**Module:**

**BIS2212 - Database Systems: Design and Online**

**Coursework Title:**

**Coursework 2 – Creativity and Standards Task**

**Authored by: Andrea Montesin Middlesex Number: M00433863 Centre: STC Training Malta Class: 2nd Year BSc IAD PT 2012 Date Handed In: 13th December, 2012 Lecturer: Mr John Debono Module Leader: Dr Nawaz Khan**

Table of Contents

[Introduction 4](#_Toc354595528)

[Working Pages! 5](#_Toc354595529)

[Connection 5](#_Toc354595530)

[Fig.1 – SQL Connection "Conn" 5](#_Toc354595531)

[New Cars. 5](#_Toc354595532)

[Fig.2 – New Cars Back End Code 6](#_Toc354595533)

[Fig.3 – New Cars SearchQuery 6](#_Toc354595534)

[Fig.4 – New Cars gvVehicle\_SelectedIndexChanged 7](#_Toc354595535)

[Used Cars. 7](#_Toc354595536)

[Fig.5 – Used Cars Back End Code 7](#_Toc354595537)

[CarsSingleView. 8](#_Toc354595538)

[Part 1 -CarsSingleView - Page\_Load: 9](#_Toc354595539)

[Fig.5 – CarsSingleView - Part1 - PageLoad 9](#_Toc354595540)

[Fig.6 – Requesting the Vehicle ID 10](#_Toc354595541)

[Fig.7 – Verifying if the user is logged in to avail btnAddToCart 10](#_Toc354595542)

[Fig.8 – Verifying UserID to enable the Delete Button 11](#_Toc354595543)

[Fig.9 – Verifying UserID to enable the Delete Button 12](#_Toc354595544)

[Part 2 -CarsSingleView - BindingVehicleData(): 12](#_Toc354595545)

[Fig.10 – BindingVehicleData() 12](#_Toc354595546)

[Part 3 -CarsSingleView - AltImgBinding(): 13](#_Toc354595547)

[Fig.11 – AltBindingVehicleData() 13](#_Toc354595548)

[Part 4 - CarsSingleView - btnImg\_Click: 14](#_Toc354595549)

[Fig.12 – btnImg\_Click 14](#_Toc354595550)

[Part 5 -CarsSingleView - Deleting a vehicle: 14](#_Toc354595551)

[Fig.13 – AltBindingVehicleData() 14](#_Toc354595552)

[Part 6 -CarsSingleView - VehicleDel() 15](#_Toc354595553)

[Fig.14 – AltBindingVehicleData() 16](#_Toc354595554)

[Part 7 -CarsSingleView - btnAddCart\_Click: 17](#_Toc354595555)

[Fig.15 – btnAddCart\_Click 17](#_Toc354595556)

[ShoppingCartVehicleVal: 17](#_Toc354595557)

[Fig.16 – ShoppingCartVehVal() 18](#_Toc354595558)

[Fig.17 – AddToCart() 19](#_Toc354595559)

[CustomerCars. 20](#_Toc354595560)

[Part1 - CustomerCars - Initialisation: 20](#_Toc354595561)

[Part2 - CustomerCars - Page\_Load: 20](#_Toc354595562)

[Fig.18 – VehicleFunction.UserVehicles Function 21](#_Toc354595563)

[Part3 - CustomerCars - gvVehicle\_RowDeleting: 21](#_Toc354595564)

[Fig.19 – VehicleFunction.VehicleDel Function 22](#_Toc354595565)

[Part3 - CustomerCars - gvVehicle\_SelectedIndexChanged: 22](#_Toc354595566)

[Fig.20 – CustomerCars Page 23](#_Toc354595567)

[VehicleSearch. 24](#_Toc354595568)

[Fig.21 – SearchBtn\_Click Event 24](#_Toc354595569)

[Part1 - CarsSingleView - Page\_Load: 24](#_Toc354595570)

[Fig.22 – VehicleFunction.SearchQuery() 25](#_Toc354595571)

[Fig.22 – CarSearch Page 25](#_Toc354595572)

[Register. 26](#_Toc354595573)

[Part1 - Register - Page\_Load: 26](#_Toc354595574)

[Fig.23 – UserValidation.UserVal() 27](#_Toc354595575)

[Fig.24 – UserNameValidation.UserVal() 28](#_Toc354595576)

[Fig.25 – UserFunction.RegUser() 29](#_Toc354595577)

[Fig.26 – Register Page 30](#_Toc354595578)

[Login, Admin Panel, User Panel and User Logged. 31](#_Toc354595579)

[Part1 - Login - Page\_Load: 31](#_Toc354595580)

[Part2 - Login - btnLogin\_Click: 31](#_Toc354595581)

[Fig.27 – btnLogin\_Click - Part 1 And Part 2 32](#_Toc354595582)

[Fig.28 – UserValFields() 33](#_Toc354595583)

[Part2 - Login - btnLogin\_Click - Continued: 33](#_Toc354595584)

[UserPanel: 33](#_Toc354595585)

[Fig.29 – UserPanel page 34](#_Toc354595586)

[Database 35](#_Toc354595587)

[User Table: 35](#_Toc354595588)

[Fig. 1.1 – User Table 36](#_Toc354595589)

[Fig. 1.2 – Populated OrderLine Table 36](#_Toc354595590)

[ShoppingCart Table: 36](#_Toc354595591)

[Fig. 1.2 – ShoppingCart Table 37](#_Toc354595592)

[Fig. 1.3 – Populated ShoppingCart Table 37](#_Toc354595593)

[Order Table: 37](#_Toc354595594)

[Fig. 1.4 – Order Table 38](#_Toc354595595)

[Fig. 1.5 – Populated Order Table 38](#_Toc354595596)

[OrderLine Table: 38](#_Toc354595597)

[Fig. 1.6 – OrderLine Table 38](#_Toc354595598)

[Fig. 1.7 – Populated OrderLine Table 39](#_Toc354595599)

[Category Table: 39](#_Toc354595600)

[Fig. 1.7 – Category Table Structure 39](#_Toc354595601)

[Fig. 1.8 – Populated Category Table 40](#_Toc354595602)

[Vehicle Table: 40](#_Toc354595603)

[Fig. 1.9 – Vehicle Table Structure 41](#_Toc354595604)

[Fig. 2.0 – Populated Vehicle Table 42](#_Toc354595605)

[References 42](#_Toc354595606)

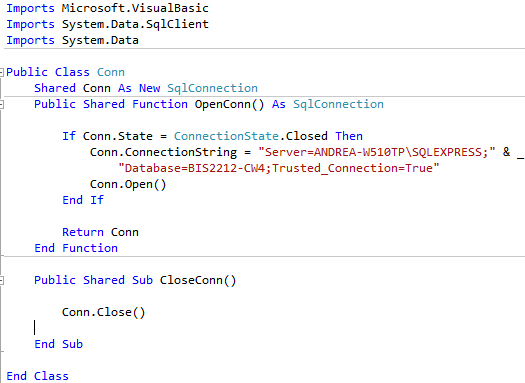
[Appendix 42](#_Toc354595607)

# Introduction

# Working Pages!

## Connection

Below we will be illustrating in Fig.1 the actual connection class, which will be used to connect our website with "**Microsoft SQL Server 2008**". The "Conn" will include a connection string with the actual connection to the local SQL Server Connection.

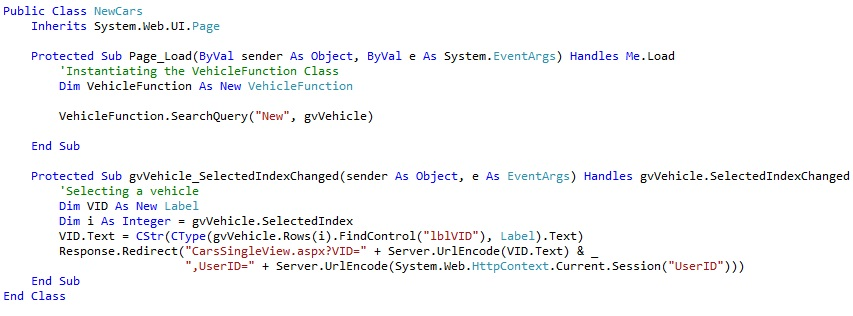


### **Fig.1 – SQL Connection "Conn"**

The connection string may be modified if our website is run on other PCs.

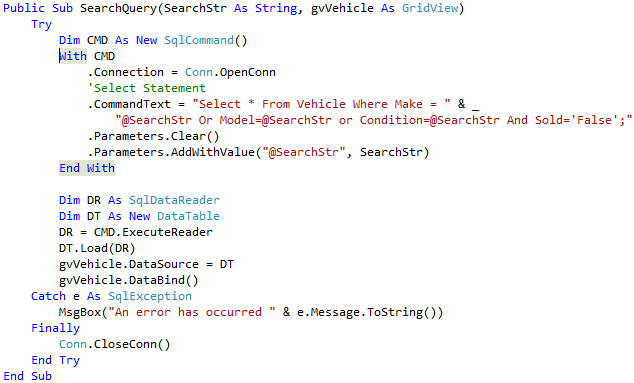
## New Cars.

The "New Cars" page will be used to display the new cars available on our (e-commerce) website. Below in Fig.2 one can find illustrated the back end design including the populated gridview for the "New Cars" page.



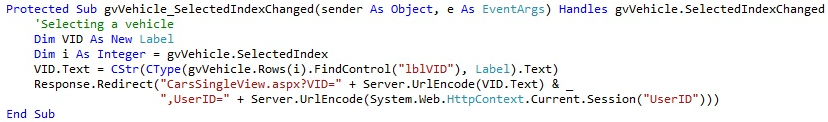
### **Fig.2 – New Cars Back End Code**

Right in the **Page\_load** for the "NewCars" page, one can find an instance of the "**VehicleFunction**" class. This instance will be used to access the "**SearchQuery**" function (shown in Fig.3) found in the same "**VehicleFunction**" class.



### **Fig.3 – New Cars SearchQuery**

The "SearchQuery" function will pass "New" through the parameter "@SearchStr" to the **Select** statement. This function will finally select all the vehicles (from the Vehicle table) where their condition is new, and bind the results to the gridview (gvVehicle).



### **Fig.4 – New Cars gvVehicle\_SelectedIndexChanged**

The second part of the NewCars page being "**gvVehicle\_SelectedIndexChanged**" (shown in Fig.4 (above)) will be used to select a vehicle and view its full description. Therefore when the user clicks on the (auto-generated) Select link button from the gridview, our website will find control on the gridview (bound) column name "**lblVID**" in order to retrieve the vehicle ID and pass it as an "encoded string" to the "**CarsSingleView**" page. Apart from the Vehicle ID the User ID is also passed to the "**CarsSingleView**" page, in order to maintain an extra user verification record, apart from the session.

## Used Cars.

The "UsedCars" page illustrated in Fig.5 uses the same logic as the "NewCars" page. The only difference between both pages is that the "UsedCars" will select all the vehicles that its condition is "Second Hand", apart from that everything is identical.



### **Fig.5 – Used Cars Back End Code**

## CarsSingleView.

This page will be used to display all the details for the selected vehicle either from the NewCars or from the UsedCars.

Below (in parts) one can find the business logic used in the "CarsSingleView" page.

### Part 1 -CarsSingleView - Page\_Load:

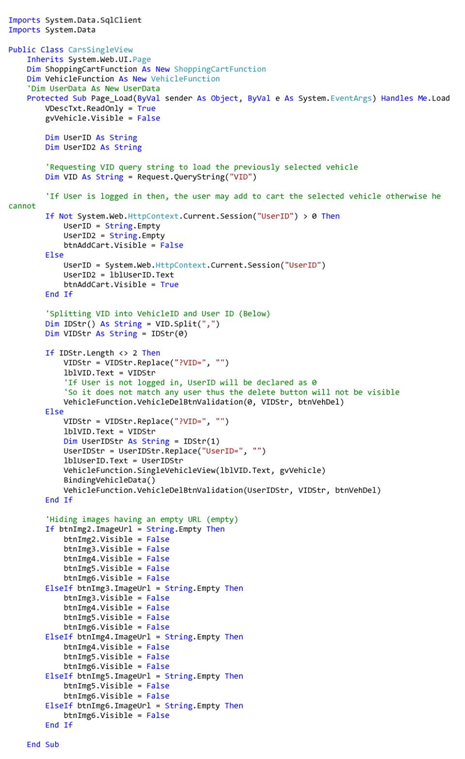


Fig.5 – CarsSingleView - Part1 - PageLoad

Firstly, we'll start by instantiating "ShoppingCartFunction" and "VehicleFunction", these will be both declared right after the declaration for the CarsSingleView class. By instantiating both classes globally will enable us to use them in the whole class and not limit us to use them just in a function or event only.

Secondly, we will being requesting the Vehicle ID which was passed from the previous page (either "NewCars" or "UsedCars") in order to get the selected vehicle and display its details in the CarsSingleView page, shown in Fig.6.



Fig.6 – Requesting the Vehicle ID

After displaying the selected vehicle we will be validating whether the user is logged in or not, thus if the user is logged in he/she may add the vehicle to the cart, but if not, the user will not be able to add any vehicles until logged in, this process is shown in Fig.7.

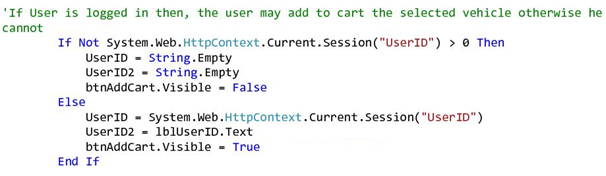


Fig.7 – Verifying if the user is logged in to avail btnAddToCart

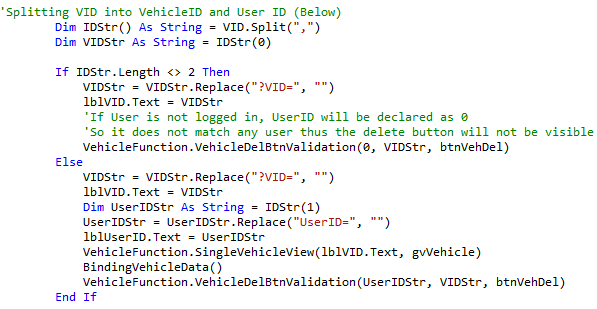


Fig.8 – Verifying UserID to enable the Delete Button

In Fig.8 we are illustrating the process where our program will split the requested vehicle ID shown in Fig.6. The vehicle ID string will contain both the VID and user ID, this is because we want to determine whether the current logged user is the one who created the selected vehicle. If the user matches as the person who created the vehicle then the delete button will be available for the user's perusal.

The "BindingVehicleData()" function, will be used to bind the data of the selected vehicle with the gridView in the current page. This function is explained in more detail below.

The below Fig.9, illustrates the process of hiding the image buttons that have an empty URL.

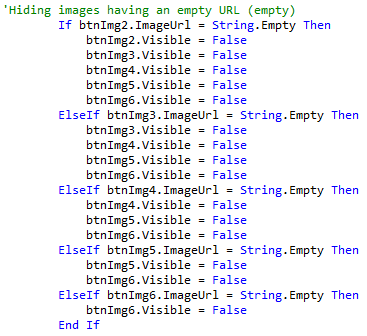


Fig.9 – Verifying UserID to enable the Delete Button

### Part 2 -CarsSingleView - BindingVehicleData():

The "BindingVehicleData()" function will be used assign the fields (text boxes, labels and main image) the data retrieved from the gridView (through gridView.FindControl). however the main image and other alternate images will be uploaded through the (AltImgBinding()) mentioned below in Fig.10.



Fig.10 – BindingVehicleData()

### Part 3 -CarsSingleView - AltImgBinding():

This function will be used to load the images from the "Images" folder (stored in the solution), and to set the image size accordingly. The main image will be directly shown in the "LargeVImg" image button, whilst the other images (up to six images in total) will be uploaded in the smaller image buttons (btnImg2 - btnImg6). The alternate images will be stored as a comma delimited string (Jaguar XF - SideRear.jpg,Jaguar XF - Int.jpg,Jaguar XF - Front.jpg), then they will be split accordingly.



Fig.11 – AltBindingVehicleData()

### Part 4 - CarsSingleView - btnImg\_Click:

The below events will fire once the button image (btnImg1-6), will be selected. Upon button click the selected button will be uploaded in the large image button (LargeVImg), for better viewing.

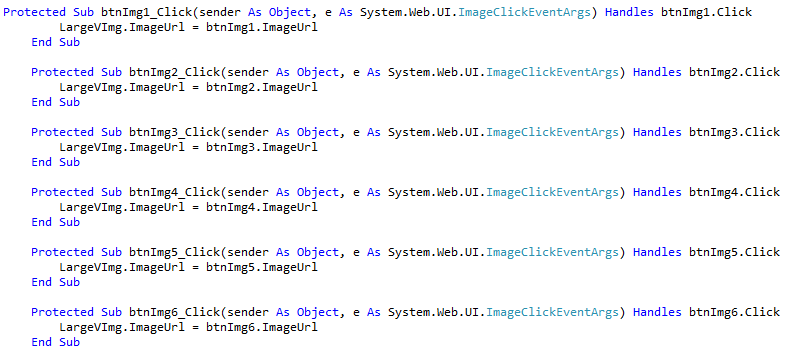


Fig.12 – btnImg\_Click

### Part 5 -CarsSingleView - Deleting a vehicle:

The below event (which will be fired upon delete button (btnVehDel) click), will be used to delete a vehicle. Upon button click the user will be prompted to confirm the selected vehicle's deletion, if he presses the "Yes" button the "delQuestion" will return "6" as a value and will proceed to the "VehicleDel()" function. If the user presses "No" then the "delQuestion" will return "7" as a value and will stop the execution and return to "PageLoad()". Below in Fig.13 this process is illustrated accordingly, whilst Part 6 will define the "VehicleDel()" function.

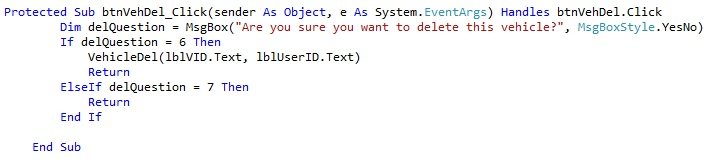


Fig.13 – AltBindingVehicleData()

Part 6 -CarsSingleView - VehicleDel()**:**

This functioned as already mentioned before, will be used to delete a vehicle. This function uses a "Delete" statement to delete the vehicle along with its images, obviously the images will also be deleted from the "Images" folder. This function is shown in Fig.14.

****

Fig.14 – AltBindingVehicleData()

Part 7 -CarsSingleView - btnAddCart\_Click:

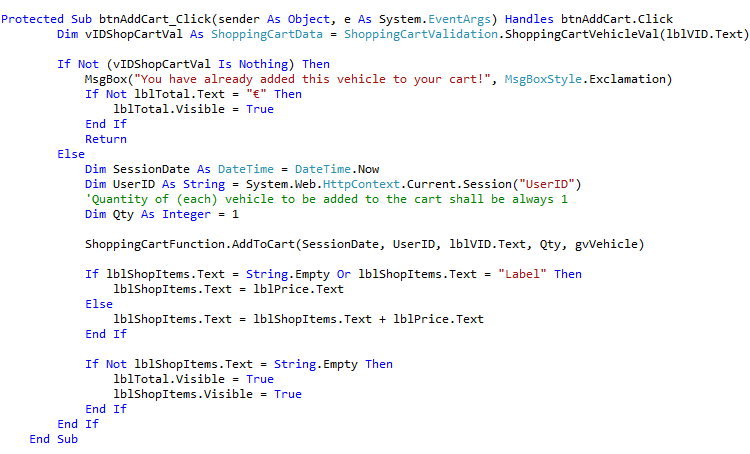
****

Fig.15 – btnAddCart\_Click

The previous figure (Fig.15) shows the whole " btnAddCart\_CLick" code below we will be explaining the whole code behind the button click event.

This event will fire when the user is logged in and clicks on the "Add to Cart" button namely "btnAddCart". However the user will not be able to click on the button if he's not logged in.

"vIDShopCartVal" will be created to validate the vehicle selected, therefore this instance will be used to access "ShoppingCartValidation" namely the "ShoppingCartVehicleVal". This functioned is described below.

### ShoppingCartVehicleVal:

Basically this function through a select statement will select the vehicle/s from the shopping cart where the vehicle ID matches, through an iteration. Shown in Fig.16.

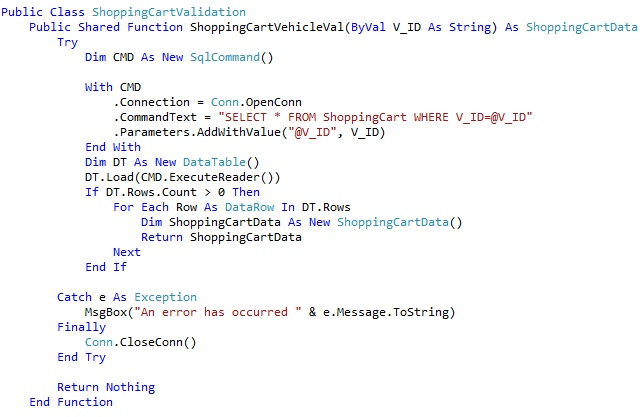


Fig.16 – ShoppingCartVehVal()

Through the ShoppingCartVehicleVal(), If the user has already added the vehicle to his cart, our solution will prompt him "That the vehicle is already added in his cart" and will stop execution of the program. If the vehicle is not present in the user's cart, then the "ShoppingCartFunction.AddToCart()" function will be used to add the vehicle to the cart.

Therefore this function will use an insert statement to insert the selected vehicle/s to the shopping cart (table).

This function is illustrated in Fig.17, and may be accessed from our solution "AppCode/ShoppingCartFunction/AddToCart().

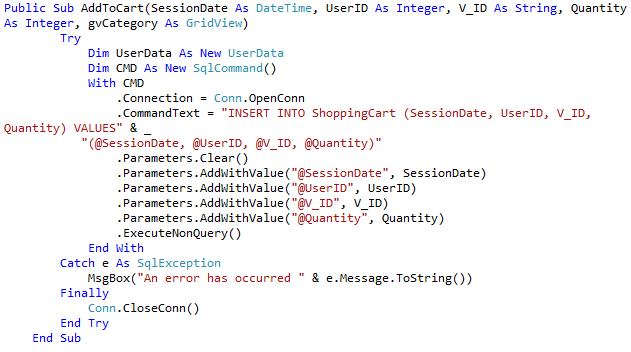
****

Fig.17 – AddToCart()

## CustomerCars.

The CustomerCars page will be utilised so that the customer can view the vehicles he add, thus his listings.

### Part1 - CustomerCars - Initialisation:

First we will start by instantiating both "VehicleFunction" and "VehicleData", these instance may be called globally throughout the page. "VehicleFunction" class will be used to get certain functions we will require for this page, and the "VehicleData" will be used as the "Getter" and "Setter" class for the Vehicle table.

### Part2 - CustomerCars - Page\_Load:

On page load we will validate the following:

**Validation Points on Page\_Load:**

Whether the user is logged in:

If the user is logged then he the solution will resume the execution, otherwise will halt the user and redirect to "Login.aspx" page.

If the user listed any vehicles:

If the user is logged in, then we will call the function "UserVehicles" from the "VehicleFunction" class (illustrated below in Fig. 18), this function will be used to select all the vehicles that were created by the current user and populate the gridView with the results (if any).

Then we will validate whether he has listed any vehicles or not, if he hasn't listed any vehicles yet, our website will notify the user. However if the user added any vehicles they will be displayed in the gridview "gvVehicle" included in the CustomersCars page.

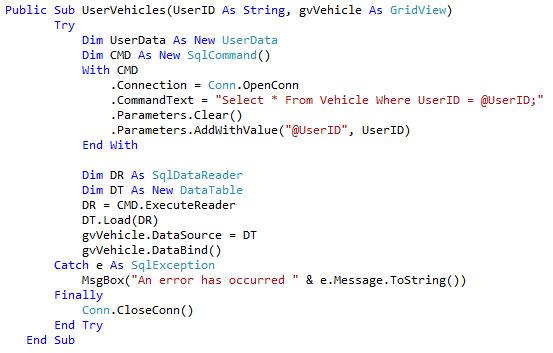


Fig.18 – VehicleFunction.UserVehicles Function

### Part3 - CustomerCars - gvVehicle\_RowDeleting:

The " gvVehicle\_RowDeleting" event will be fired when the user clicks the "Delete" link in the gridview.

Upon firing of the event the solution will prompt the user whether he really intends to delete the selected vehicle. If the answer is yes (6), then the solution will get the selected vehicle ID "V\_ID" from the gridview, and check whether the vehicle to be deleted is sold or not. However, if the vehicle will be marked as sold on the Vehicle Table then the user will not to be able to delete it, for (our) record purposes. On the other hand if the vehicle is not sold, then the solution will continue deleting the vehicle and refreshing the "gvVehicle" gridview. The function that will allow the user/s to delete a vehicle is the "VehicleDel" function which may be accessed from the "VehicleFunction" class.

The "VehicleDel" function includes a delete statement, which consequently will delete the selected vehicle. This function is shown below in Fig.19.

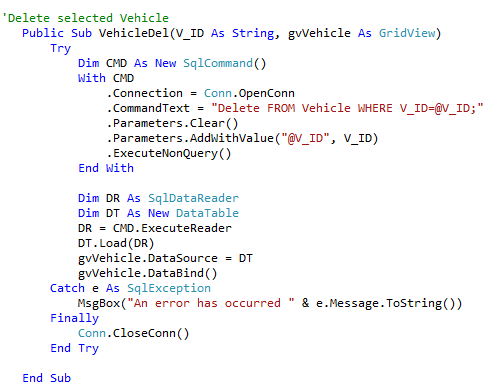


Fig.19 – VehicleFunction.VehicleDel Function

### Part3 - CustomerCars - gvVehicle\_SelectedIndexChanged:

Finally this event will fire upon the user's "Select" link button (embedded in the gvVehicle), and will continue to load the selected vehicle in the "CarsSingleView.aspx" page. The "CarsSingleView " page was already explained previously in the "CarsSingleView" section.

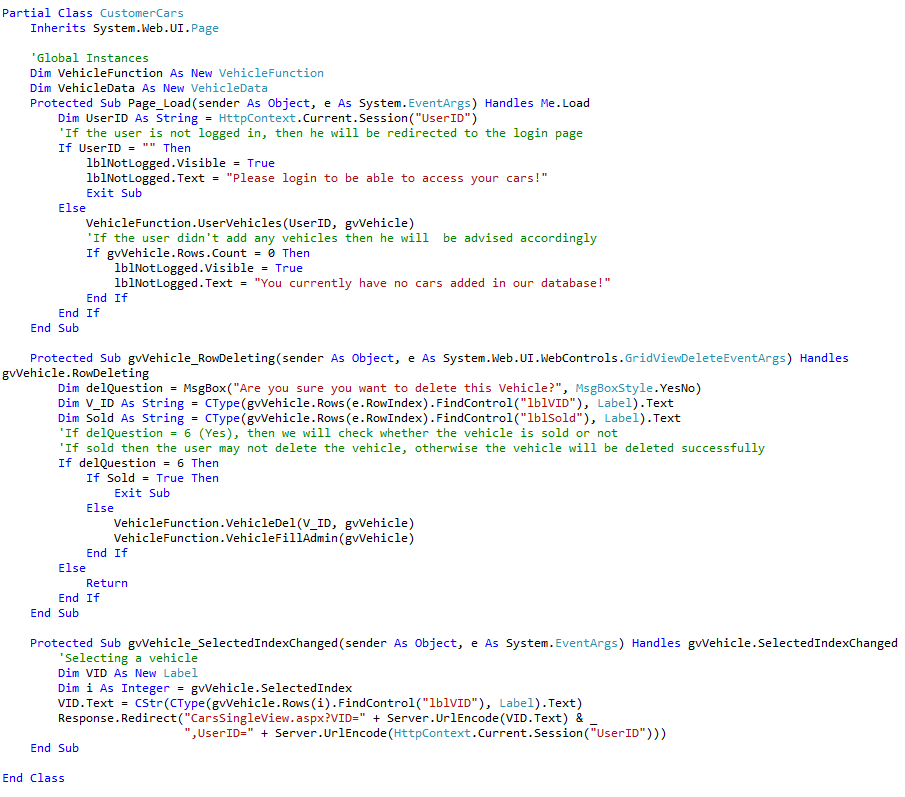


Fig.20 – CustomerCars Page

## VehicleSearch.

The VehicleSearch page aka: "CarSearch.aspx", is a page where the user may search for a vehicle, the search terms may include either the make (Toyota), model (Auris) or condition (New). The back end code behind the search bar is located in the MasterPage, in order to be accessed no matter on which page the user is navigating.

Upon pressing the search button, our solution will encode the search text (inputted in txtbSearch) and will navigate to the "CarSearch" page.

The code behind the search button click "btnSearch\_Click" is as follows (Fig.21).



Fig.21 – SearchBtn\_Click Event

### Part1 - CarsSingleView - Page\_Load:

Initially on page\_load we will be requesting the "SearchString" previously passed from the search bar. Upon requesting the "SearchStr" we will be validating for the following eventualities:

**Validation Points on Page\_Load:**

1. Whether the "SearchStr" is empty, thus the user just clicked on the search button without entering any text.
2. Whether the search term the user used is found in our database.

On both validation points the user will be advised if the condition of the above points result to true.

However if point 1 results to "False" then we will run the "VehicleFunction.SearchQuery" function.

This function (shown below in Fig.22), will select the details from the Vehicle table (through a select statement), and will search on either make, model or condition. Finally the results of the "SearchQuery" will be populated on the "gvVehicle" gridView. Otherwise if no data is found, then validation point 2 will result to "True".

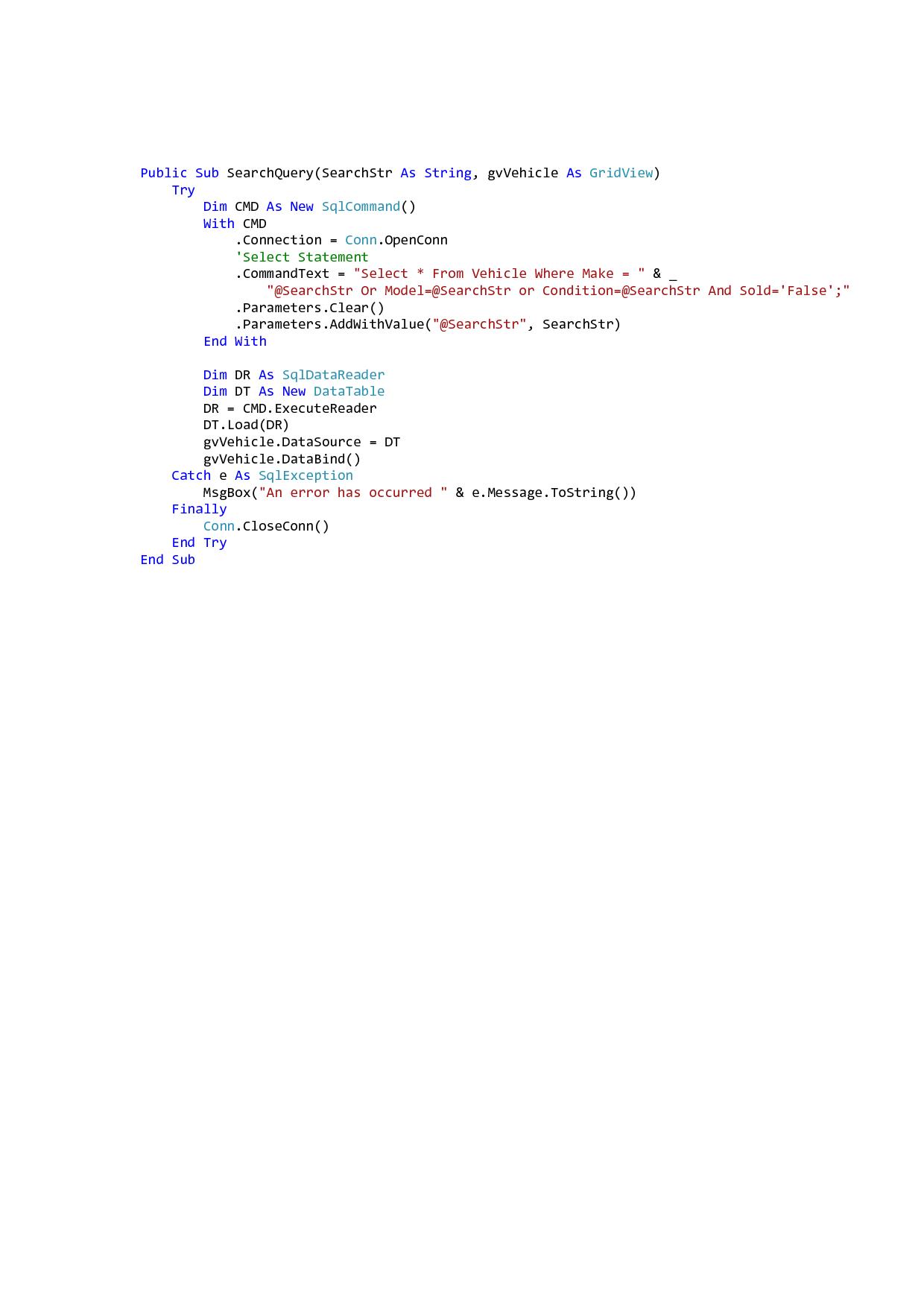


Fig.22 – VehicleFunction.SearchQuery()

The gvVehicle\_SelectedIndexChanged is identical as in the "NewCars, UsedCars, MyCars" sections. Below one can find the full code of the "CarSearch" page.

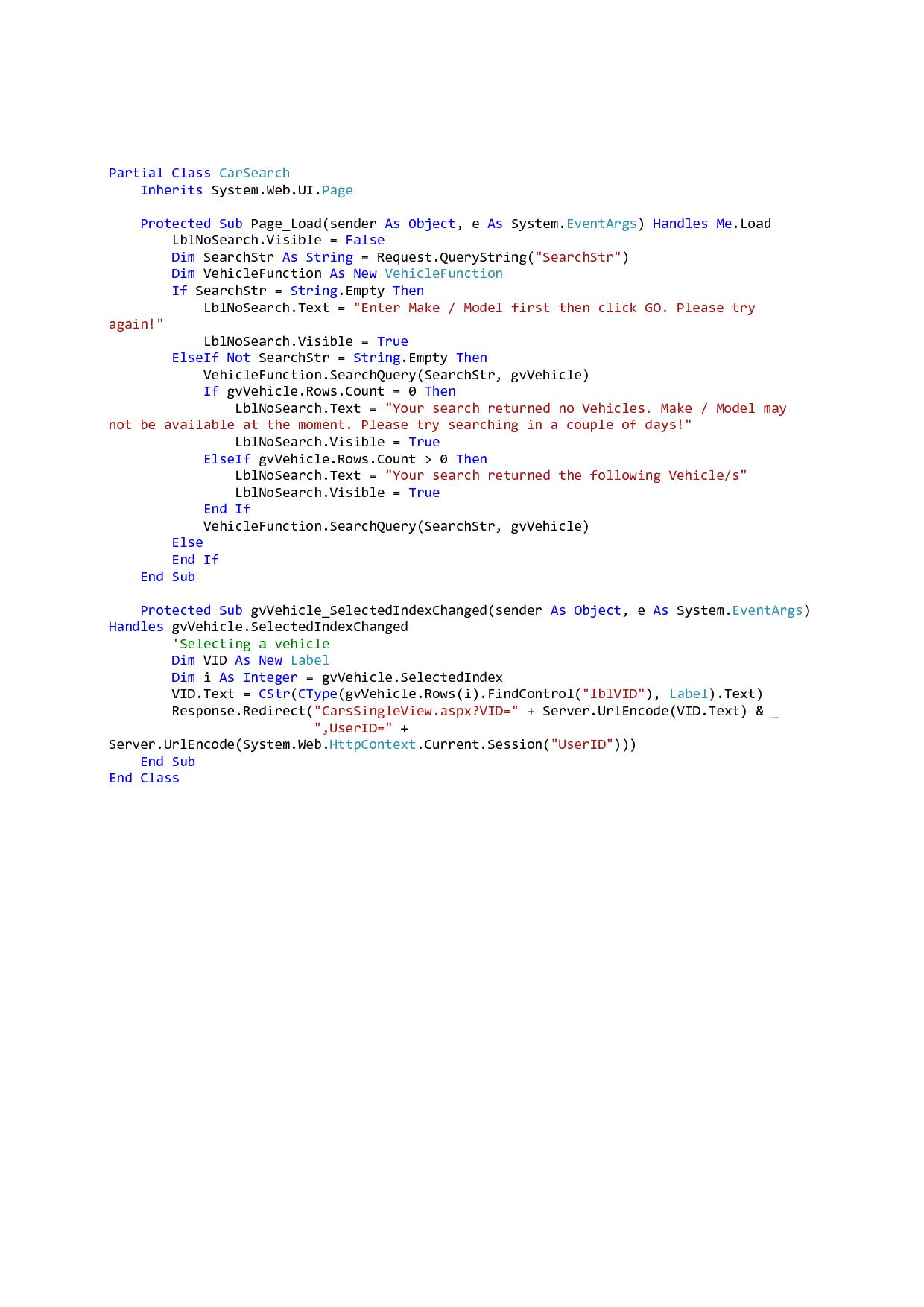
****

Fig.22 – CarSearch Page

## Register.

If the user is not yet registered, the user may browse our vehicles but is restrained only to that, since only registered users can buy a vehicle. Therefore the "Registered" page will allow users to create an account with us. To register, the user must enter the details accordingly, in the textboxes provided. Below we are including the validation points upon pressing the register button (regBtn).

### Part1 - Register - Page\_Load:

**Validation Points on regBtn\_Click:**

1. Our solution will validate the below mentioned textboxes if left empty:
   1. Blank name (NameTxt)
   2. Blank surname (SurnameTxt)
   3. Blank UserName (UserNameTxt)
   4. Password (PWDtxt)
2. Another validation point will check whether both passwords inputted by the user in the password textboxes (PWDtxt & PWDMatchtxt) match. If the passwords do not match the program will warn the user and stop execution, on the contrary if the passwords match, the solution will continue.
3. The next validation point will verify, whether the combination of username and password inputted by the user exists in our "User" table and whether the username (only) exists on the same table. This will be done as follows:
   1. The "UserValidation" class will be instantiated, following this the UserVal() will verify if username and password exists in our database, and UserNameVal() will verify if the username(only) exists in the User table. Both functions are illustrated below in Fig.23 and Fig.24 respectively.

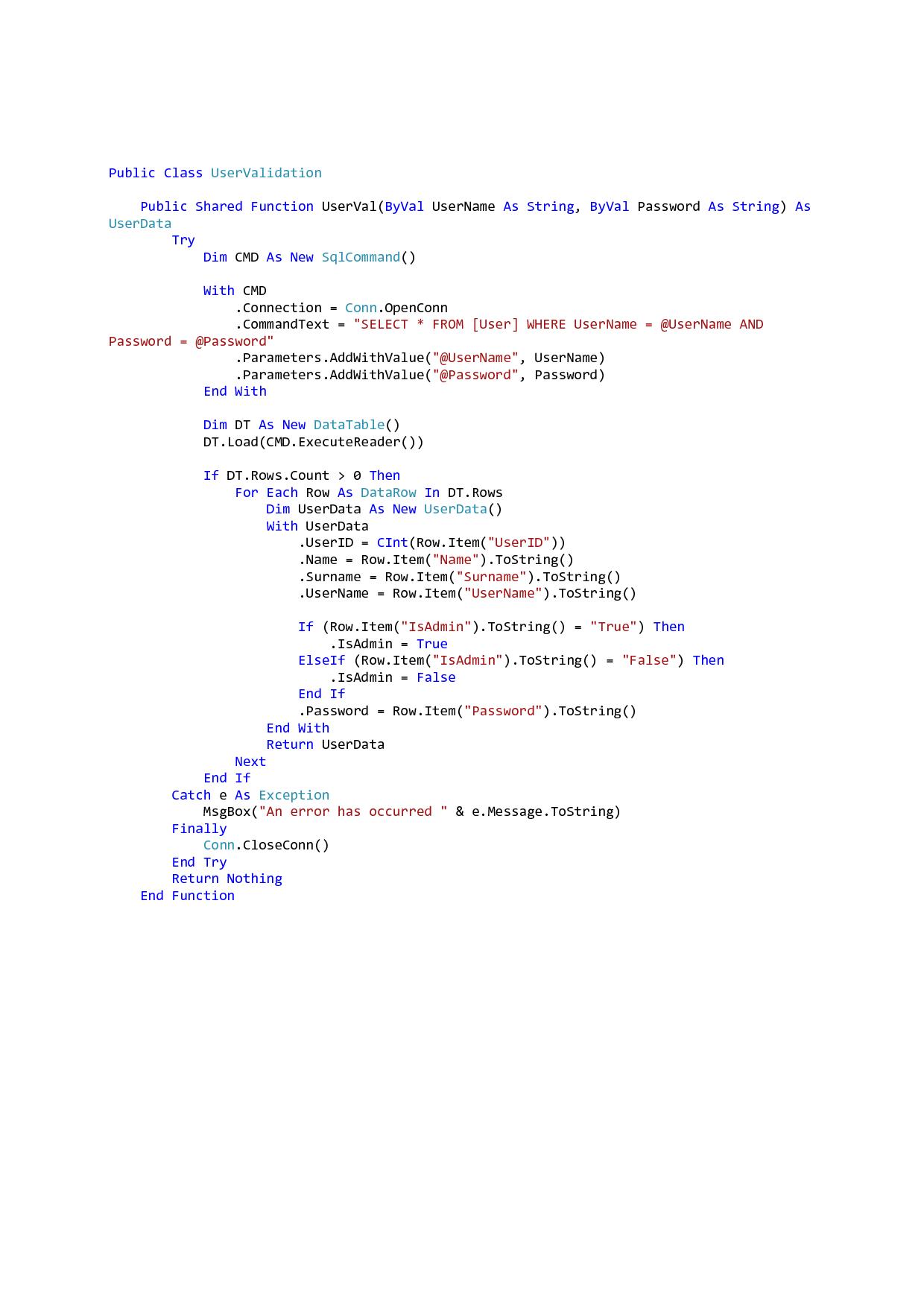


Fig.23 – UserValidation.UserVal()

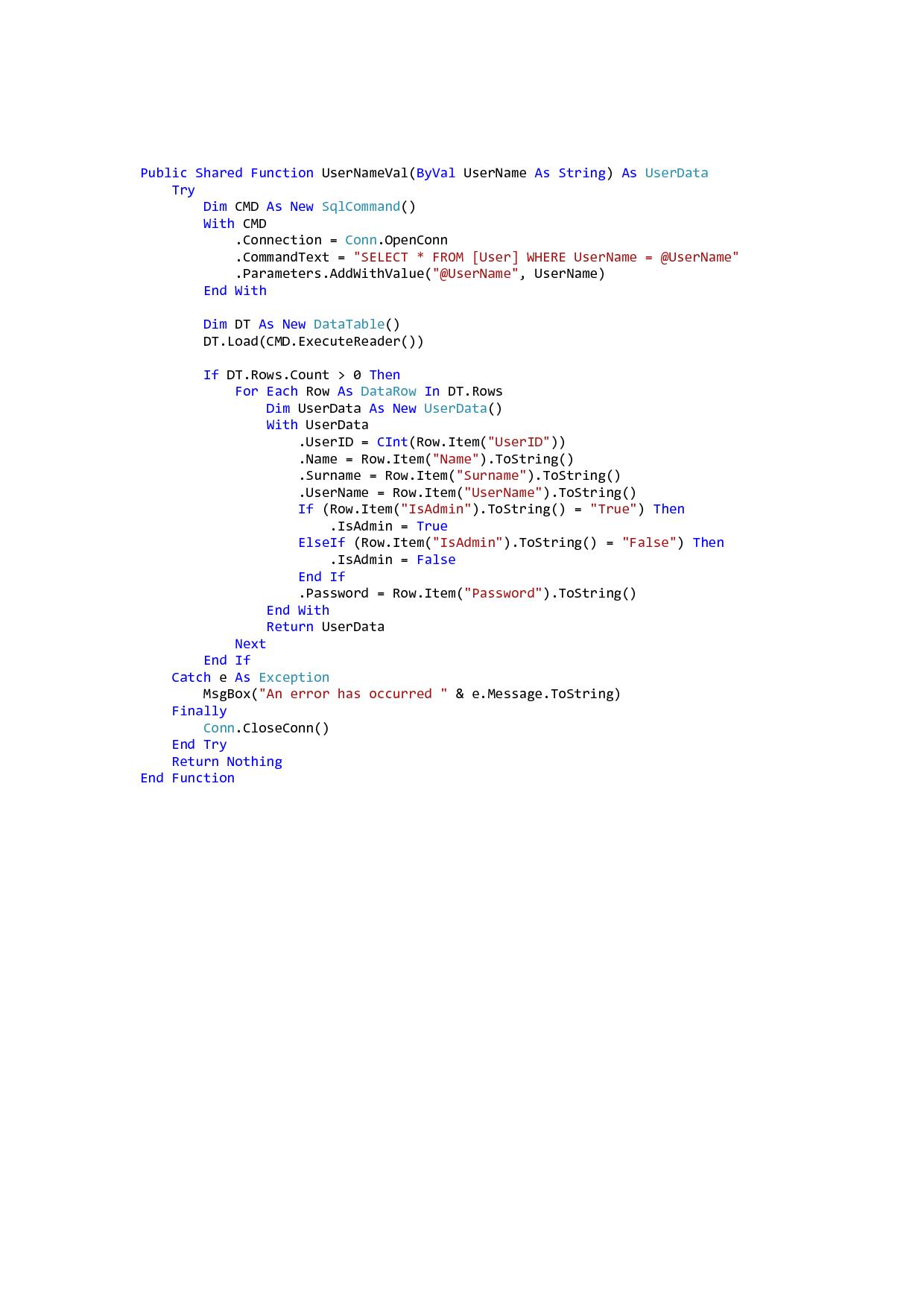


Fig.24 – UserNameValidation.UserVal()

If the above mentioned validation points do not execute, therefore the data inputted is correct, then the program will continue to add the user to the "User" table accordingly. This will be done by calling the "RegUser()" function from the (instantiated) class "UserFunction".

The "RegUser()" function (shown in Fig.25) will retrieve all the data from the textboxes "Register" page and will insert this data into the "User" table. This function is illustrated in Fig.26, and can be accessed from the "UserFunction" class.

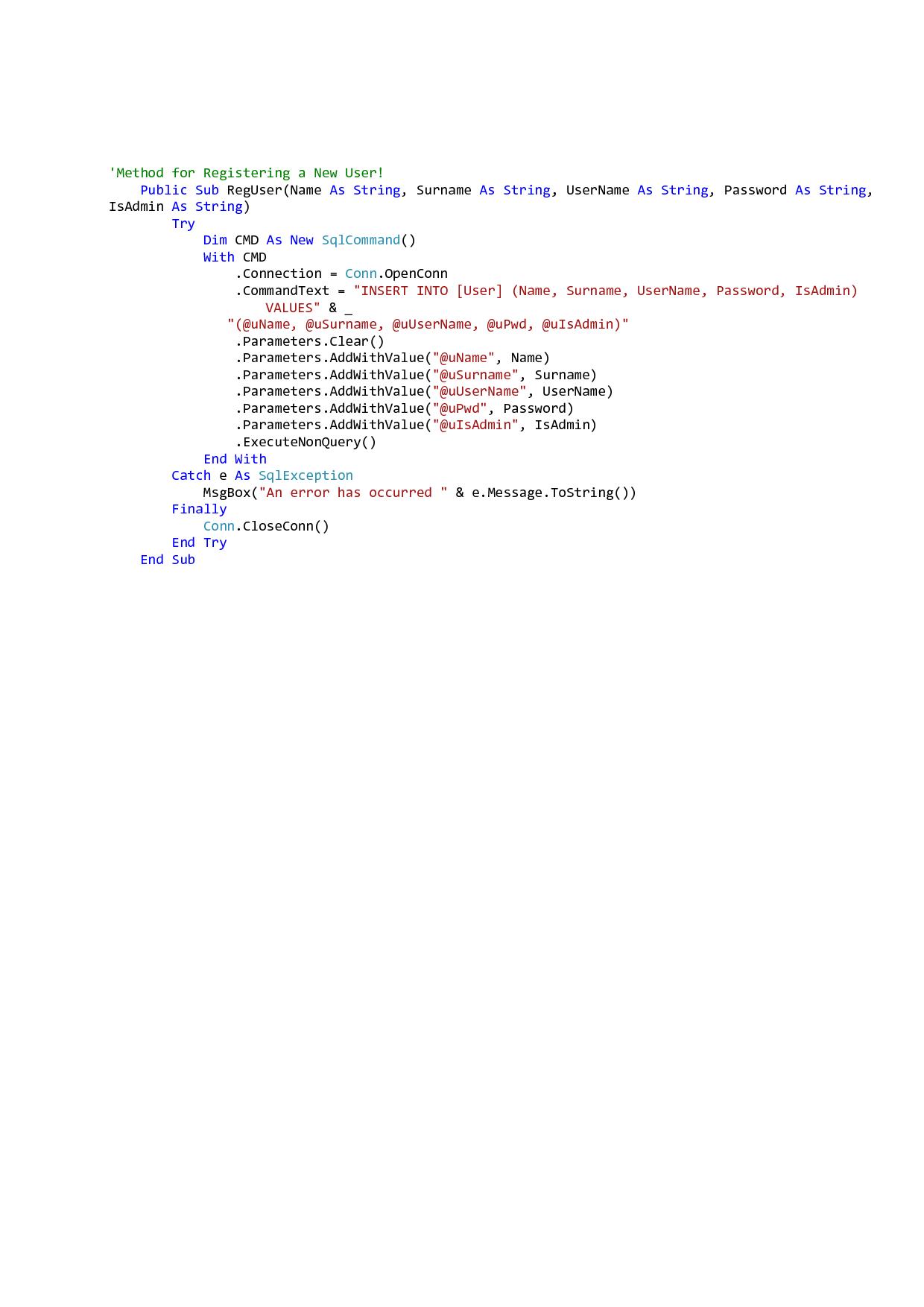
****

Fig.25 – UserFunction.RegUser()

# C:\Users\Montesin.Montesin-PC\Desktop\A&A AutoTrading\ScreenShots\BackEnd Pages\Register\RegisterPage1.jpg

Fig.26 – Register Page

## Login, Admin Panel, User Panel and User Logged.

The Login page will be the actual login page, this page contains various functions that will lead to the Admin Panel / User Panel, depending on whether the logged in user is an administrator or not, and will also update UserLogged link button.

### Part1 - Login - Page\_Load:

The Page\_Load in our case doesn't contain any notable code, since it contains only some properties for certain labels, these properties may including disabling visibility to selected labels.

### Part2 - Login - btnLogin\_Click:

When the user click on the login button "btn\_Login" , the clickable event will fire accordingly. We will start this event by instantiating the "UserData" class. Secondly to make sure that there will be no conflict between sessions, we added the option to remove all sessions upon button click.

If the UserID string is empty then we will validate the username and password textboxes (txtUserName and txtPWD) through the "UserValFields()" function. The "UserID" string is a string that will be passed from the "ShoppingCart" page, which will be documented accordingly later on.

**UserValFields():**

This function will provide the necessary validation on the username and password textboxes (txtUserName and txtPWD). Therefore the validation points will validate the following:

**Validation Points on btnLogin\_Click:**

1. Will validate on blank username and password textboxes.
2. Will validate whether the username and password entered exist in the "User" table.
3. If the user didn't pass the validation points included in the UserValFields(), then the program will refrain from redirecting the user to either the "UserPanel" or "AdminPanel".

If the above mentioned validation points result in "False" then our solution will continue by adding the sessions for the "UserID", "UserName" and "IsAdmin".

The event "btnLogin\_Click" is illustrated in Fig.27, whilst "UserValFields()" function is shown in Fig.28 .



Fig.27 – btnLogin\_Click - Part 1 And Part 2

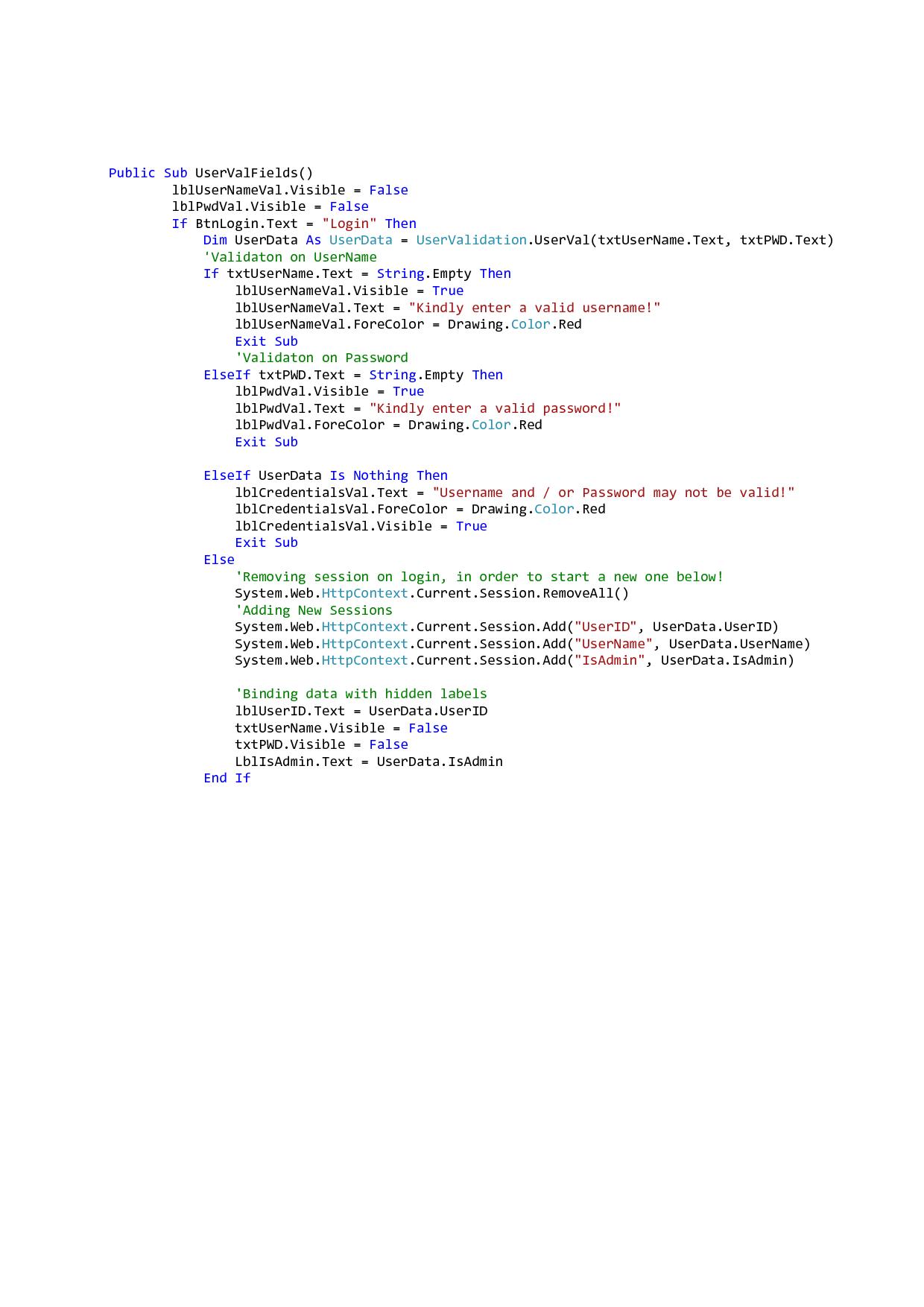


Fig.28 – UserValFields()

### Part2 - Login - btnLogin\_Click - Continued:

If the user passes all the validations in the "UserValFields()" then if the user will be redirected to either the "UserPanel" or to the "AdminPanel" pages, obviously this depend on whether the user is an admin or not. (Both the "UserPanel" and "AdminPanel" will be documented further below. N.B: Part 3 will be documented later on in the ShoppingCart section.

## 

## UserPanel:

If the user is a normal user, thus he is not an administrator, when he logs in to our website, the solution will redirect him to the "UserPanel".

The "UserPanel" shown in Fig.29, will enable the user to add a vehicle, delete a vehicle and view his past orders.

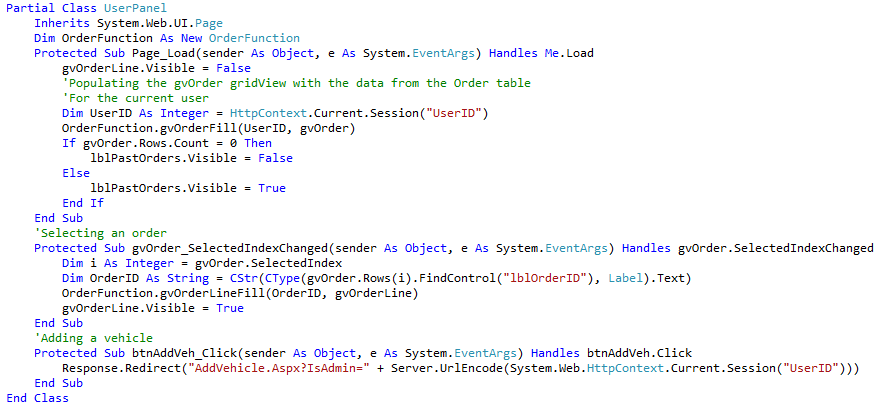


Fig.29 – UserPanel page

### Part1 - UserPanel - page\_Load:

On page load, the solution will retrieve the "UserID" of the current user from the session, in order to be able to pass this ID to the "OrderFunction.gvOrderFill()" function. This function depicted in Fig.30 will be used to select all the orders, made by the current user (if any), and then populate the gridview with the results.

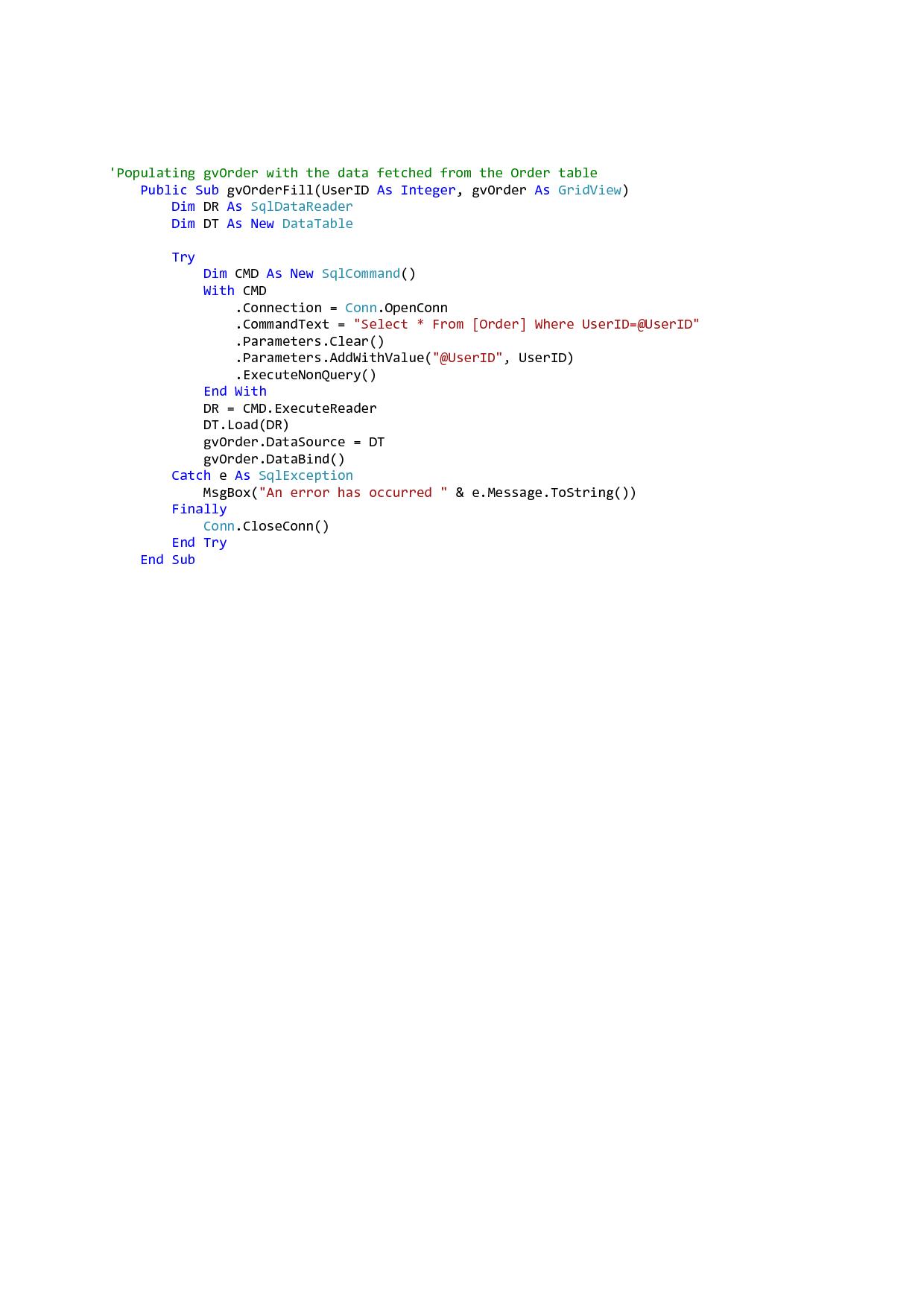


Fig.30 – OrderFunction.gvOrderFill() Function

### Part2 - UserPanel - gvOrder\_SelectedIndexChanged:

After the "gvOrderFill()" function was executed and the "gvOrder" was populated accordingly with the results, the user may now select an order from the same gridview. The selected order will then be populated in the "gvOrderLine" gridview.

### Part3 - UserPanel - btnAddVeh\_Click:

This event will fire when the user clicks the add vehicle button, namely "btnAddVeh". This snippet will then redirect the user to the "AddVehicle" page and will encode the "IsAdmin" string which contains the "UserID", to be later validated in the "AddVehicle" page. This feature is also available in the "AdminPanel" so that the administrators will be able to add vehicles.

## AddVehicle:

The add vehicle option was intended only for administrators, however we decided to add this option as well for the user as an extra feature. This extra feature will enable (normal) users to add their own vehicles as well.

### Part1 - AddVehicle - page\_Load:

On page\_Load (Fig.31) we will call the "DropDownProperties()" function in order to populate the dropdown lists included on the "AddVehicle" page.

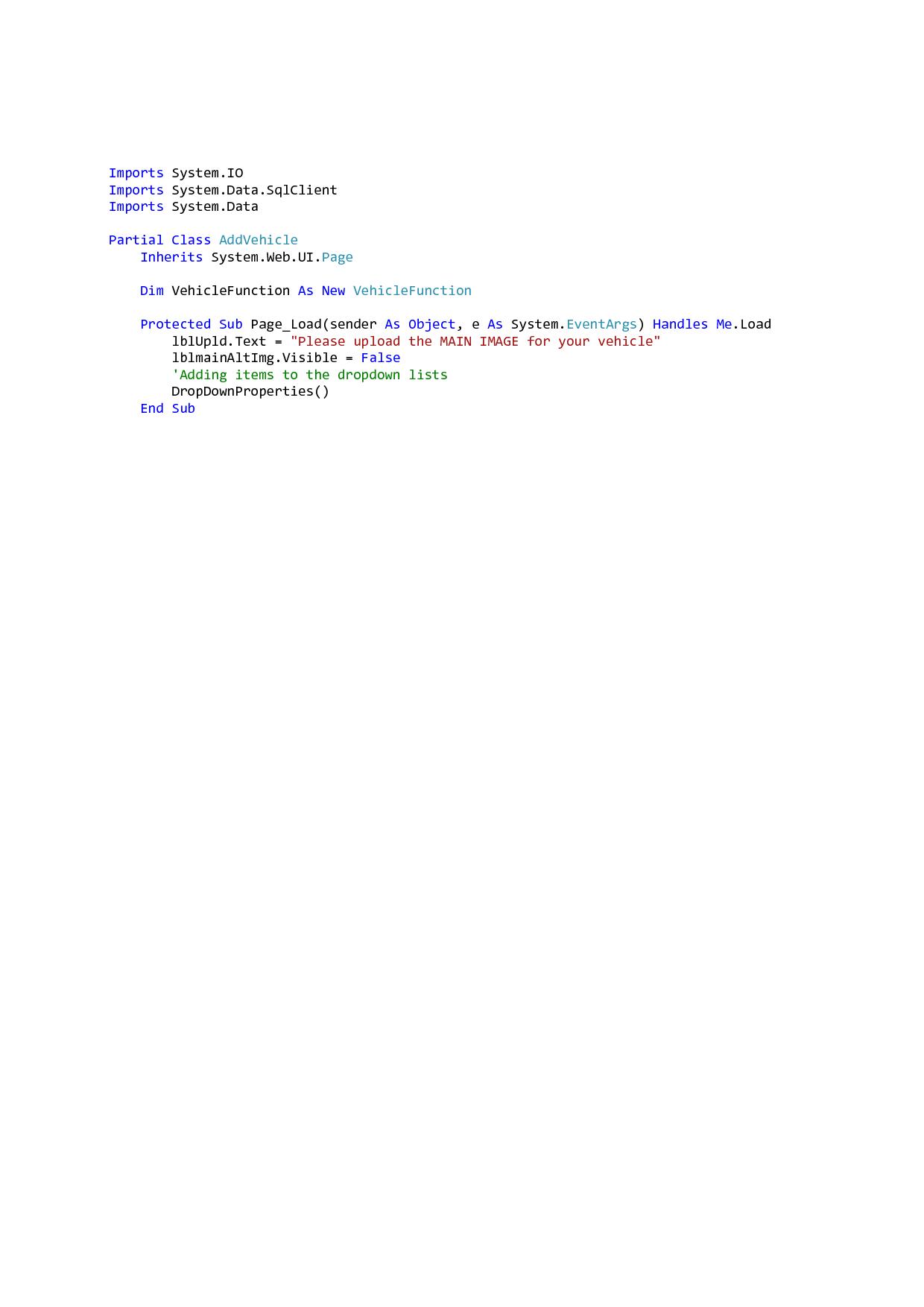


Fig.31 – AddVehicle page\_Load

### Part2 - AddVehicle - DropDownProperties():

This function will be used to populate the dropdown lists with the correct data, accordingly.

The "ddlYr" which will represent the vehicle's year of manufacture, will be populated through a "Do, Until" iteration, with years starting from 1950, up to the current year. All the other drop down lists available will be each populated with the correct data.

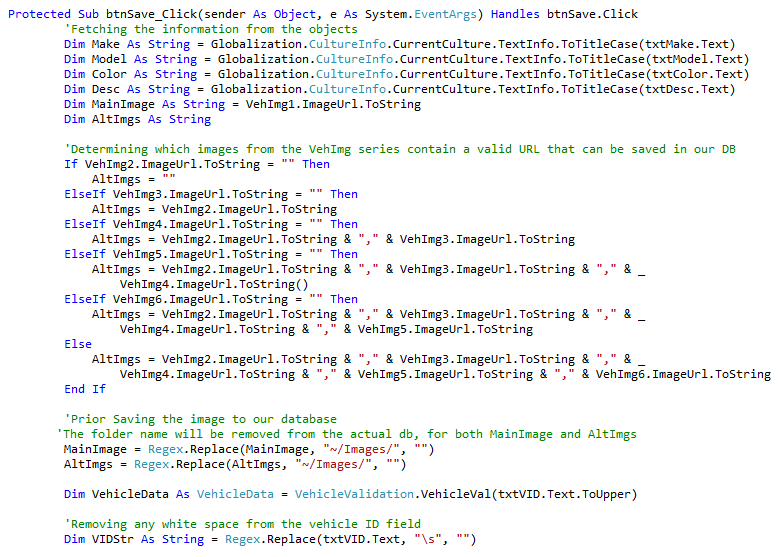


### Fig.32 – DropDownProperties()

### Part3 - AddVehicle - btnSave\_Click - Part 1:

In the first part of this function our solution will retrieve the text entered in the vehicle property textboxes. Though we want to keep our "Vehicle" table as clean as possible, we thought that this option  would help us to keep the data standardised. The globalization option will be used to convert the text fetched from the text boxes to title case.

The second option in the button click event will determine the images which do not have an empty URL, then these images will be concatenated in a comma delimited format. This format will help us to save all the alternate images as one whole string in the "AlternateImgs" column in the "Vehicle" table. In order to save the images in a clean format, the folder name will be removed from the image's URL prior saving them.



### Fig.33 – btnSave\_Click - Part1

**Validation Points on btnSave\_Click:**

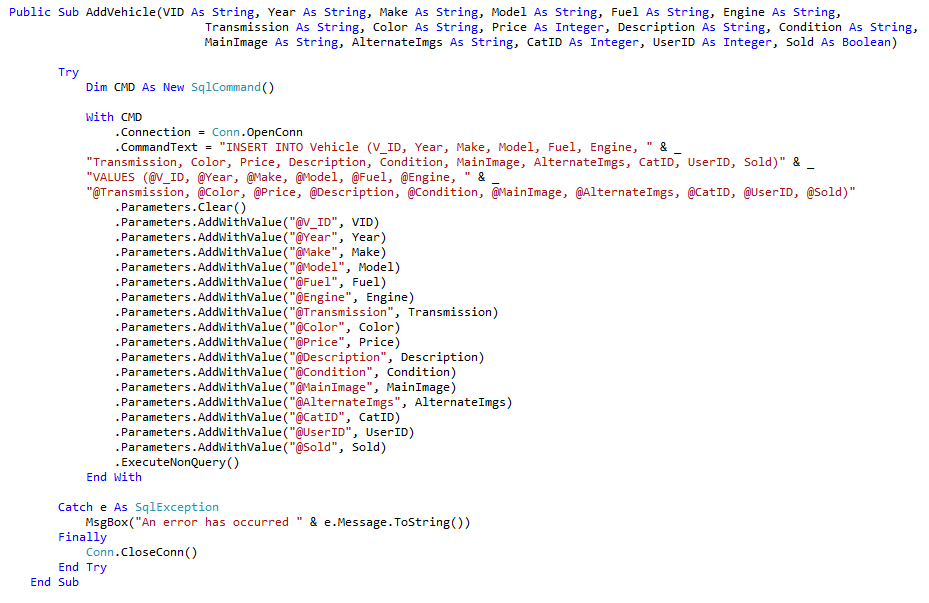
1. We will now validate whether the vehicle ID (V\_ID) entered by the user is already existent in our database. This validation will be processed through the "VehicleVal()" function found in the "VehicleValidation" class, this function (through a select statement) will take the vehicle ID inputted by the user and search through the "Vehicle" table. This function is illustrated in Fig.34.
2. Validation on all text boxes, drop down lists and other fields that require user input, will be validated accordingly. This validation will occur to avoid blank data and erroneous data.



### Fig.34 – btnSave\_Click - Part2

If all data will be correct, then after the validation process, the solution will carry on to actually add the vehicle to the "Vehicle" table. The function that will add the vehicle to the table is the "AddVehicle()" which is located in the "VehicleFunction" class. This function will update the "Vehicle" table with the data inputted by the user.

Fig.35 illustrates the "AddVehicle" function.



### Fig.35 – VehicleFunction.AddVehicle()

## AdminPanel:

The "AdminPanel" will be used by administrators, to view, add and delete vehicles, add, edit and delete categories and to view users. All functions may be accessed by clicking on the buttons provided in the "AdminPanel".

The view, add and delete functions are identical to the ones available in the "UserPanel" with one difference. The only difference is that the administrators may delete all vehicles added by all users and all administrators that are not marked as sold, while the user may only delete the vehicles he added.

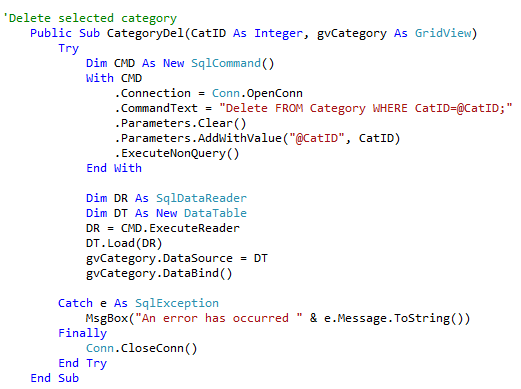
The administrators may also add, edit and delete the categories (which are not associated with any vehicles).

### Part1 - AdminPanel - gvCategories\_RowDeleting:

The below function will be used to validate, whether the category to be deleted is still associated with any vehicle, which if no will finally will result in deleting the selected category. The function that will actually delete the selected category from the "Category" table is the "Category Del" which may be accessed from the "CategoryFunction" class. Both " gvCategories\_RowDeleting" and "CategoryDel" are shown in Fig.36 and Fig.£7 respectively.

### 

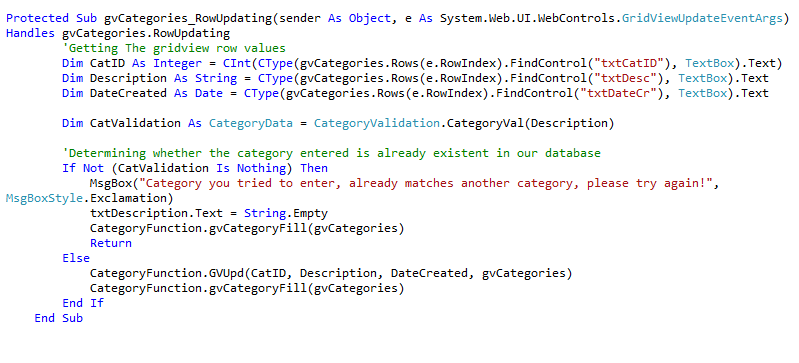
### Fig.36 – gvCategories\_RowDeleting



### Fig.37 – CategoryFunction.CategoryDel()

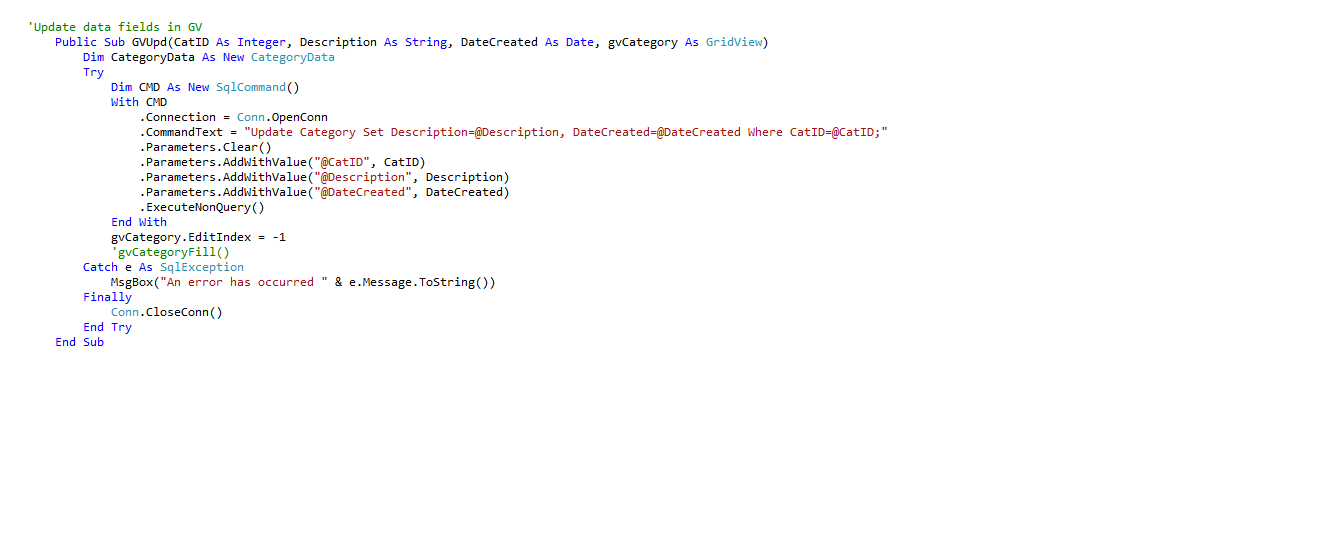
### Part2 - AdminPanel - gvCategories\_RowUpdating:

The Admin will also be able to update/edit categories (one by one), this process is shown below in Fig.38.



### Fig.38 – gvCategories\_RowUpdating

The function "GVUPD" (illustrated in Fig.39), will use an update statement, which will retrieve the inputted category and update the category table accordingly.



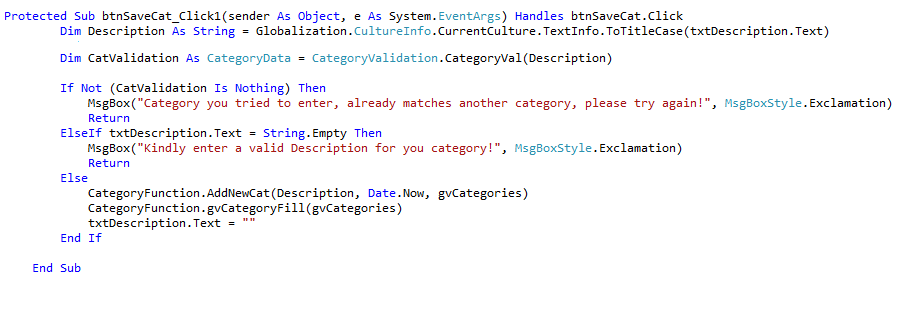
### Fig.39 – CategoryFunction.GVUpd()

### Part3 - AdminPanel - btnSave\_Click:

Upon adding any categories "btnSave\_Click", (shown in Fig.40) the inputted category will be validated and therefore these validation points will include:

**Validation Points on btnSave\_Click:**

1. Will validate and verify whether the category to be saved already exists.
2. Will validate whether the "txtDescription" (category description) is left blank or not.



### Fig.40 – btnSaveCat\_Click

Upon finishing the validating process then the "AddNewCat" function available in the "CategoryFunction" class will add the category accordingly. This function is displayed in Fig.41.



### Fig.41 – CategoryFunction.AddNewCat()

## ShoppingCart.

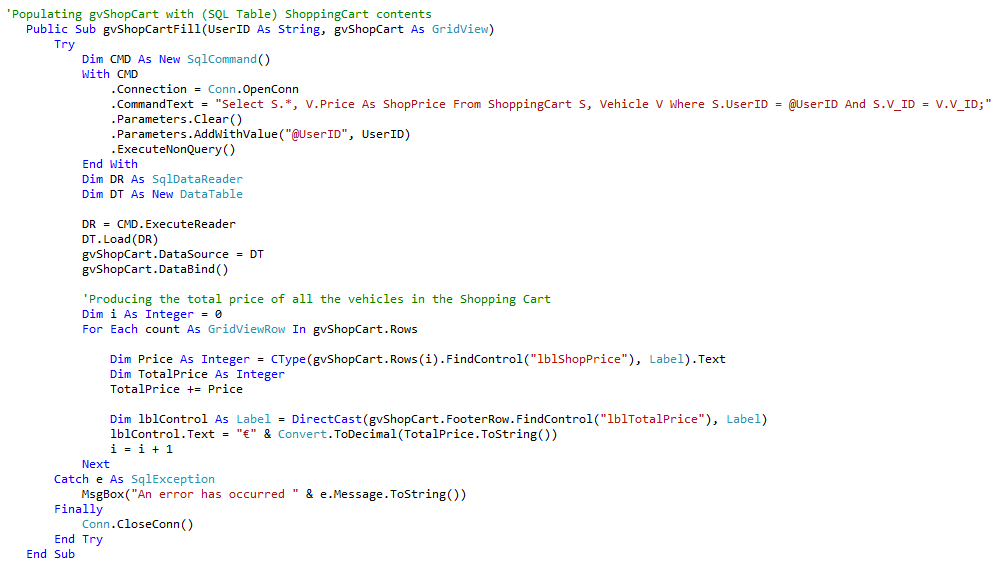
The shopping cart will be used by the user to add a vehicle (he intends to buy) to it. The shopping cart has certain features that make it more secure than other pages. Below we will be adding a full description of the "ShoppingCart" page.

### Part1 - ShoppingCart - page\_Load:

First we will start by instantiating the "ShoppingCartFunction" and the "OrderFunction" then we will be adding a decimal variable "decPrice" and a string variable "CredentialMatch". Both instances and variables will be used globally, thus throughout the whole page.

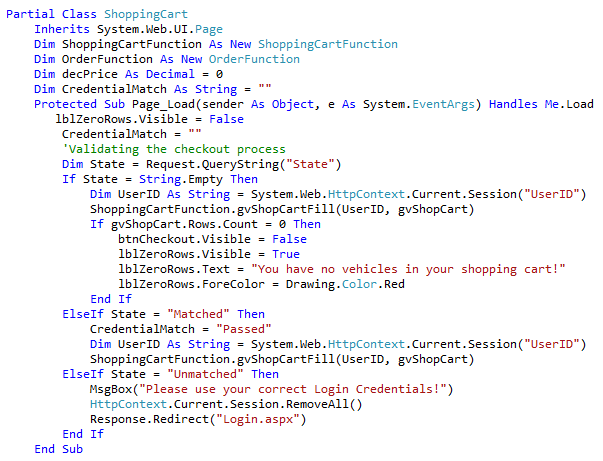
**The "CredentialMatch" string and "State" string will be used in conjunction with the "btnCheckout".**

On page\_Load our solution will be checking if the user has any items in his shopping cart, this will be done through the "gvShopCartFill()" function (shown in Fig.43) and may be accessed from the "ShoppingCartFunction" class.



### Fig.42 – ShoppingCartFunction.gvShopCartFill()

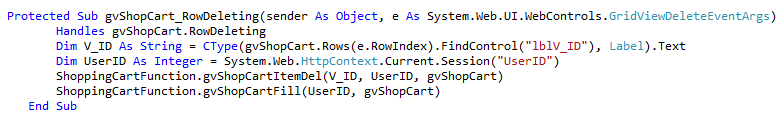
If the shopping cart of the current user does not contain any vehicles then the user will be prompted accordingly.



### Fig.43 – ShoppingCart page\_Load

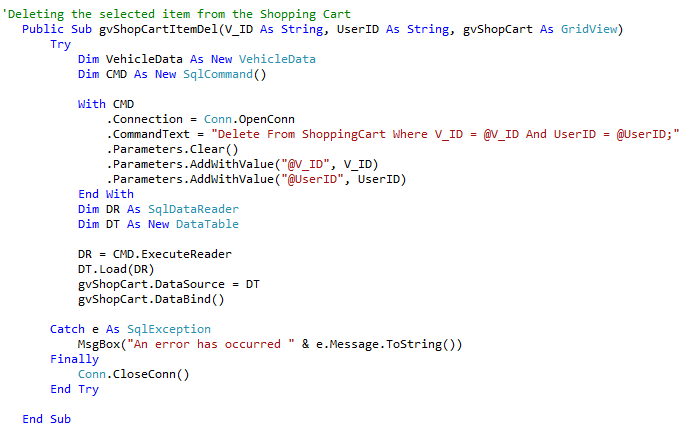
Part2 - ShoppingCart - gvShopCart\_RowDeleting:

This code will be used to select the shopping cart item and delete if from the cart accordingly. The "gvShopCart\_RowDeleting" event is shown in Fig.44.



### Fig.44 – **gvShopCart\_RowDeleting**

This event will fire upon the user pressing the "Delete" link button embedded in grid view, this event will access the "gvShopCartItemDel" available in the "ShoppingCartFunction" class, in order to get and delete the selected vehicle. The "gvShopCartItemDel" is depicted in Fig.45.



### Fig.45 – ShoppingCartFunction.**gvShopCartItemDel()**

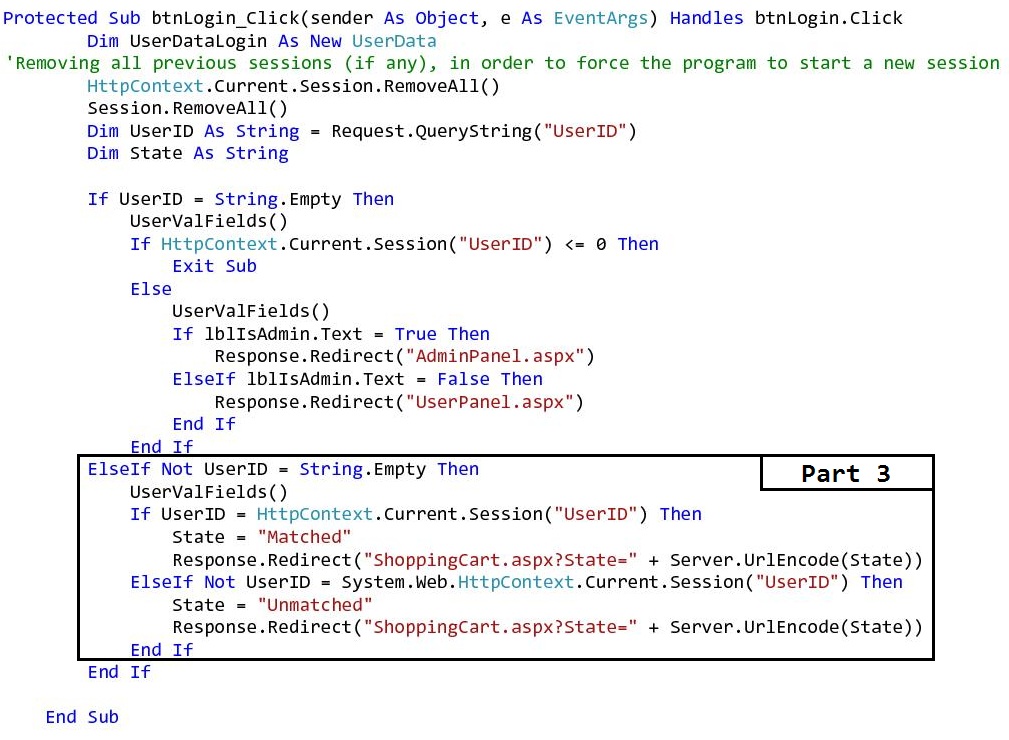
Part 3 - ShoppingCart - btnCheckout\_Click:

This will be the final step for purchasing the vehicle/s, however as already mentioned this event, requires more attention and additional security, due to its importance and functionality. When the user clicks on the checkout button the user will be redirected to the "Login" page and will have his "UserID" encoded as well (in order to be verified on the login page). Fig.46 illustrates the code underneath the "btnCheckout\_Click".



### Fig.46 – btnCehckout\_Click

Now we will describe the part 3 of the login process:

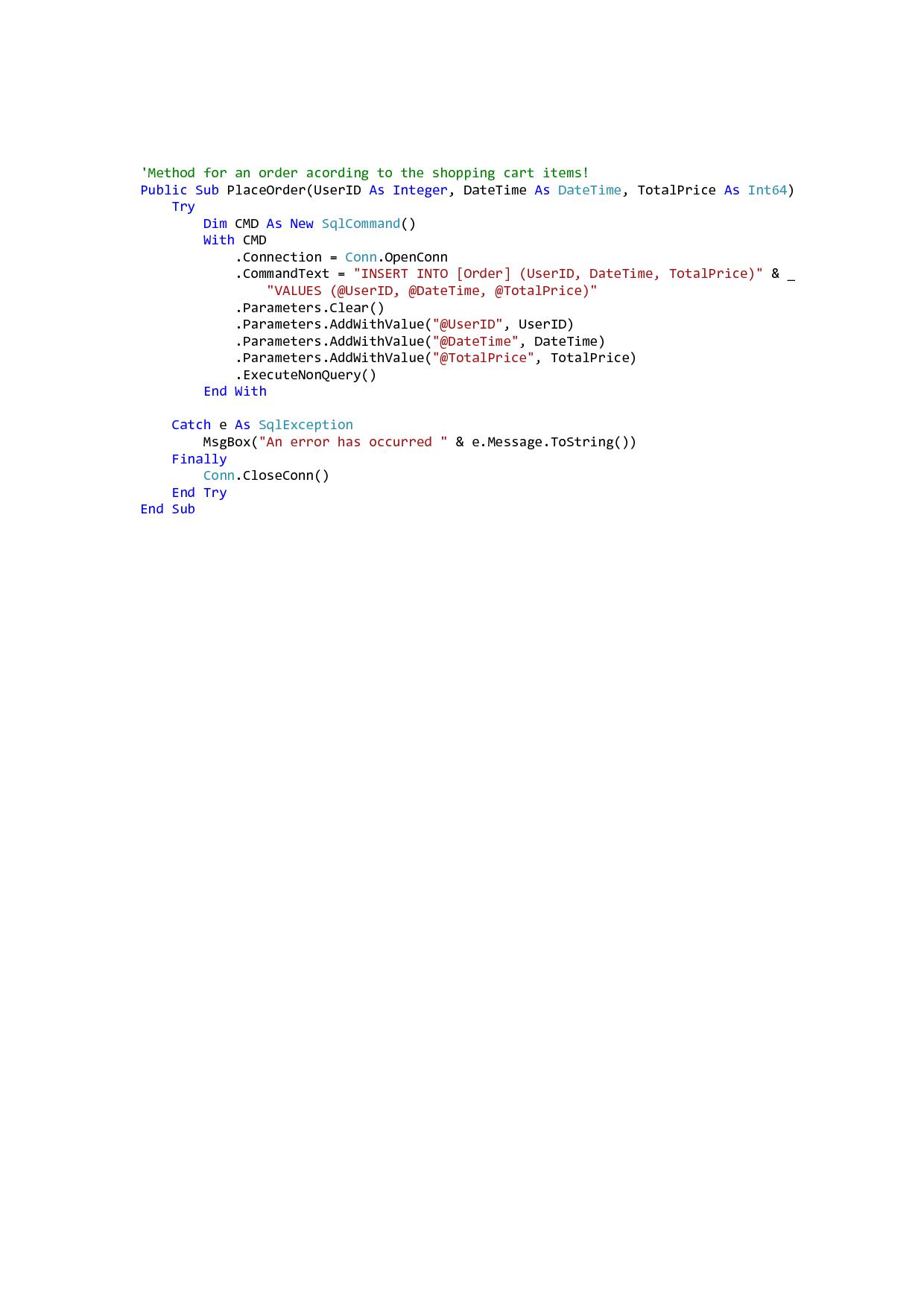
****

### Fig.47 – Part3 - Login

After the user has been redirected to the login page upon checkout button, the UserID will be decoded upon the login "page\_Load". the user will be prompted to re-log in the website in order to conduct a two-step login process. The decoded UserID will be then matched with the current login session, if matched then we will redirect the user to the checkout page and this time we will encode the "State = Matched" string. When the user will be redirected to the shopping cart page if the State is matched then the user may click on the checkout button once more so that he may finalise his purchase. otherwise the user login does not match with the UserID, State will be set as "Unmatched" and will be redirected to the login page again. If the state is "Unmatched" then it means that the user who added the vehicle/s to the shopping and the current logged in user, do not have the same login credential, therefore the order cannot be made.

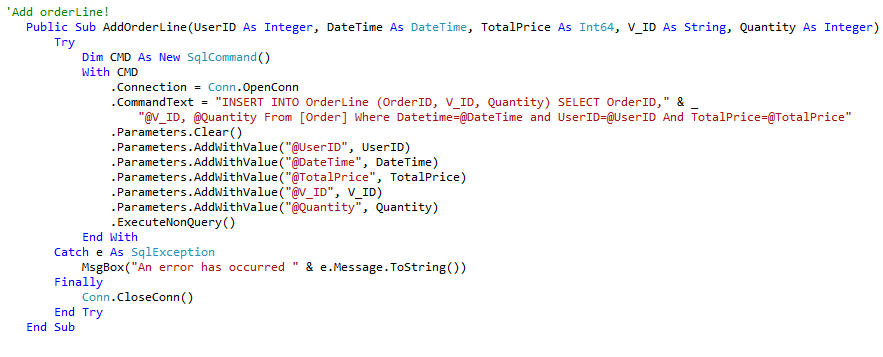
Part 4 - ShoppingCart - adding the order and orderline accordingly and marking the vehicle/s as sold:

When the user checks out and completes the purchase, our solution will create an order through the "PlaceOrder()" function, shown in Fig.48



### Fig.48 – OrderFunction.PlaceOrder()

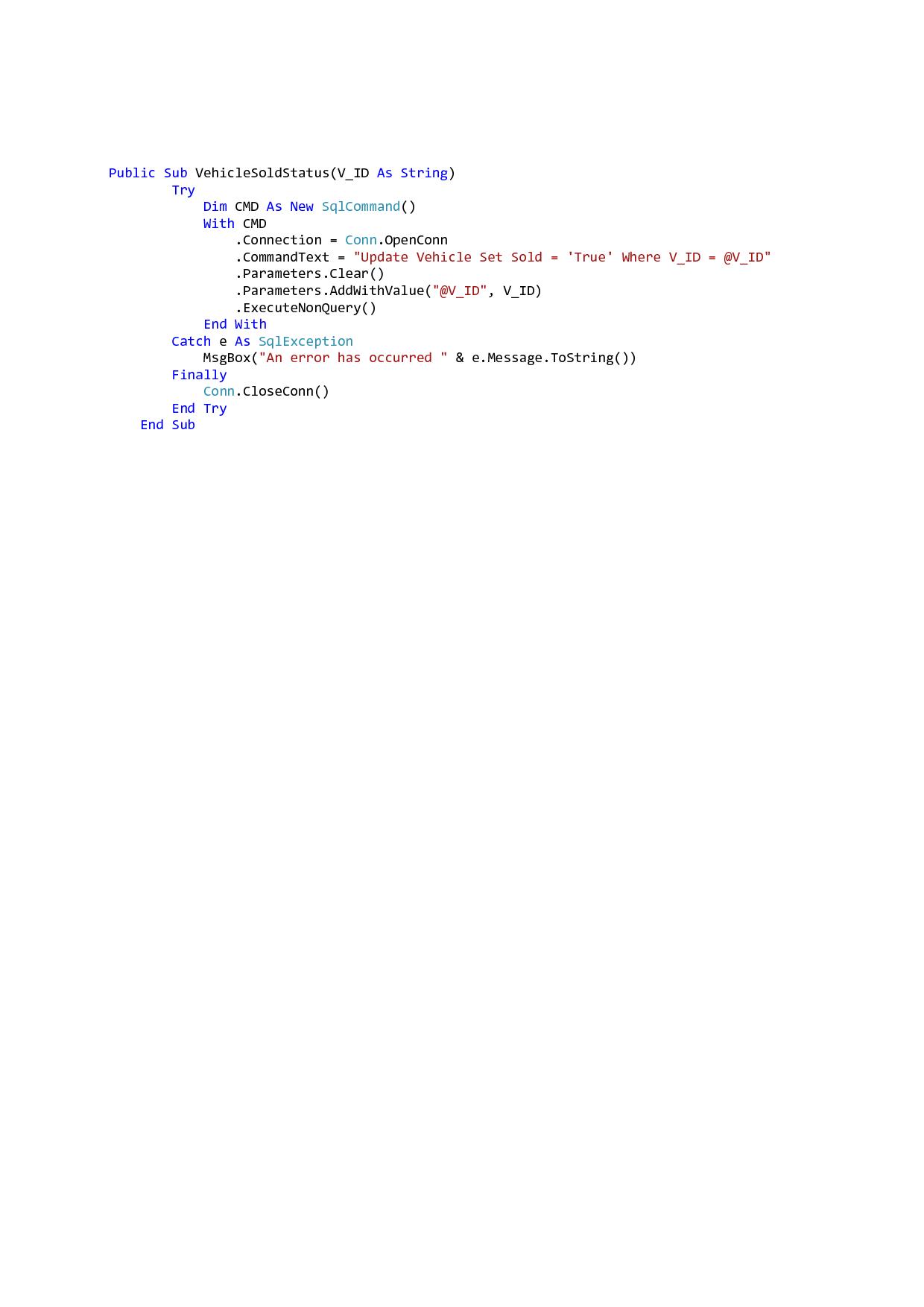
Then the appropriate orderlines for the current order will be created, through the "AddOrderLine()" function, shown in Fig.49.



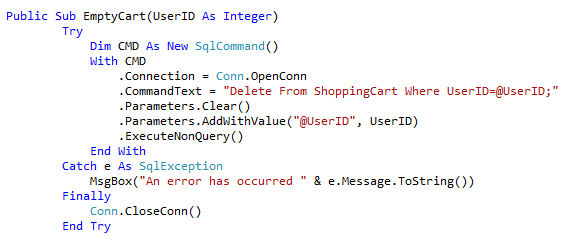
### Fig.49 – OrderFunction.AddOrderLine()

After creating the orderline, then the next step is to mark the vehicle as sold through the "VehicleSoldStatus()" function, and to empty the cart through the "EmptyCart()" function. The vehicle will be marked as sold, since our product is not like many others, therefore no two identical vehicles that have the same identical vehicle ID may exist.

These functions are both shown in Fig.50 and Fig.51 respectively.



### Fig.50 – OrderFunction.VehicleSoldStatus()



### Fig.51 – OrderFunction.EmptyCart()

# Database

In this section we will document and illustrate the database we designed for this coursework.

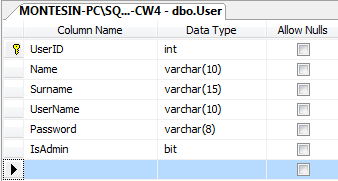
Our database name is "**BIS2212-CW4**", it consists of six tables as mentioned below:

* Category
* Order
* OrderLine
* ShoppingCart
* User
* Vehicle

A detailed explanation for each table can be found below.

## User Table:

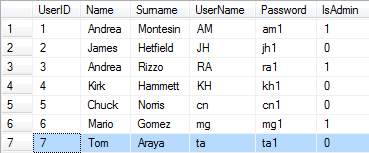
The User table will be used to hold user information, such as name, user name, password, etc. Shown below in Fig. 1.1 is the data structure of the User table and the data types respectively.



### **Fig. 1.1 – User Table**

The IsAdmin property in the User table will be used to identify whether a user is an administrator or not, the IsAdmin is of bit data type, which takes either 1 or 0 or True/False.

The figure below (Fig. 1.2) shows the User table populated with records.

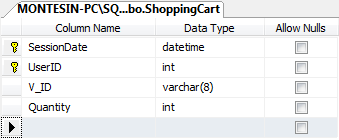


### **Fig. 1.2 – Populated OrderLine Table**

## ShoppingCart Table:

The shopping cart table will be used as a temporary table, therefore the user will add a vehicle to this table, the vehicle selected will stay in the ShoppingCart until the user decides to either buy the vehicle or removes it.

Shown below (in figure 1.2) one can find the table structure (columns and their data types respectively) of the Shopping Cart.



### **Fig. 1.2 – ShoppingCart Table**

Columns:

* SessionDate - The time and date the user added the vehicles to the cart.
* UserID - The (unique) user identification number for that respective user.
* V\_ID - A unique vehicle identification number.
* Quantity - Since the V\_ID is unique, the quantity of a vehicle added to the cart shall be always single (1).

Fig. 1.3 shows the ShoppingCart table populated with two vehicles.

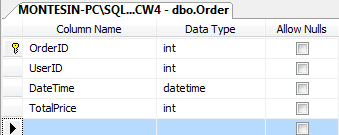


### **Fig. 1.3 – Populated ShoppingCart Table**

## Order Table:

As soon as the vehicles have been bought, an order will be created, thus the Order tables will be used to hold data about the orders processed.

The columns and their data types for the Order table are shown below (Fig. 1.4).



### **Fig. 1.4 – Order Table**

Columns:

* OrderID - A unique (order) identification number.
* UserID - The (unique) user identification number for that respective user.
* DateTime - The time and date the order was placed.
* TotalPrice - The total price for the particular order.

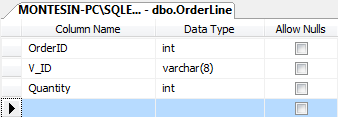
Fig. 1.5 shows the Order table populated with a single order.



### **Fig. 1.5 – Populated Order Table**

## OrderLine Table:

After the order is created the OrderLine Table will be populated with the data derived from the Order table. Therefore, depending if the user order two vehicles, then two OrderLine tuples will be created. The OrderLine is made up of the below mentioned columns (Fig. 1.6)



### **Fig. 1.6 – OrderLine Table**

Columns:

* OrderId - The unique Order number.
* V\_ID - A unique vehicle identification number.
* Quantity - Since the V\_ID is unique, the quantity of a vehicle added to the cart shall be always single (1).

A populated OrderLine Table will look as the one in Fig. 1.7.

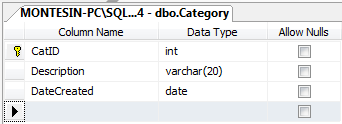


### **Fig. 1.7 – Populated OrderLine Table**

## Category Table:

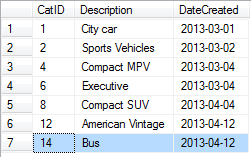
The category table will be used to categorise the different types of vehicle, for example a family car is different from a sport vehicle, so the category table will be used as an intermediary in order to allow identification between the different types of vehicles.

Illustrated in **Fig.1.7** one can find the Category table structure, including the column names and their data types respectively.



### **Fig. 1.7 – Category Table Structure**

Once populated with data the category table will look as in the below figure **Fig.1.8**

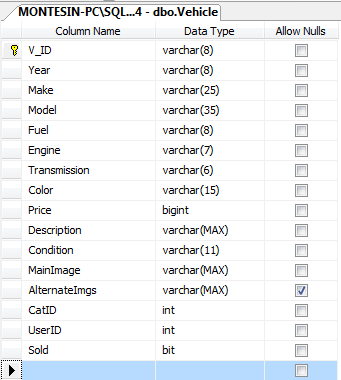
****

### **Fig. 1.8 – Populated Category Table**

## Vehicle Table:

The Vehicle table will hold all the relevant information about the vehicles listed in the website. This table is made up of various vehicle properties such as fuel type, transmission, price, condition, etc. These properties will be used to define the vehicle.

The Vehicle table data structure is illustrated below in Fig. 1.9.

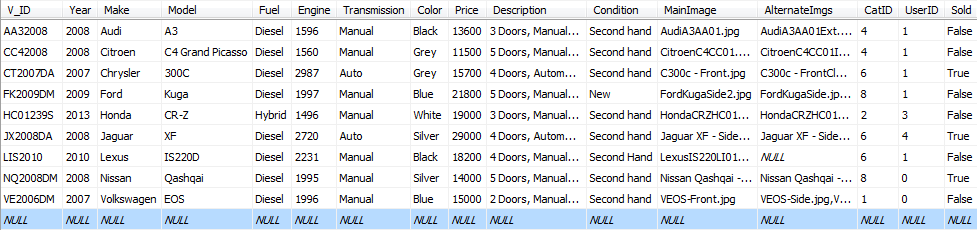


### **Fig. 1.9 – Vehicle Table Structure**

Columns:

* MainImage - Will be used to hold the path for the (user chosen) main image
* AlternateImgs - If the user decides to add more images other than the main image, this property will be used to hold the path for each alternate image.
* Sold - Upon vehicle creation the Sold property will be declared "False" since the vehicle has not been sold yet. If the Vehicle is bought than the Sold property will be updated to "True", so as to avoid relisting the vehicle when as we said before the quantity for each vehicle is 1.

The Vehicle table once populated will look as in Fig. 2.0



### **Fig. 2.0 – Populated Vehicle Table**

# References

# Appendix