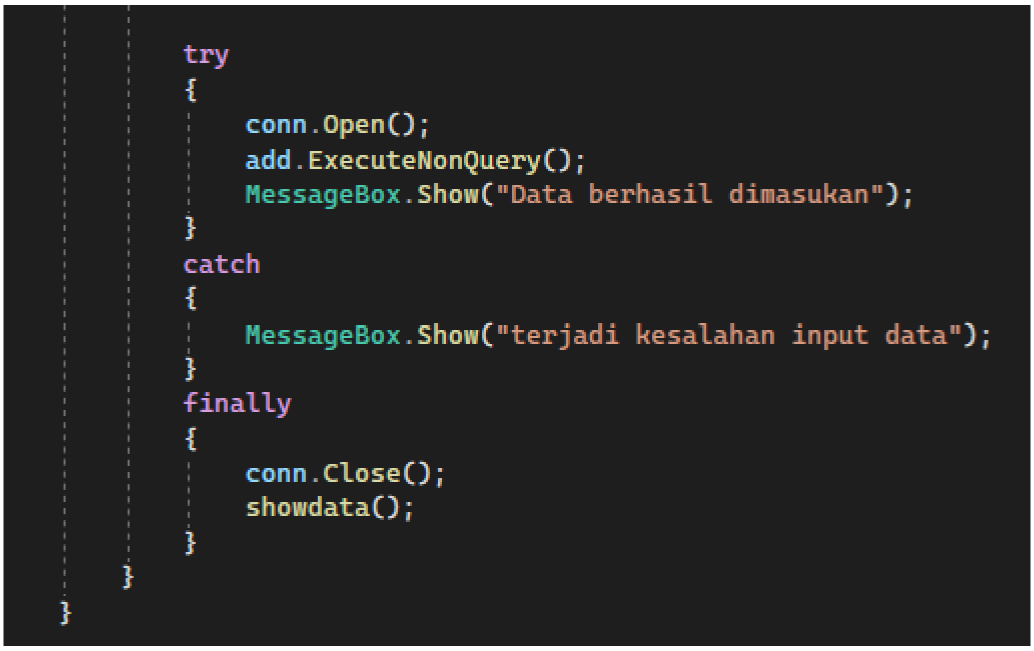


```
1 reference
private void SellerForm_Load(object sender, EventArgs e)
{
    showdata();
}
```

When the class is successfully created, we enter it into SellerForm_load so that the data can exit into the "DataGridView".

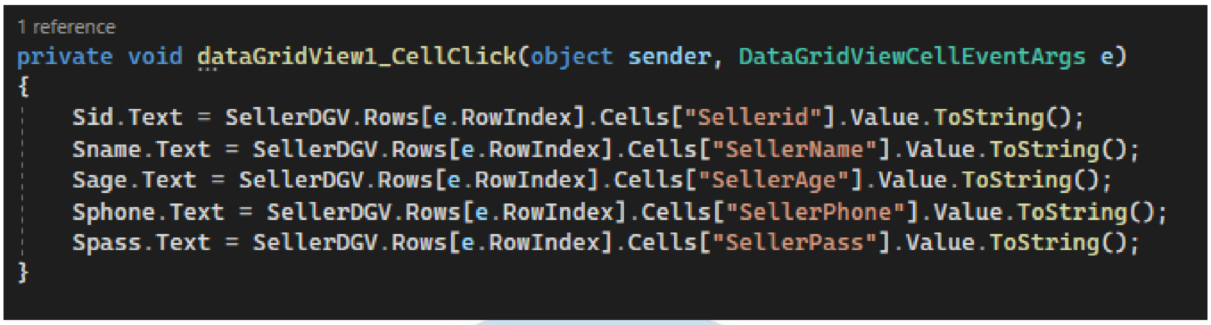


```
1 reference
private void button4_Click(object sender, EventArgs e)
{
    if (Sname.Text == "" || Sage.Text == "" || Sphone.Text == "" || Spass.Text == "")
    {
        MessageBox.Show("Please input the data");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "insert into SellerTbl(SellerName, SellerAge, SellerPhone, SellerPass) " +
            "values (@SellerName, @SellerAge, @SellerPhone, @SellerPass)";
        SqlCommand add = new SqlCommand(query, conn);

        add.Parameters.AddWithValue("@SellerName", Sname.Text);
        add.Parameters.AddWithValue("@SellerAge", int.Parse(Sage.Text));
        add.Parameters.AddWithValue("@SellerPhone",Sphone.Text);
        add.Parameters.AddWithValue("@SellerPass", Spass.Text);
```



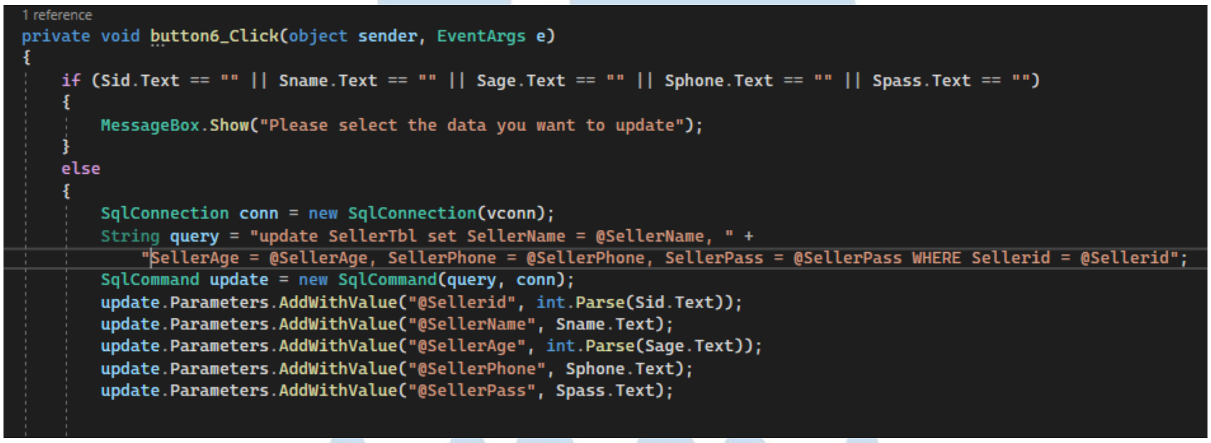
```
        try
        {
            conn.Open();
            add.ExecuteNonQuery();
            MessageBox.Show("Data berhasil dimasukan");
        }
        catch
        {
            MessageBox.Show("terjadi kesalahan input data");
        }
        finally
        {
            conn.Close();
            showdata();
        }
    }
}
```

The showdata function above has the same function as the functions in the Category and Product forms. However, what distinguishes it from other forms is that the data displayed in the "DataGridView" is the data in the SellerTbl table (Sellerid, SellerName, SellerAge, SellerPhone).

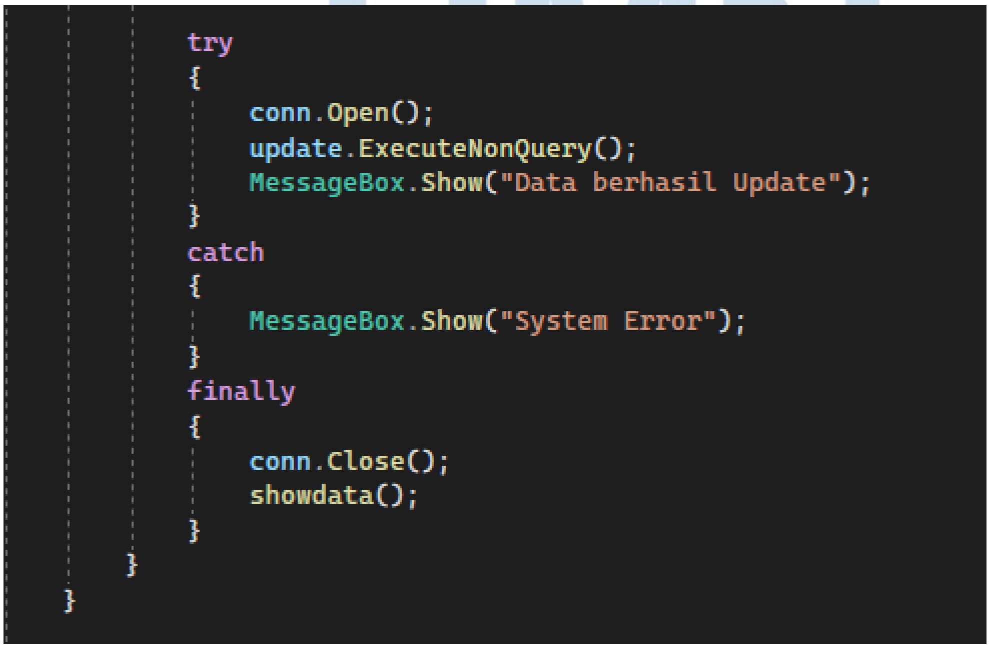


```
1 reference
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    Sid.Text = SellerDGV.Rows[e.RowIndex].Cells["Sellerid"].Value.ToString();
    Sname.Text = SellerDGV.Rows[e.RowIndex].Cells["SellerName"].Value.ToString();
    Sage.Text = SellerDGV.Rows[e.RowIndex].Cells["SellerAge"].Value.ToString();
    Sphone.Text = SellerDGV.Rows[e.RowIndex].Cells["SellerPhone"].Value.ToString();
    Spass.Text = SellerDGV.Rows[e.RowIndex].Cells["SellerPass"].Value.ToString();
}
```

The code above has a function to enter the value according to the row selected by the user into the TextBox. The value entered into the TextBox is the value in the selected table row.

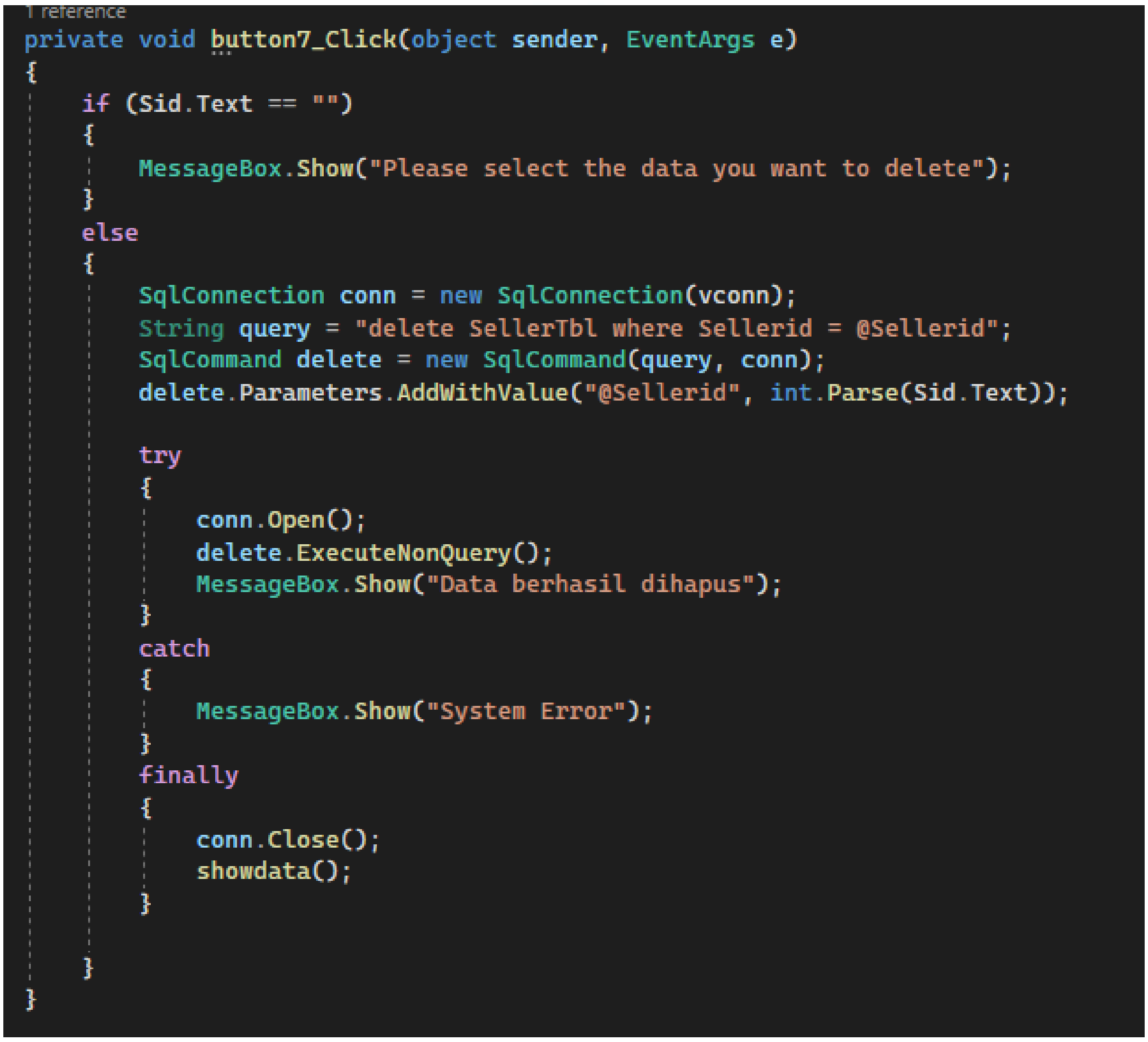


```
1 reference
private void button6_Click(object sender, EventArgs e)
{
    if (Sid.Text == "" || Sname.Text == "" || Sage.Text == "" || Sphone.Text == "" || Spass.Text == "")
    {
        MessageBox.Show("Please select the data you want to update");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "update SellerTbl set SellerName = @SellerName, " +
            "SellerAge = @SellerAge, SellerPhone = @SellerPhone, SellerPass = @SellerPass WHERE Sellerid = @Sellerid";
        SqlCommand update = new SqlCommand(query, conn);
        update.Parameters.AddWithValue("@Sellerid", int.Parse(Sid.Text));
        update.Parameters.AddWithValue("@SellerName", Sname.Text);
        update.Parameters.AddWithValue("@SellerAge", int.Parse(Sage.Text));
        update.Parameters.AddWithValue("@SellerPhone", Sphone.Text);
        update.Parameters.AddWithValue("@SellerPass", Spass.Text);
```



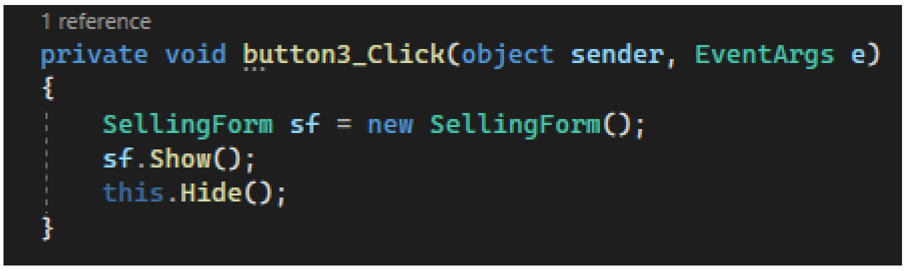
```
            try
            {
                conn.Open();
                update.ExecuteNonQuery();
                MessageBox.Show("Data berhasil Update");
            }
            catch
            {
                MessageBox.Show("System Error");
            }
            finally
            {
                conn.Close();
                showdata();
            }
        }
    }
```

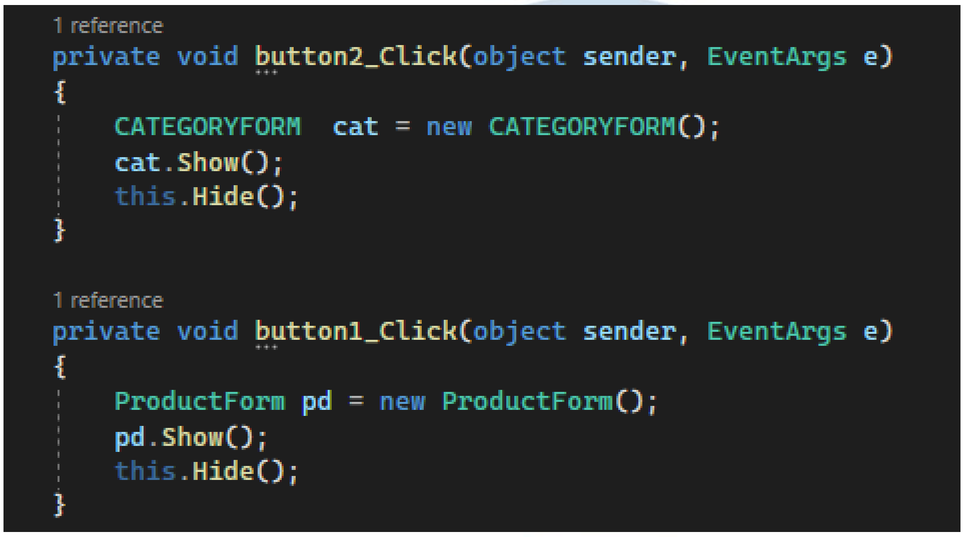
The code above is the code in the "Edit" button which has the same function as the edit button in CategoryForm and ProductForm. What differs from CategoryForm is that the value that is updated to the table is the SellerTbl table with columns SellerName, SellerAge, SellerPhone, SellerPass. If the data is successfully updated, it will issue a messageBox "Data successfully updated" and if it fails, it will issue a messageBox "System Error". If a data is successfully updated, the data in "DataGridView" will immediately show updated data with the class in "showdata".



```
1 reference
private void button7_Click(object sender, EventArgs e)
{
    if (Sid.Text == "")
    {
        MessageBox.Show("Please select the data you want to delete");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "delete SellerTbl where Sellerid = @Sellerid";
        SqlCommand delete = new SqlCommand(query, conn);
        delete.Parameters.AddWithValue("@Sellerid", int.Parse(Sid.Text));

        try
        {
            conn.Open();
            delete.ExecuteNonQuery();
            MessageBox.Show("Data berhasil dihapus");
        }
        catch
        {
            MessageBox.Show("System Error");
        }
        finally
        {
            conn.Close();
            showdata();
        }

    }
}
```

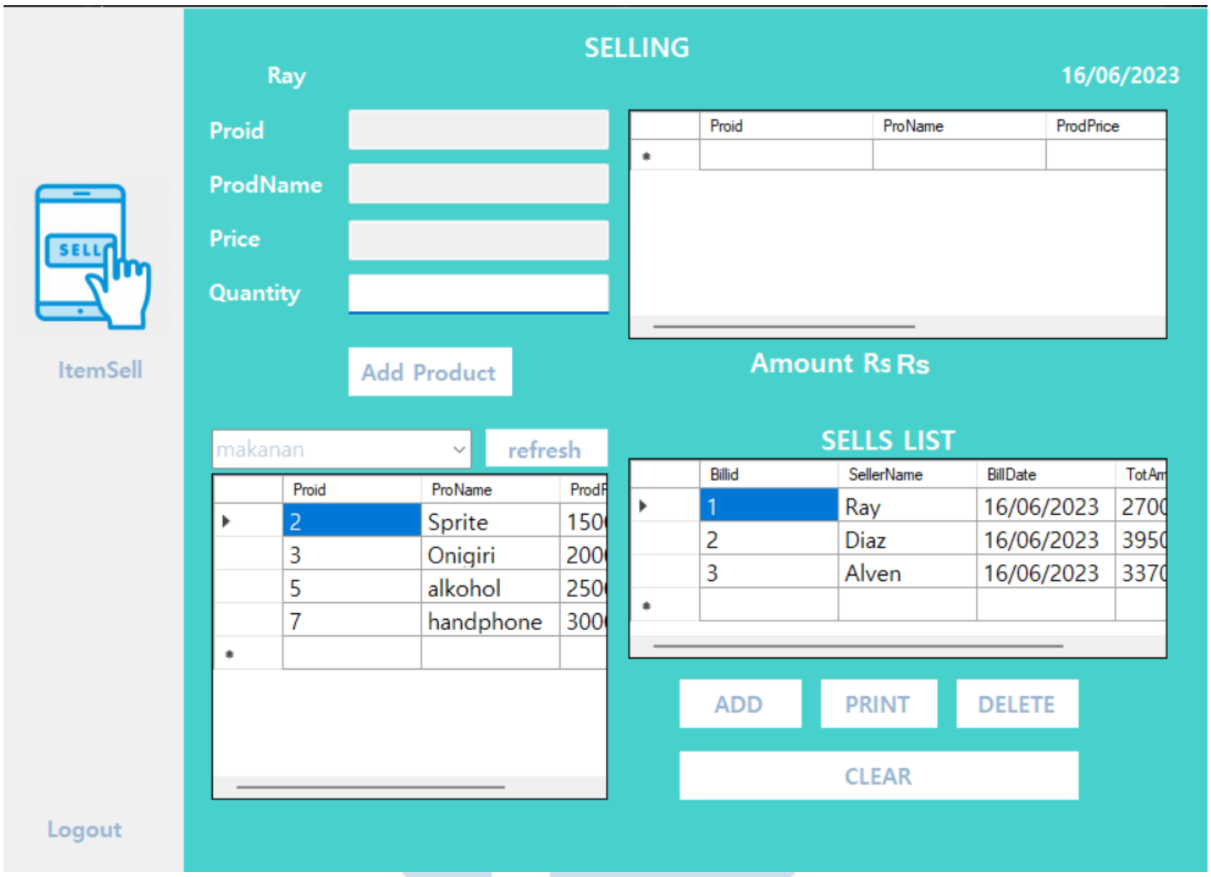
Pada code diatas merupakan code yang ada pada button “Delete” yang memiliki fungsi yang sama dengan fungsi yang button Delete yang ada pada CategoryForm dan ProductForm. Yang membedakan dengan CategoryForm dan ProductForm adalah Nilai yang didelete kedalam tabel adalah tabel SellerTbl dengan column ProName, ProQty, ProdPrice, ProdCat, Catd. Jika sebuah data berhasil kedelete maka akan mengeluarkan messageBox “Data Berhasil dihapus” dan jika gagal akan mengeluarkan messageBox “System Error”. Jika sebuah data berhasil ke delete maka data yang ada pada “DataGridView” akan langsung menunjukkan data update dengan class yang ada pada “showdata”.


```
1 reference
private void button3_Click(object sender, EventArgs e)
{
    SellingForm sf = new SellingForm();
    sf.Show();
    this.Hide();
}
```


```
1 reference
private void button2_Click(object sender, EventArgs e)
{
    CATEGORYFORM  cat = new CATEGORYFORM();
    cat.Show();
    this.Hide();
}

1 reference
private void button1_Click(object sender, EventArgs e)
{
    ProductForm pd = new ProductForm();
    pd.Show();
    this.Hide();
}
```

In the code above there are 3 buttons that lead to different forms according to the code. Button 1 will lead to the "SellingForm" form, the second button will lead to the "CategoryForm" form, and the third button will lead to the "ProductForm" form.

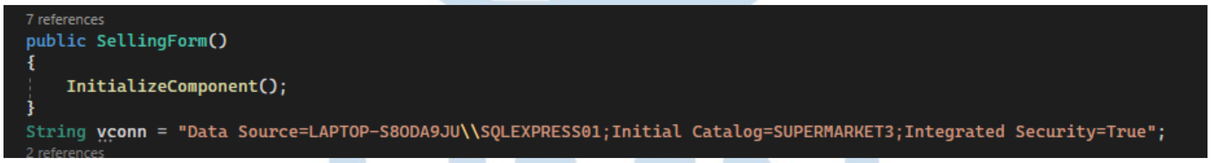
SELLING FORM



Proid	ProName	ProdPrice

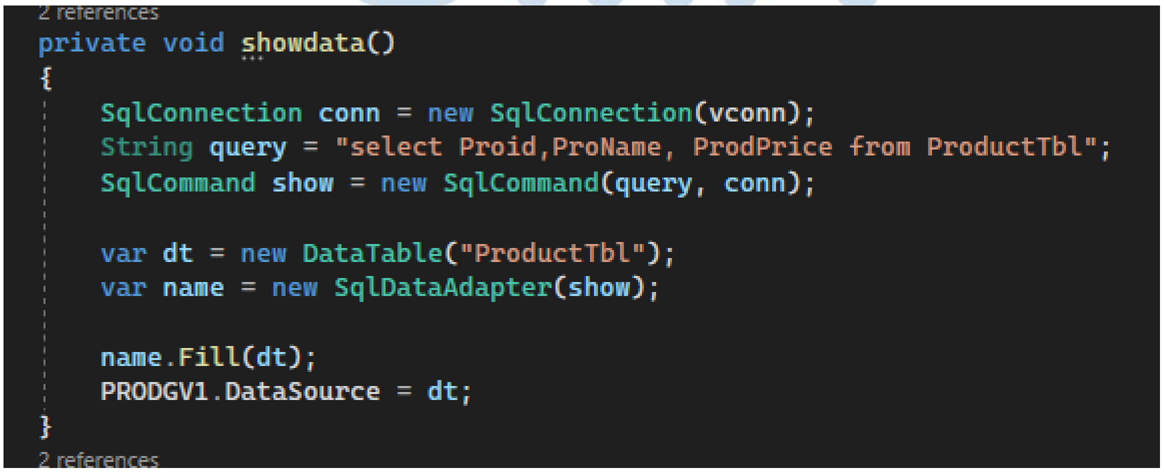
Proid	ProName	ProdP
2	Sprite	150
3	Onigiri	200
5	alkohol	250
7	handphone	300

Billid	SellerName	BillDate	TotAm
1	Ray	16/06/2023	270
2	Diaz	16/06/2023	395
3	Alven	16/06/2023	337



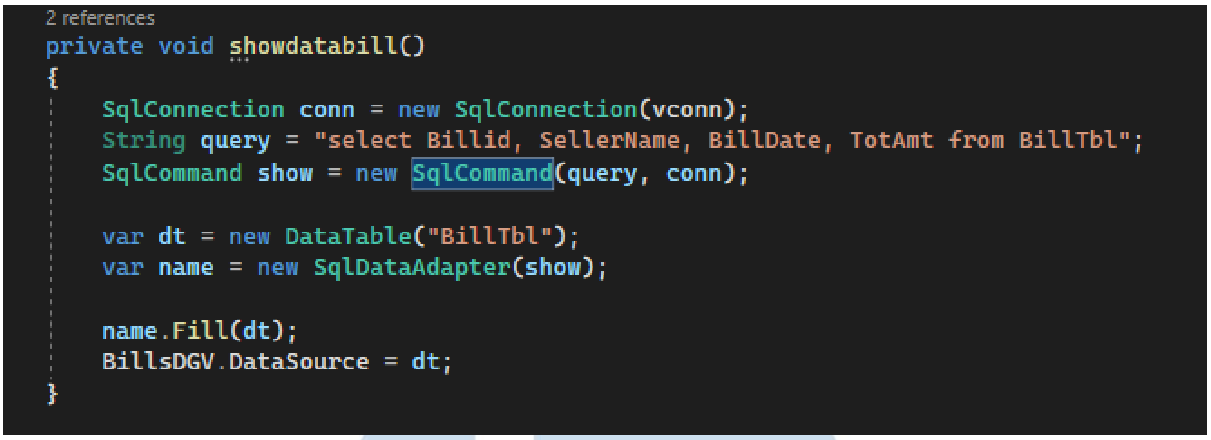
```
7 references
public SellingForm()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
2 references
```

The code above is the code to call the database that we use. In the code above, the database that we use is called "SUPERMARKET3".



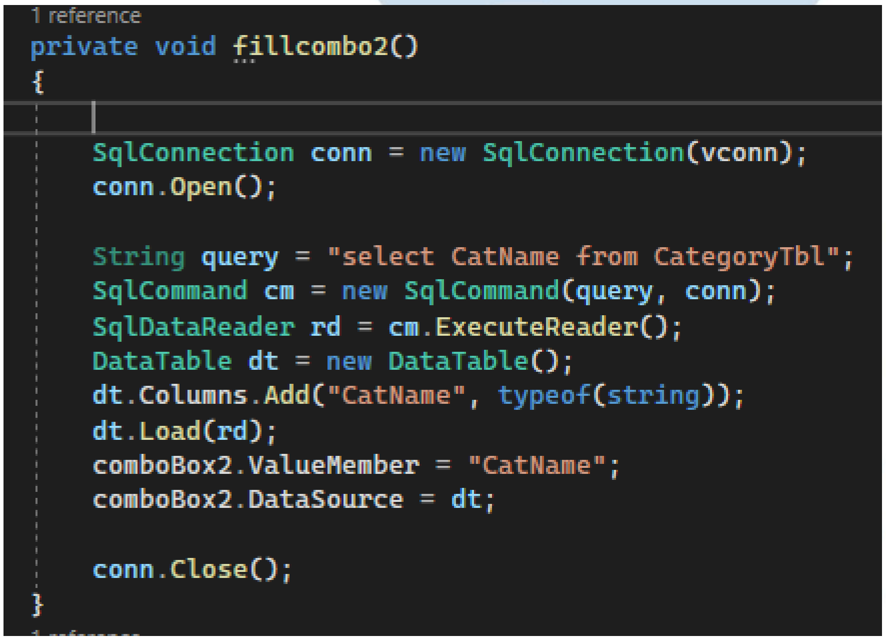
```
2 references
private void showdata()
{
    SqlConnection conn = new SqlConnection(vconn);
    String query = "select Proid,ProName, ProdPrice from ProductTbl";
    SqlCommand show = new SqlCommand(query, conn);

    var dt = new DataTable("ProductTbl");
    var name = new SqlDataAdapter(show);

    name.Fill(dt);
    PRODGV1.DataSource = dt;
}
2 references
```

The showdata function above has the same function as the function in ProductForm. Because the code above aims to enter what products are available in our supermarket, and sellers can choose what items are offered in our products, so it has the same function as "showdata" on ProductForm.



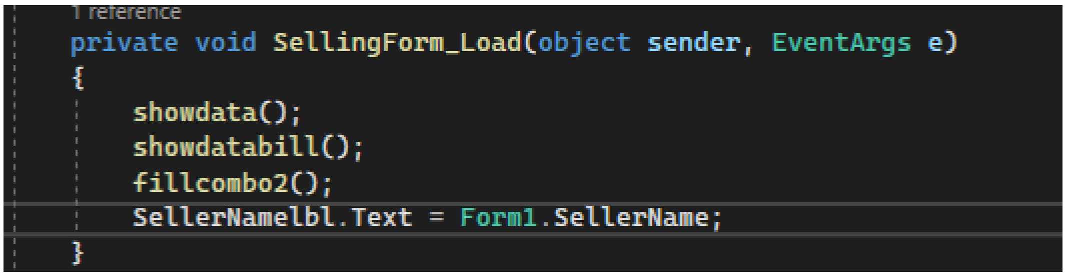
```
2 references
private void showdatabill()
{
    SqlConnection conn = new SqlConnection(vconn);
    String query = "select Billid, SellerName, BillDate, TotAmt from BillTbl";
    SqlCommand show = new SqlCommand(query, conn);

    var dt = new DataTable("BillTbl");
    var name = new SqlDataAdapter(show);

    name.Fill(dt);
    BillsDGV.DataSource = dt;
}
```

The showdata function above has the same function as the functions in ProductForm, SellerForm, and CategoryForm. But what distinguishes it from other forms is that the data displayed in the "DataGridView" is data in the Billid table (SellerName, BillDate, TotAmt). Then the data is shown in the "DataGridView" named "BillsDGV".



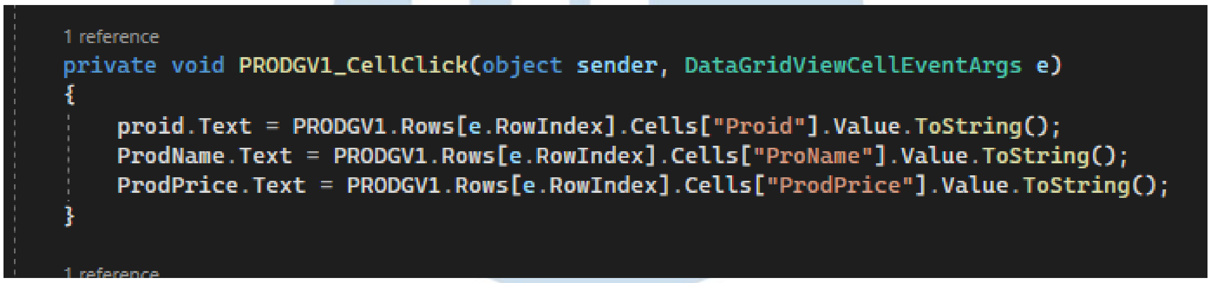
```
1 reference
private void fillcombo2()
{

    SqlConnection conn = new SqlConnection(vconn);
    conn.Open();

    String query = "select CatName from CategoryTbl";
    SqlCommand cm = new SqlCommand(query, conn);
    SqlDataReader rd = cm.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Columns.Add("CatName", typeof(string));
    dt.Load(rd);
    comboBox2.ValueMember = "CatName";
    comboBox2.DataSource = dt;

    conn.Close();
}
```

The code above has a function to show all the values in the CategoryTbl table in the CatName column. All values will be shown in ComboBox2. So that once we see all the values in ComboBox2, all the values come from the CategoryTbl table.



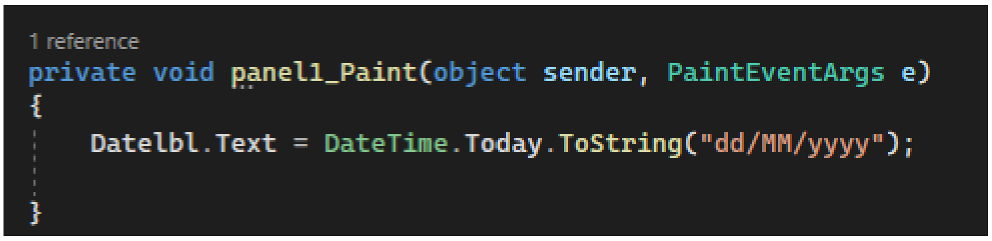
```
1 reference
private void SellingForm_Load(object sender, EventArgs e)
{
    showdata();
    showdatabill();
    fillcombo2();
    SellerNamelbl.Text = Form1.SellerName;
}
```

When all classes have been successfully created, we only need to include the class in "SellingForm_Load". The purpose of the class is entered into the form so that all classes can run according to their function and according to their purpose. Then there is the word "SellerNamelbl.Text". The value of this value is taken from SellerName which is on form1/login page. The goal is for customers to find out who is serving them in shopping at FreshSupermarket.



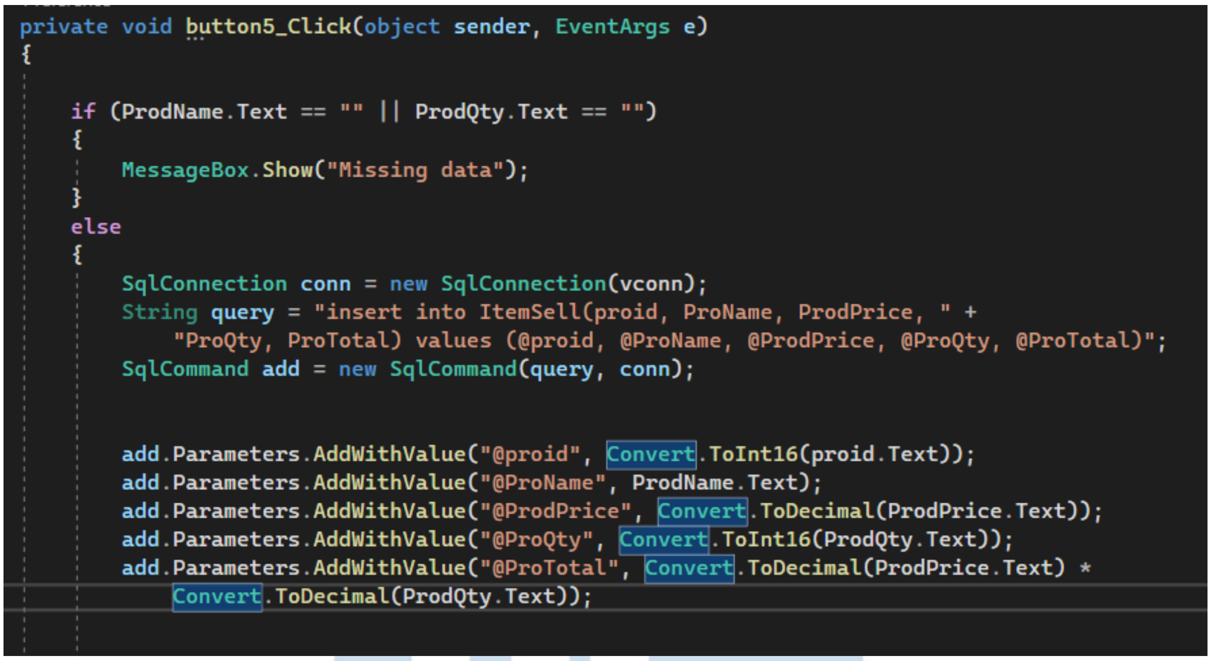
```
1 reference
private void PRODGV1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    proid.Text = PRODGV1.Rows[e.RowIndex].Cells["Proid"].Value.ToString();
    ProdName.Text = PRODGV1.Rows[e.RowIndex].Cells["ProName"].Value.ToString();
    ProdPrice.Text = PRODGV1.Rows[e.RowIndex].Cells["ProdPrice"].Value.ToString();
}

1 reference
```

The code above has a function to enter each value in each row that is clicked. The purpose of that is so that the data in the DataGridView according to the row that the seller clicks can come out in the TextBox that is available.The code above has a function to enter each value in each line that is clicked. The purpose of this is so that the data in the DataGridView matches the rows that the seller clicks can come out in the available TextBoxes.

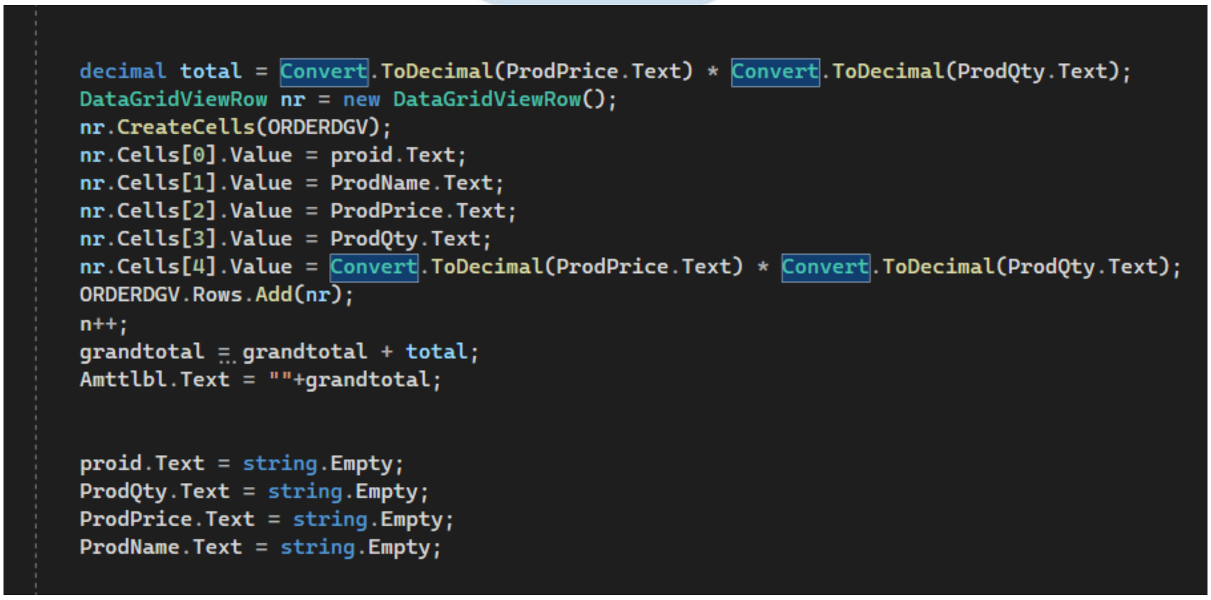


```
1 reference
private void panel1_Paint(object sender, PaintEventArgs e)
{
    Datelbl.Text = DateTime.Today.ToString("dd/MM/yyyy");

}
```

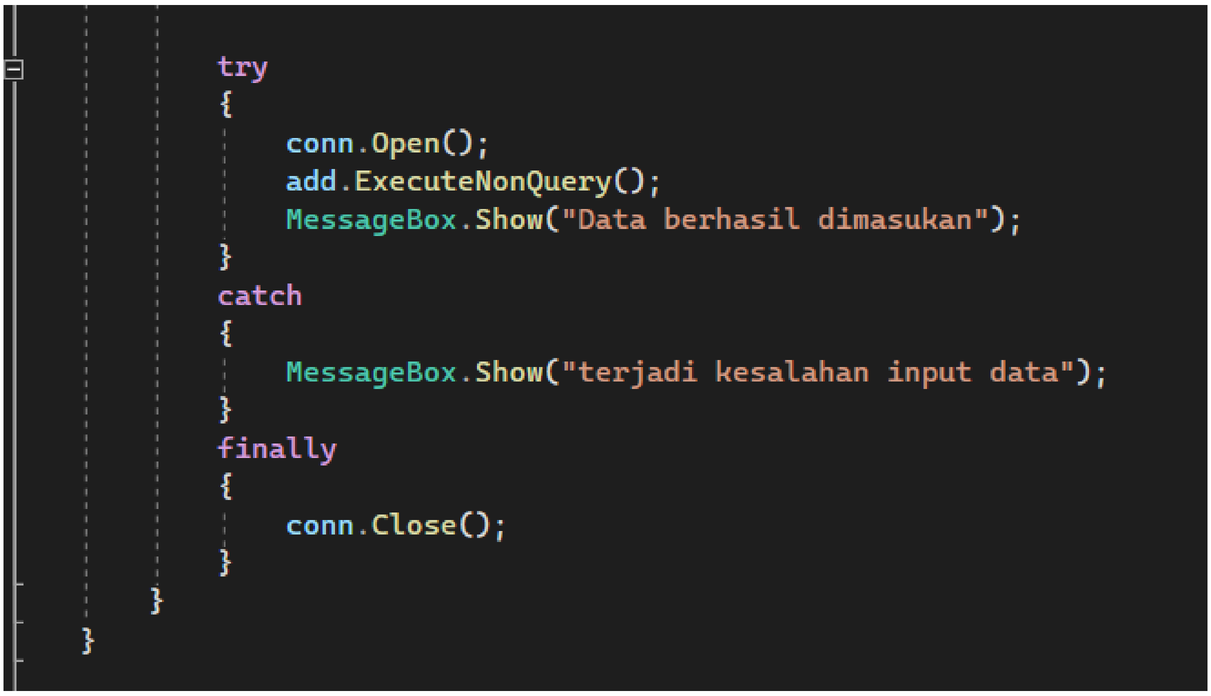
Then the panel above has a function to enter the day when the transaction occurred. We can see the date day month and also the year when the transaction occurred.


```
private void button5_Click(object sender, EventArgs e)
{

    if (ProdName.Text == "" || ProdQty.Text == "")
    {
        MessageBox.Show("Missing data");
    }
    else
    {
        SqlConnection conn = new SqlConnection(vconn);
        String query = "insert into ItemSell(proid, ProName, ProdPrice, " +
            "ProQty, ProTotal) values (@proid, @ProName, @ProdPrice, @ProQty, @ProTotal)";
        SqlCommand add = new SqlCommand(query, conn);


        add.Parameters.AddWithValue("@proid", Convert.ToInt16(proid.Text));
        add.Parameters.AddWithValue("@ProName", ProdName.Text);
        add.Parameters.AddWithValue("@ProdPrice", Convert.ToDecimal(ProdPrice.Text));
        add.Parameters.AddWithValue("@ProQty", Convert.ToInt16(ProdQty.Text));
        add.Parameters.AddWithValue("@ProTotal", Convert.ToDecimal(ProdPrice.Text) *
            Convert.ToDecimal(ProdQty.Text));
```

The code above is the code in the "add" button which has the same function as the add button in CategoryForm, ProductForm, SellerForm. What differs from CategoryForm, ProductForm, SellerForm is that the values entered into the table are the ItemSell table with columns SellerName, SellerAge, SellerPhone, SellerPass.

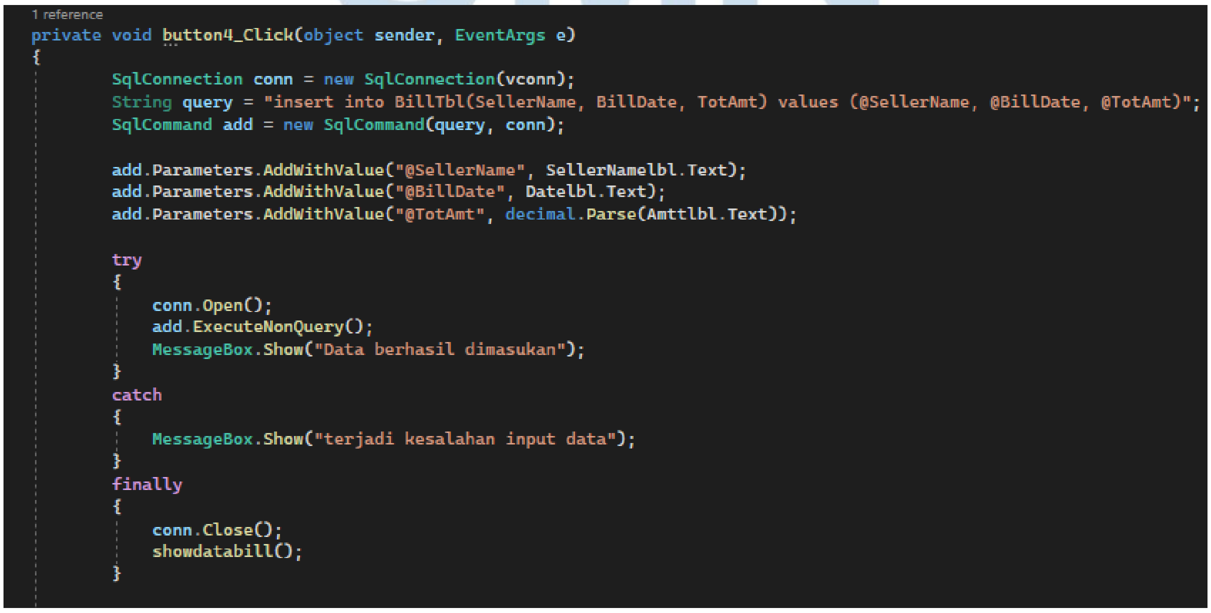

```
        decimal total = Convert.ToDecimal(ProdPrice.Text) * Convert.ToDecimal(ProdQty.Text);
        DataGridViewRow nr = new DataGridViewRow();
        nr.CreateCells(ORDERDGV);
        nr.Cells[0].Value = proid.Text;
        nr.Cells[1].Value = ProdName.Text;
        nr.Cells[2].Value = ProdPrice.Text;
        nr.Cells[3].Value = ProdQty.Text;
        nr.Cells[4].Value = Convert.ToDecimal(ProdPrice.Text) * Convert.ToDecimal(ProdQty.Text);
        ORDERDGV.Rows.Add(nr);
        n++;
        grandtotal = grandtotal + total;
        Amttlbl.Text = ""+grandtotal;


        proid.Text = string.Empty;
        ProdQty.Text = string.Empty;
        ProdPrice.Text = string.Empty;
        ProdName.Text = string.Empty;
```

However, we do not show the data entered into the database in the "DataGridView" of item sell. But we created our own table with the same columns as the ItemSell table. Then we enter each value with index 0 to 4. So that all values enter the table according to the TextBox value we take. Then in the total price section. We create a formula that is price * qty. and the data will come out in the table and also stored in the database. Then we create a Decimal type with the name "grandtotal" to hold all the totals purchased by the customer. Then create a label with the name "Amttlbl.text" so that the total of all items is calculated into the label. Making it easier for the customer to see the total he bought. Because the above function is a function on the add button, when you click the button all the values in the TextBox will disappear as if they were entered into the "DataGridView" table.



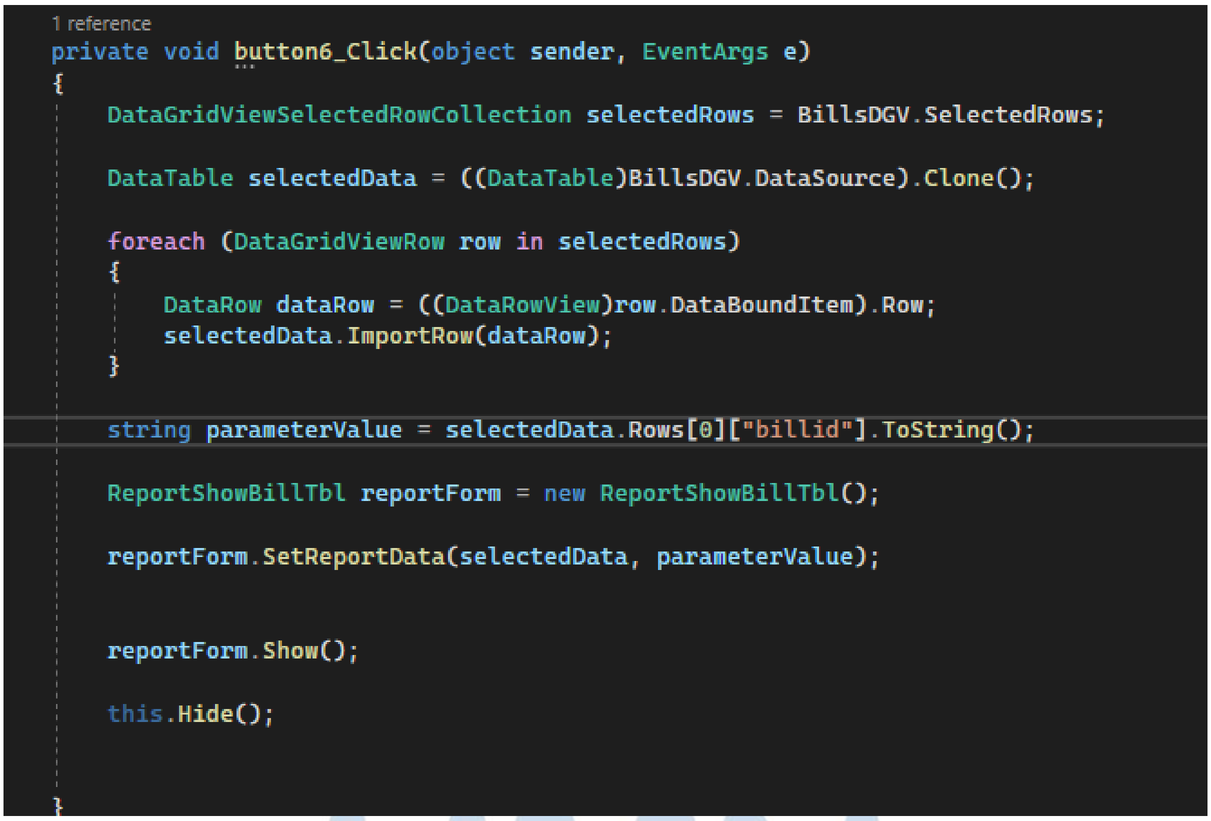
```
                try
                {
                    conn.Open();
                    add.ExecuteNonQuery();
                    MessageBox.Show("Data berhasil dimasukan");
                }
                catch
                {
                    MessageBox.Show("terjadi kesalahan input data");
                }
                finally
                {
                    conn.Close();
                }
            }
        }
```

Then when a data is successfully entered it will issue a messagebox "Data Successfully entered" and when the data fails to be entered or finds an error it will issue a messageBox "Data input error occurred".



```
1 reference
private void button4_Click(object sender, EventArgs e)
{
        SqlConnection conn = new SqlConnection(vconn);
        String query = "insert into BillTbl(SellerName, BillDate, TotAmt) values (@SellerName, @BillDate, @TotAmt)";
        SqlCommand add = new SqlCommand(query, conn);

        add.Parameters.AddWithValue("@SellerName", SellerNamelbl.Text);
        add.Parameters.AddWithValue("@BillDate", Datelbl.Text);
        add.Parameters.AddWithValue("@TotAmt", decimal.Parse(Amttlbl.Text));

        try
        {
            conn.Open();
            add.ExecuteNonQuery();
            MessageBox.Show("Data berhasil dimasukan");
        }
        catch
        {
            MessageBox.Show("terjadi kesalahan input data");
        }
        finally
        {
            conn.Close();
            showdatabill();
        }
```

The code above is the code in the "add" button which has the same function as the add button in CategoryForm, ProductForm, SellerForm. What distinguishes it from CategoryForm, ProductForm, SellerForm is that the values entered into the table are the BillTbl table with columns SellerName, BillDate, TotAmt.



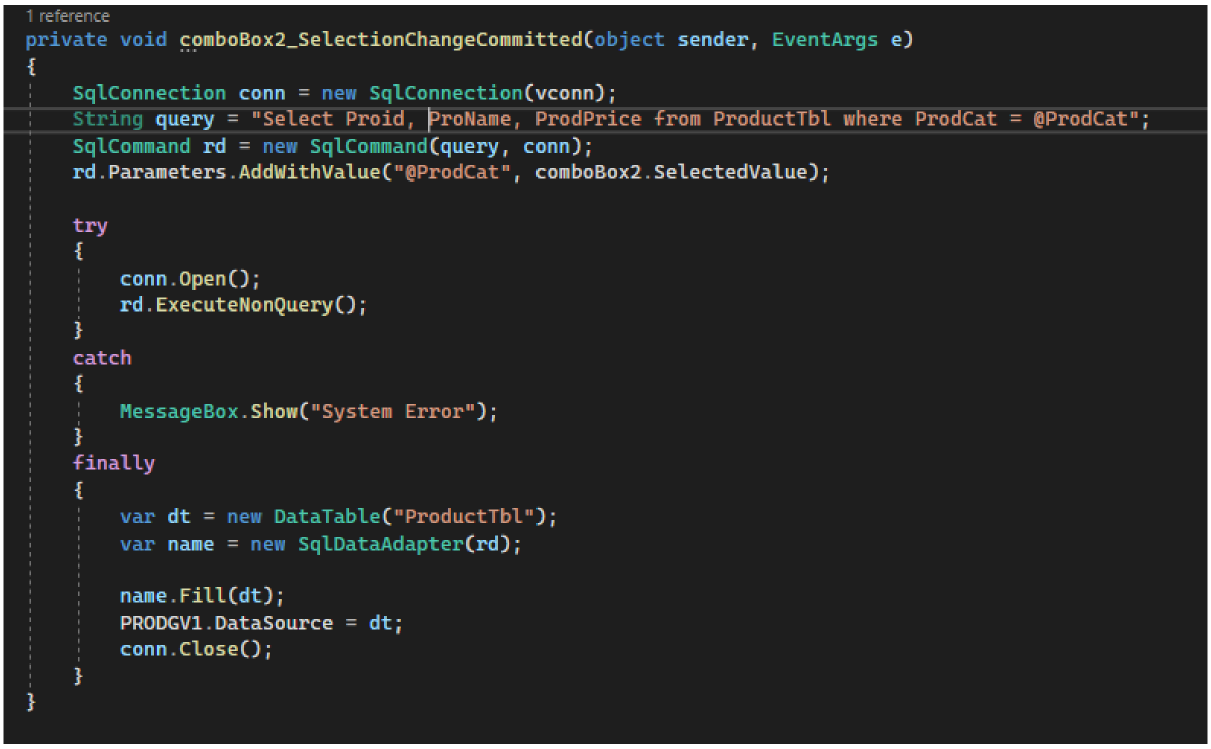
```
1 reference
private void button6_Click(object sender, EventArgs e)
{
    DataGridViewSelectedRowCollection selectedRows = BillsDGV.SelectedRows;

    DataTable selectedData = ((DataTable)BillsDGV.DataSource).Clone();

    foreach (DataGridViewRow row in selectedRows)
    {
        DataRow dataRow = ((DataRowView)row.DataBoundItem).Row;
        selectedData.ImportRow(dataRow);
    }

    string parameterValue = selectedData.Rows[0]["billid"].ToString();

    ReportShowBillTbl reportForm = new ReportShowBillTbl();

    reportForm.SetReportData(selectedData, parameterValue);


    reportForm.Show();

    this.Hide();


}
```

The code above has a function to print reports using crystal report filtering with the selected index on the DataGridView according to the desired row.

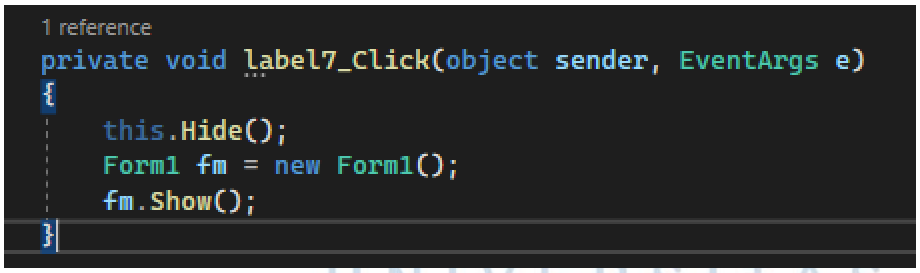
1. “DataGridViewSelectedRowCollection selectedRows = BillsDGV.SelectedRows;” function to create a selectedRows collection that contains the selected rows in BillsDGV.
2. “DataTable selectedData = ((DataTable)BillsDGV.DataSource).Clone();” function to create a table selectedData with the same structure as the source table (BillsDGV). This table will be used to store the selected row data.
3. Via a foreach loop, each row selected in BillsDGV is imported into selectedData using the ImportRow method. The goal is to copy the selected data rows into the selectedData table.
4. “string parameterValue = selectedData.Rows[0]["billid"].ToString();” function to fetch value from column "billid"(bill ID) of first row in selectedData. This value will be used as a parameter for the report.
5. “reportForm.SetReportData(selectedData, parameterValue);” function to set the report data by calling the SetReportData method on the reportForm and providing selectedData and

parameterValue as arguments. This method aims to populate data in the report using the selectedData table and setting the report parameter with the parameterValue value.

6. "reportForm. Show();" function to display the report form.
7. "this. Hide();" function to hide the SellingForm form

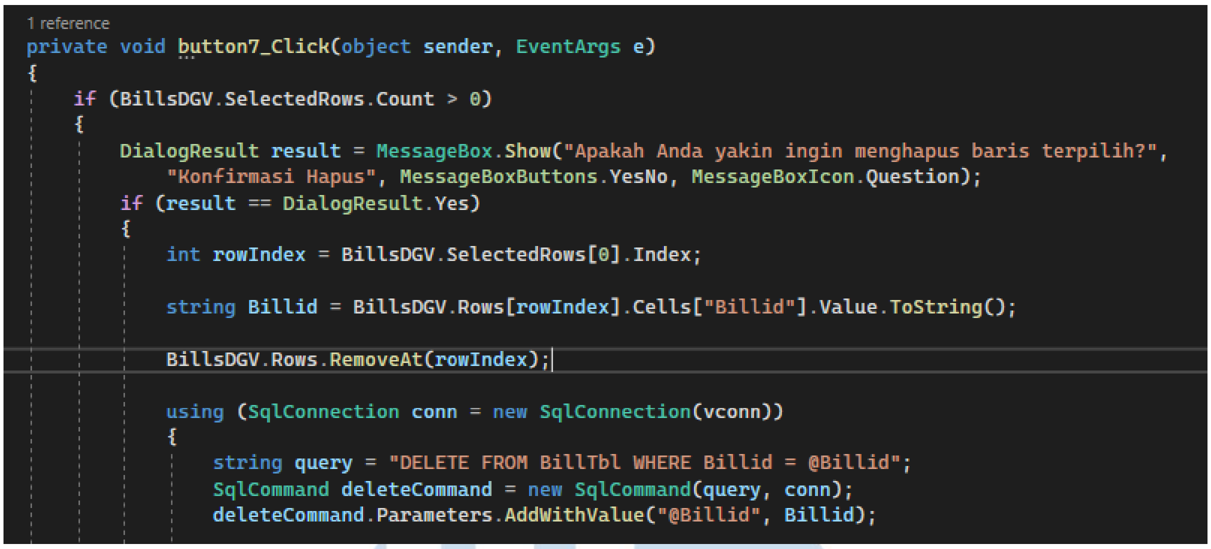
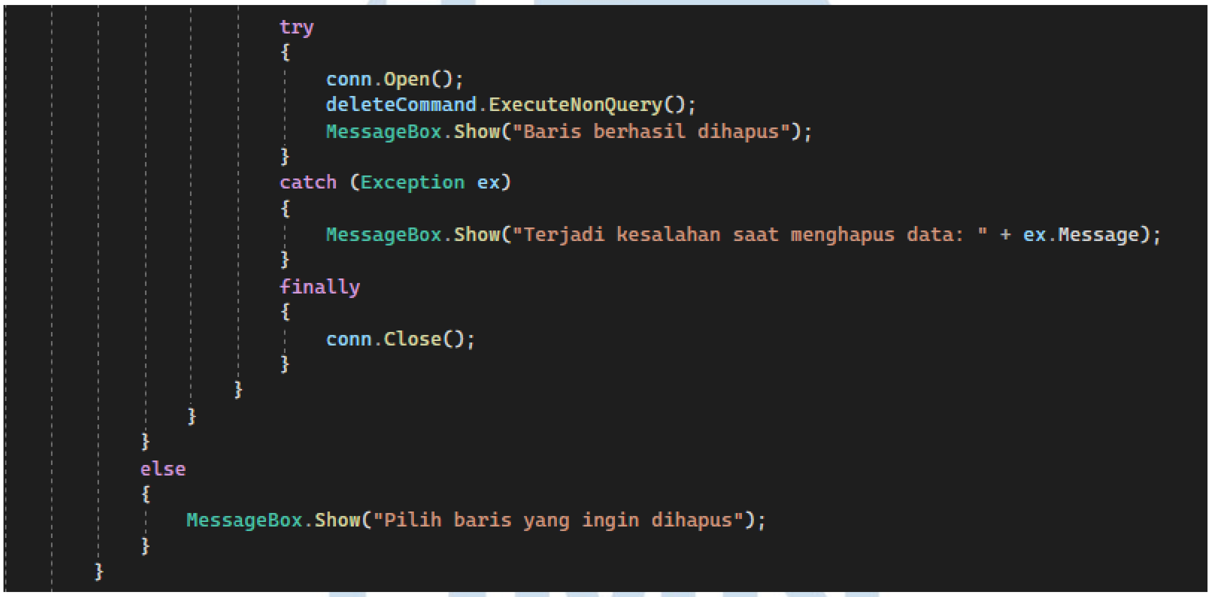


```
1 reference
private void comboBox2_SelectionChangeCommitted(object sender, EventArgs e)
{
    SqlConnection conn = new SqlConnection(vconn);
    String query = "Select Proid, ProName, ProdPrice from ProductTbl where ProdCat = @ProdCat";
    SqlCommand rd = new SqlCommand(query, conn);
    rd.Parameters.AddWithValue("@ProdCat", comboBox2.SelectedValue);

    try
    {
        conn.Open();
        rd.ExecuteNonQuery();
    }
    catch
    {
        MessageBox.Show("System Error");
    }
    finally
    {
        var dt = new DataTable("ProductTbl");
        var name = new SqlDataAdapter(rd);

        name.Fill(dt);
        PRODGV1.DataSource = dt;
        conn.Close();
    }
}
```



```
1 reference
private void label7_Click(object sender, EventArgs e)
{
    this.Hide();
    Form1 fm = new Form1();
    fm.Show();
}
```

Above is the code to log out on the Seller menu. So that when the seller finishes working the seller can click on the item menu.

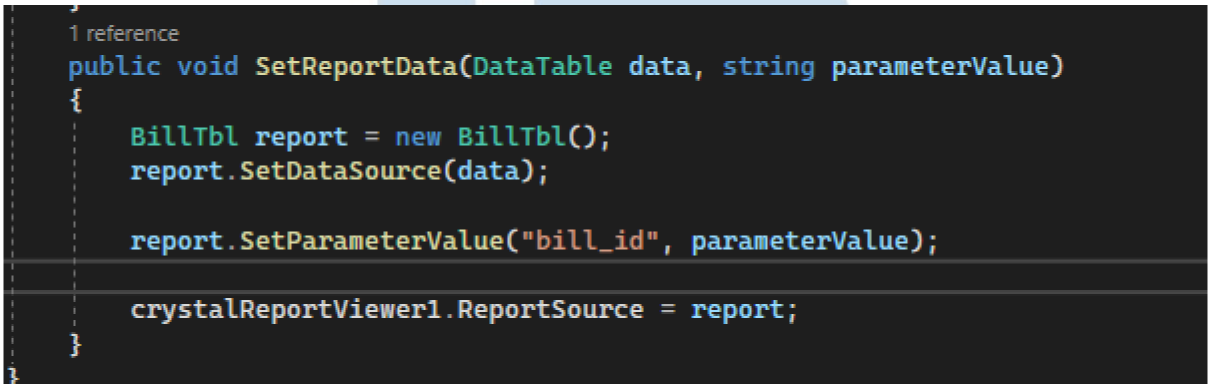

```
1 reference
private void button7_Click(object sender, EventArgs e)
{
    if (BillsDGV.SelectedRows.Count > 0)
    {
        DialogResult result = MessageBox.Show("Apakah Anda yakin ingin menghapus baris terpilih?",
            "Konfirmasi Hapus", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (result == DialogResult.Yes)
        {
            int rowIndex = BillsDGV.SelectedRows[0].Index;

            string Billid = BillsDGV.Rows[rowIndex].Cells["Billid"].Value.ToString();

            BillsDGV.Rows.RemoveAt(rowIndex);

            using (SqlConnection conn = new SqlConnection(vconn))
            {
                string query = "DELETE FROM BillTbl WHERE Billid = @Billid";
                SqlCommand deleteCommand = new SqlCommand(query, conn);
                deleteCommand.Parameters.AddWithValue("@Billid", Billid);
```

```
                try
                {
                    conn.Open();
                    deleteCommand.ExecuteNonQuery();
                    MessageBox.Show("Baris berhasil dihapus");
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Terjadi kesalahan saat menghapus data: " + ex.Message);
                }
                finally
                {
                    conn.Close();
                }
            }
        }
    }
    else
    {
        MessageBox.Show("Pilih baris yang ingin dihapus");
    }
}
```

The code above is a function that is useful for deleting the row index in the BillTbl table according to the options chosen by the Seller or by the Admin. so that the code above if detailed will be as follows:

1. "if (BillsDGV.SelectedRows.Count > 0)" serves to check whether there is a row selected in BillsDGV. If there are no rows selected, a message will be displayed that the rows to be deleted must be selected first.
2. "DialogResult result = MessageBox.Show(...)" serves to display a confirmation dialog to the user by using MessageBox to confirm whether the user is sure that he/she wants to delete the selected rows. The user can choose "Yes" or "No". If the user selects "Yes", the selected row will be deleted from BillsDGV.

3. "BillsDGV.Rows.RemoveAt(rowIndex);" serves to remove the selected rows from BillsDGV by using RemoveAt and providing rowIndex as the index of the rows to be removed.
4. The messageBox "row removed successfully" is displayed if the removal is successful. If an error occurs, it will issue a messageBox "An error occurred while deleting data".
5. the finally section serves to ensure that the connection to the database is closed after the operation is completed.
6. In the last section, if the seller or admin does not select index on the DataGridView but clicks on the table, it will issue a messageBox "Select the row you want to delete".

REPORT SHOW BILL CUSTOMER

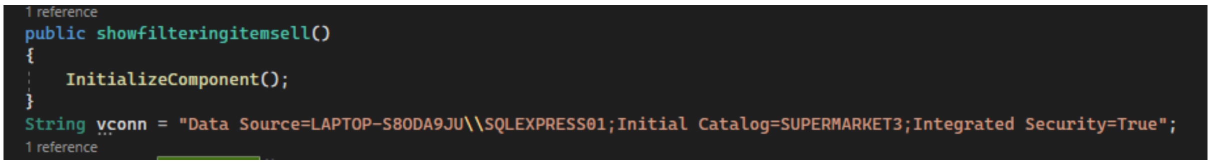


```
1 reference
public void SetReportData(DataTable data, string parameterValue)
{
    BillTbl report = new BillTbl();
    report.SetDataSource(data);

    report.SetParameterValue("bill_id", parameterValue);

    crystalReportViewer1.ReportSource = report;
}
```

The code above is part of crystal report filtering, which is filtering by creating receipts for customers. First we create a parameter with the name "bill_id. The following is a detailed explanation:

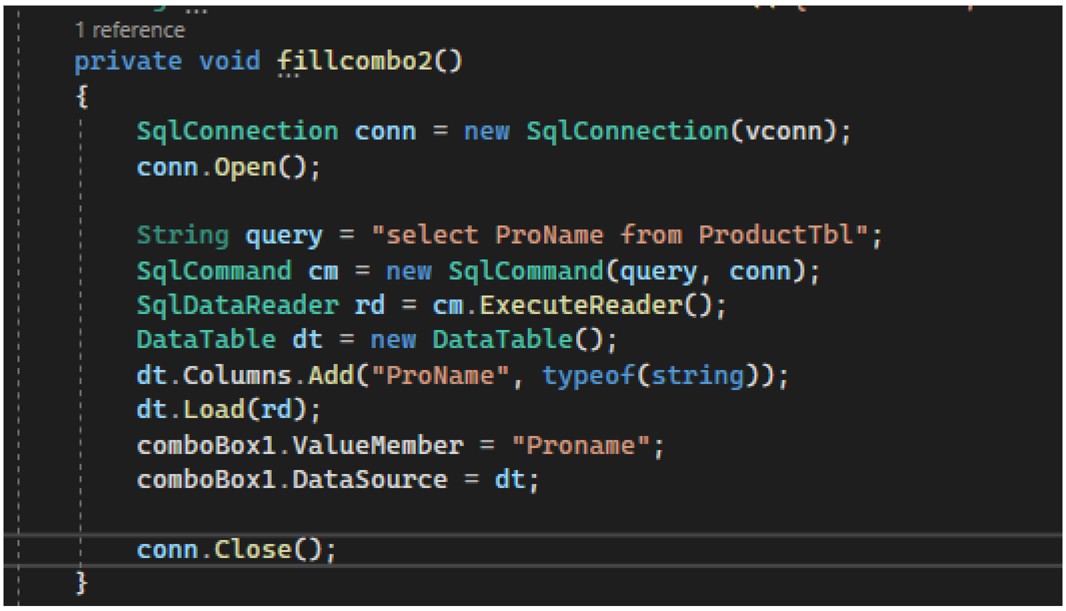
1. Create a BillTbl object instance with the name report.
2. Set the data to be displayed in the report using the SetDataSource method on the report object. The data to be displayed comes from the DataTable given as the data argument.
3. Set the report parameter values using the SetParameterValue method on the report object. The parameter values are set using the parameterValue argument given.
4. Set the report object as the report source in crystalReportViewer1 using the ReportSource property.

FILTERING ITEM SELL



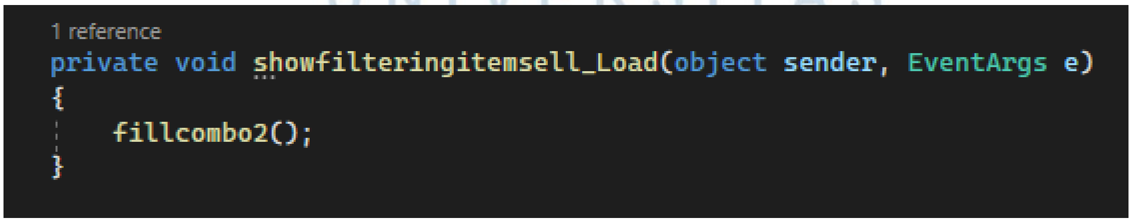
```
1 reference
public showfilteringitemsell()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
1 reference
```

The code above is the code to call the database that we use. In the code above the database we use is named "SUPERMARKET3".



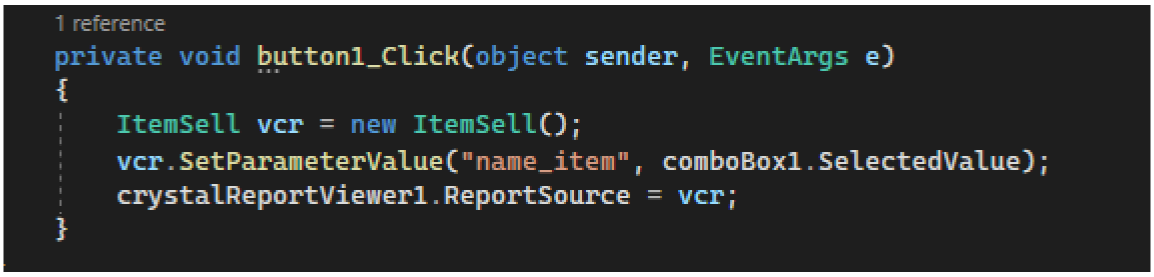
```
1 reference
private void fillcombo2()
{
    SqlConnection conn = new SqlConnection(vconn);
    conn.Open();

    String query = "select ProName from ProductTbl";
    SqlCommand cm = new SqlCommand(query, conn);
    SqlDataReader rd = cm.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Columns.Add("ProName", typeof(string));
    dt.Load(rd);
    comboBox1.ValueMember = "Proname";
    comboBox1.DataSource = dt;

    conn.Close();
}
```

FillCombo has the same function as the previous form where the class will provide the value of ProductName from the ProductTbl table in ComboBox1. So that all products in ProName will enter ComboBox1.



```
1 reference
private void showfilteringitemsell_Load(object sender, EventArgs e)
{
    fillcombo2();
}
```

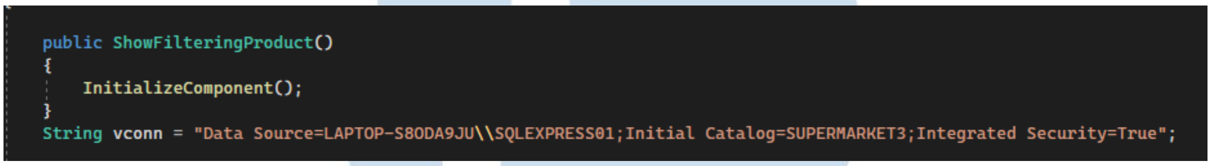
So that all values can run we need to implement it into the form, so that the code can run



```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    ItemSell vcr = new ItemSell();
    vcr.SetParameterValue("name_item", comboBox1.SelectedValue);
    crystalReportViewer1.ReportSource = vcr;
}
```

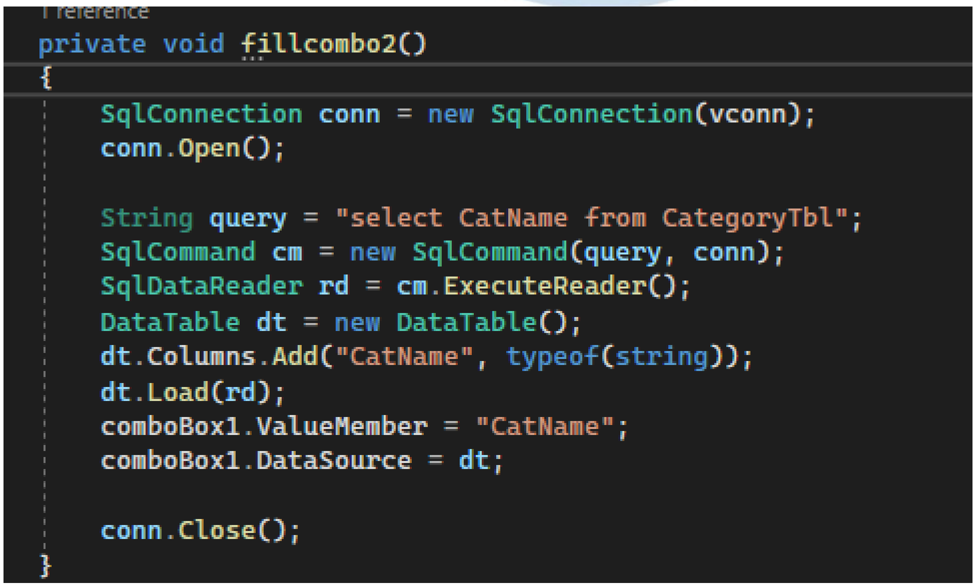
The code above is a Crystal report filtering to filter by item name on each product that has been sold. We create a parameter named "name_item" in order to filter based on the item name of each product. Then every time we choose a value in the combobox in the form, the value that comes out will match the combobox that has been selected by the seller or admin.

FILTERING PRODUCT



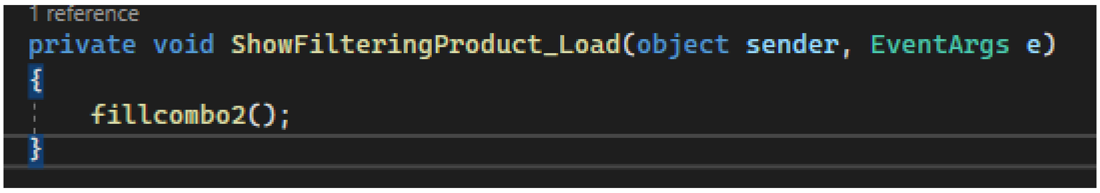
```
public ShowFilteringProduct()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
```

The code above is the code to call the database that we use. In the code above the database we use is named "SUPERMARKET3".



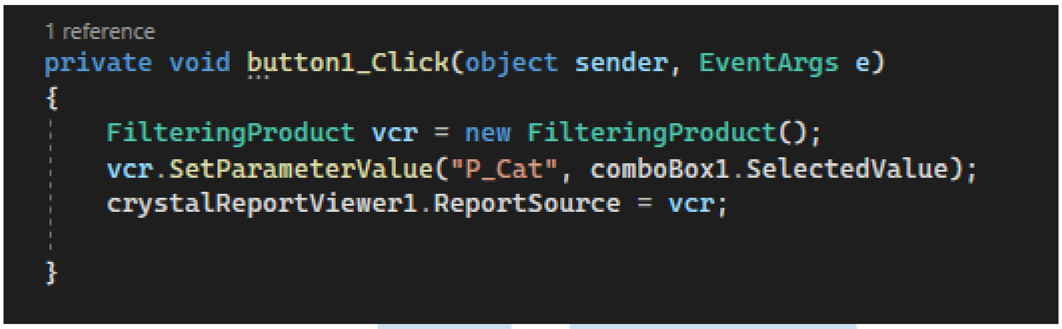
```
1 reference
private void fillcombo2()
{
    SqlConnection conn = new SqlConnection(vconn);
    conn.Open();

    String query = "select CatName from CategoryTbl";
    SqlCommand cm = new SqlCommand(query, conn);
    SqlDataReader rd = cm.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Columns.Add("CatName", typeof(string));
    dt.Load(rd);
    comboBox1.ValueMember = "CatName";
    comboBox1.DataSource = dt;

    conn.Close();
}
```

FillCombo has the same function as the previous form where the class will provide the value of CatName from the CategoryTbl table in ComboBox1. So that all categories in CatName will be entered into ComboBox1.



```
1 reference
private void ShowFilteringProduct_Load(object sender, EventArgs e)
{
    fillcombo2();
}
```

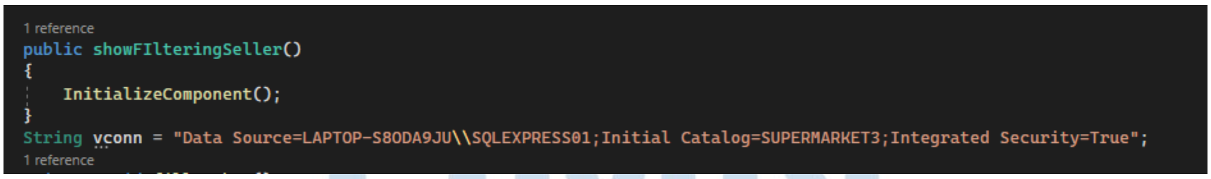
In order for all values to run we need to implement them into the form, so that the code can run



```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    FilteringProduct vcr = new FilteringProduct();
    vcr.SetParameterValue("P_Cat", comboBox1.SelectedValue);
    crystalReportViewer1.ReportSource = vcr;

}
```

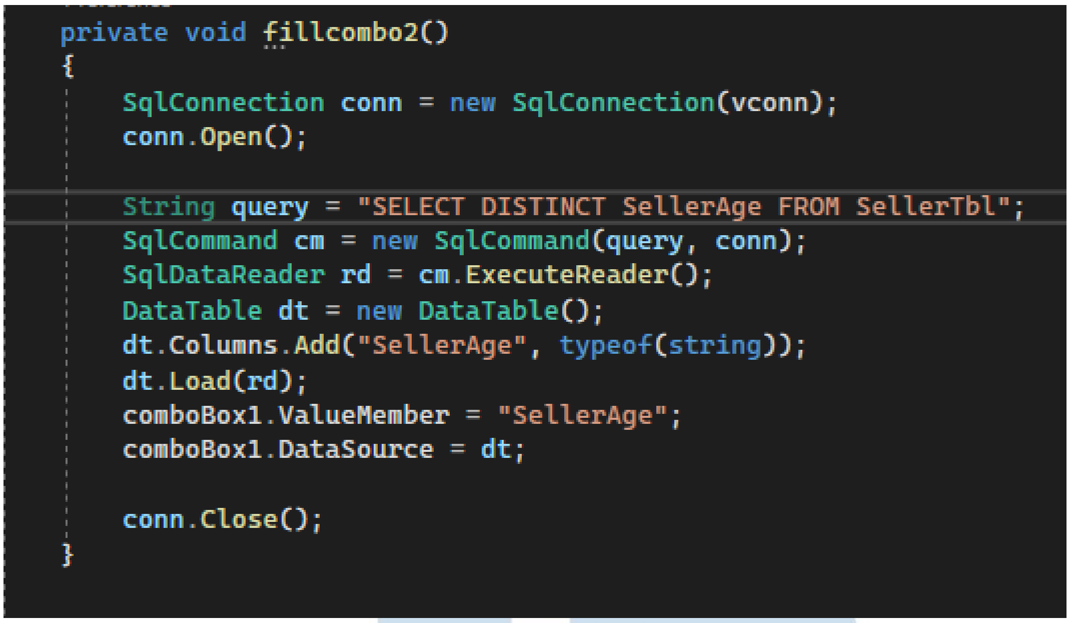
The code above is a Crystal report filtering to filter by category name on each product owned by our supermarket. We create a parameter named "P_cat" in order to filter based on the category of the product. Then, every time we choose a value in the combobox in the form, the value that comes out will match the combobox that has been selected by the admin.

FILTERING SELLER



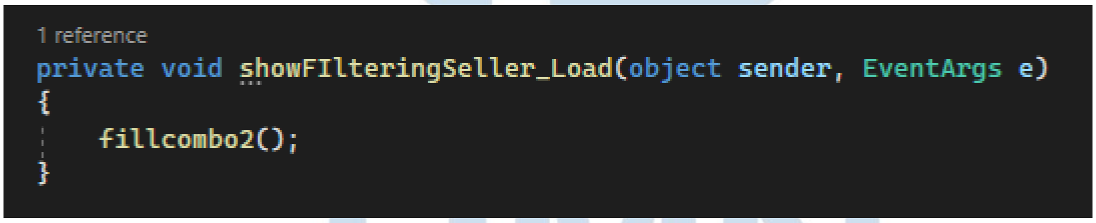
```
1 reference
public showFIlteringSeller()
{
    InitializeComponent();
}
String vconn = "Data Source=LAPTOP-S8ODA9JU\\SQLEXPRESS01;Initial Catalog=SUPERMARKET3;Integrated Security=True";
1 reference
```

The code above is the code to call the database that we use. In the code above the database we use is named "SUPERMARKET3".



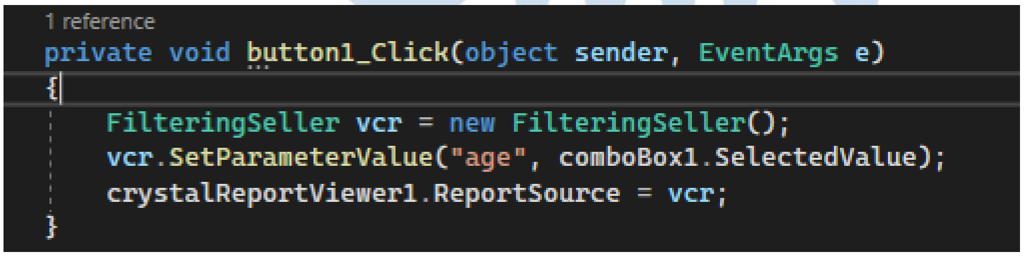
```
private void fillcombo2()
{
    SqlConnection conn = new SqlConnection(vconn);
    conn.Open();

    String query = "SELECT DISTINCT SellerAge FROM SellerTbl";
    SqlCommand cm = new SqlCommand(query, conn);
    SqlDataReader rd = cm.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Columns.Add("SellerAge", typeof(string));
    dt.Load(rd);
    comboBox1.ValueMember = "SellerAge";
    comboBox1.DataSource = dt;

    conn.Close();
}
```

FillCombo has the same function as the previous form where the class will provide the value of SellerAge from the SellerTbl table in ComboBox1. So that all categories in SellerAge will enter ComboBox1. DISTINCT above has a function when a value has a twin value, it will be displayed once to represent the value.



```
1 reference
private void showFIlteringSeller_Load(object sender, EventArgs e)
{
    fillcombo2();
}
```

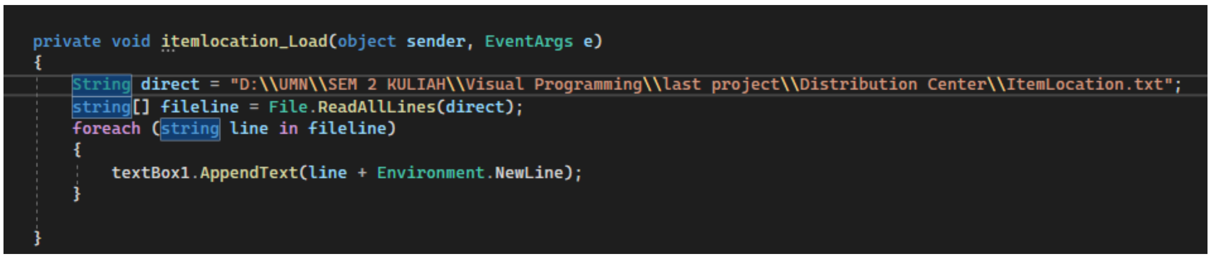
So that all values can run we need to implement it into the form, so that the code can run



```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    FilteringSeller vcr = new FilteringSeller();
    vcr.SetParameterValue("age", comboBox1.SelectedValue);
    crystalReportViewer1.ReportSource = vcr;
}
```

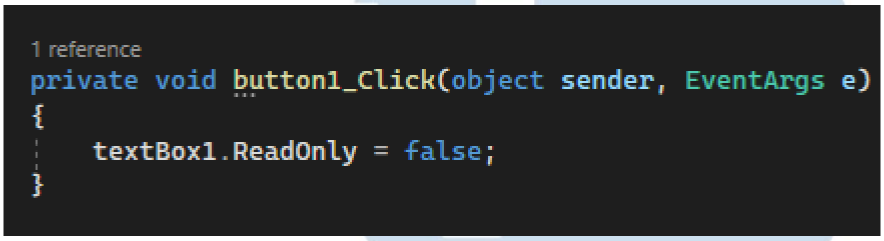
seller tersebut The code above is a Crystal report filtering to filter by category name on each product owned by our supermarket. We create a parameter named "age" in order to filter based on the category of the product. Then, every time we choose a value in the combobox in the form, the value that comes out will match the combobox that has been selected by the admin.

ITEM LOCATION



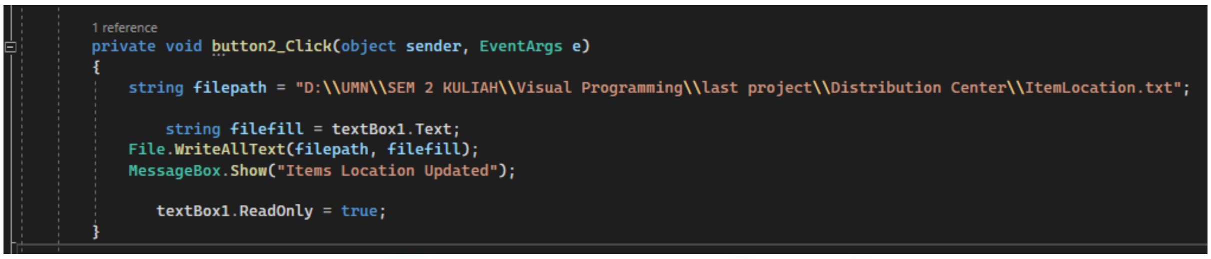
```
private void itemlocation_Load(object sender, EventArgs e)
{
    String direct = "D:\\UMN\\SEM 2 KULIAH\\Visual Programming\\last project\\Distribution Center\\ItemLocation.txt";
    string[] fileline = File.ReadAllLines(direct);
    foreach (string line in fileline)
    {
        textBox1.AppendText(line + Environment.NewLine);
    }

}
```

In the code above, when itemlocation is loaded, the lines of text from the file "ItemLocation.txt" are read using File.ReadAllLines() and stored in the fileline array. Next, through the foreach loop, each line of text is appended to textBox1 using the AppendText() method. Thus, the contents of the file "ItemLocation.txt" will be displayed in textBox1 with each line separated by Environment.NewLine.



```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    textBox1.ReadOnly = false;
}
```

When we click the Edit button, we can make various updates to the item location. Based on the code above, our readonly is false.



```
1 reference
private void button2_Click(object sender, EventArgs e)
{
    string filepath = "D:\\UMN\\SEM 2 KULIAH\\Visual Programming\\last project\\Distribution Center\\ItemLocation.txt";

        string filefill = textBox1.Text;
    File.WriteAllText(filepath, filefill);
    MessageBox.Show("Items Location Updated");

        textBox1.ReadOnly = true;
}
```

In the code above, when the "button2" button is clicked, the contents of textBox1 will be retrieved and stored in the filefill variable. after that, using the File.WriteAllText() method, the contents of the filefill will be written into the file specified by filepath. After that, a dialog message will appear notifying that "Items Location Updated". then textBox1 will be set as "ReadOnly" (not editable) by setting the ReadOnly property to true.

Roles for each team members:

1. Ray Anthony Pranoto - 00000066655 (Project Manager, Application Developer, Database Design, Proposal)
2. Christopher Abie Diaz Doviano - 00000067692 (Application Developer, Proposal, Database Design)
3. Nicholas Alven Gandra - 00000066511- (Proposal, Concept)
4. Adryel Ethantyo - 00000068165 - (Proposal, Video Edit)
5. Yoga Saputra - 00000065491 - (Proposal, Concept, Flowchart)

Link Gdrive:

https://drive.google.com/drive/folders/1x2Ae5KzlZIdJxAwgin5QxgEHW4GjzVrH?usp=sharing