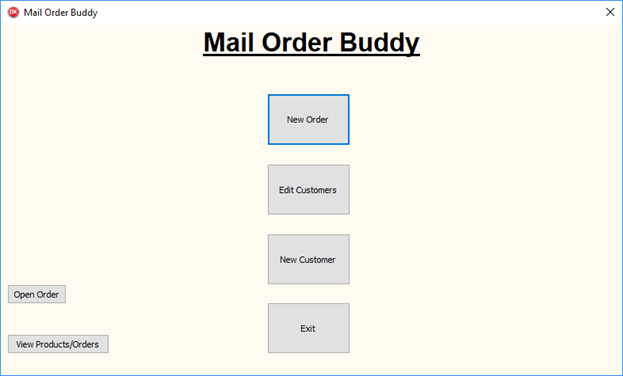
Documented Design

The structure of the system is a graphical user interface which the user can interact with to create orders, view customer details and add new customers. It has many forms that each perform a required task. For example, the ‘Main’ screen shows very clearly how the user can get to each of the forms and carry out their required job. The code is fully functional and meets all of the objectives that I set out in the project analysis part of the document, it does this very quickly and efficiently.



EditCustomerBtn: Allows the user to edit a customer to edit a pre-existing customer.

NewOrderBtn: takes the user to the new order page via the customer search form.

NewCustomerBtn: Allows the user to add a new customer and enter all their details.

OpenOrderBtn: takes the user to the Open Orders Form.

ExitBtn: Exits the program.

ViewOrdersBtn: Takes the user to a from where all the tables from the database are displayed, from this they can view orders and see which products they need to order.

The code performs all the operations that the user currently does by hand, this I have been able to implement and turn in to a graphical user interface, with the algorithms that I have wrote running behind each button click. The software works by soring the: Customers, the orders that the customers place, the order details and lastly the products that the company stock inside a database, I have chosen to use a Microsoft Access database file as the way of storing the data simply because they are easier to work, from an aspect of writing code to interact with the file and making them perform in a way in which you want, this is because you can use SQL queries, compared with a CSV file or other table based file, where code may be slightly harder to write and interact with the certain data that the user requires; all of this data can be added by the user through the program. The program is very easy to navigate with all the forms having buttons that clearly describe and tell the user what their purpose is, also every form has a ‘home’ button that will take the user straight back to the ‘Main’ screen, this clears what the user is currently doing at that point in time and will start the form from a fresh the next time that it is opened. Thus, meaning if you click the wrong button you don’t have to restart the program, or go through creating an order, just to get back to the ‘Main’ screen, this makes for better user interaction and feel to the program, knowing that you can move about it freely and go from screen to screen without being stuck in a loop to get back to the ‘Main’ screen.

How the Forms are linked

This diagram shows how all the different forms are linked and how the user will be able to navigate around the program and carry out the procedure that they currently do in a paper-based system.

Open Order Button

Open Orders Form

Edit Customer Button

Edit Customer Button

Create Order Button

Main Menu

New Order Button

New Order Customer Search Form

Edit Customer Search Form

New Customer Details Form

New Order Details Form

Edit Customer Details Form

View Orders and Products Form

View Products/Orders Button

New Customer Button

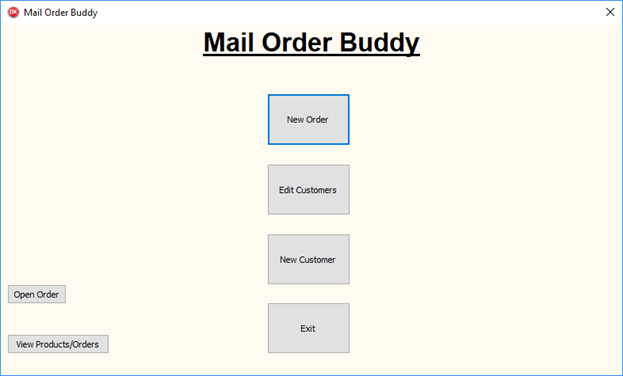
Key:

Sample text: Button Label

What each form does:

I this bit I will explain how each form works, with the task that it carries out and how it interacts with the other forms.

Main Menu Form: this form consists of some very simple buttons which all link to other form that carry out the paper-based tasks that the user is currently doing. It is a very easy to navigate form and does not need much explaining to work your way around it.



NewOrderBtn: takes the user to the new order page via the customer search form.

EditCustomerBtn: Allows the user to edit a customer to edit a pre-existing customer.

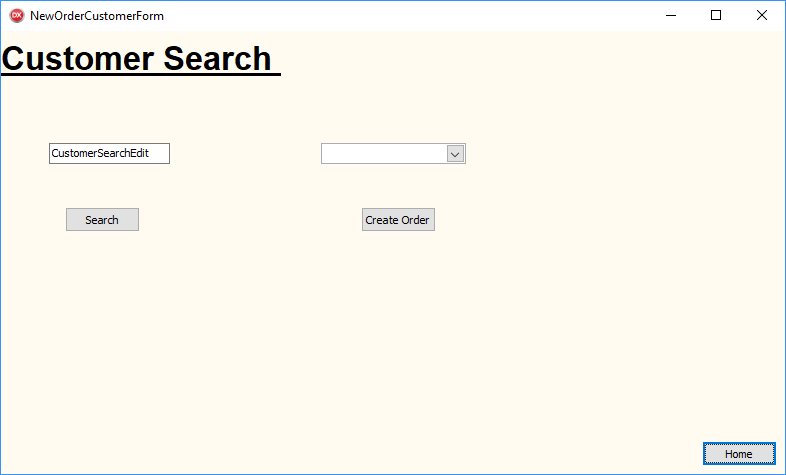
NewCustomerBtn: Allows the user to add a new customer and enter all their details.

OpenOrderBtn: takes the user to the Open Orders Form.

ViewOrdersBtn: Takes the user to a from where all the tables from the database are displayed, from this they can view orders and see which products they need to order.

ExitBtn: Exits the program.

New Order Customer Search Form: this form consists of an ‘Edit Box’ where the user can input their required search term, this could be the first few words of the ‘CompanyName’ or it could be the whole thing. This value is then taken and injected in to a SQL query, that is then set as the ‘Command Text’ for the ‘Customers’ table in the database, this subsequently updates the ‘CustomersSet’ in the program. From this you can use the ‘DBComboLookUp’ to select the ‘CompanyName’ that you were looking for. Once the user has found the required ‘CompanyName’ they can then click the ‘Create Order’ button and it will take them to the ‘New Order Details Form’ where they will create the order for the customer.



DBComboLookUp that displays the selected records form the ‘Customers’ table.

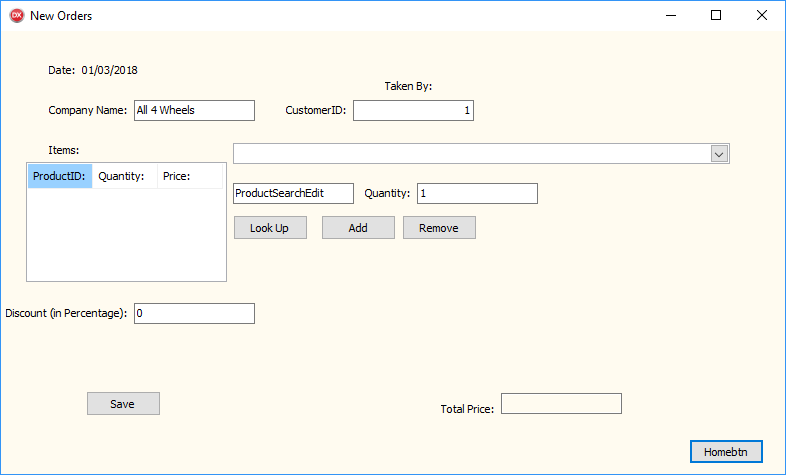
Edit box where the user can enter their search term for the customers ‘ContactName’.

Create Order button that takes the user to the New Order Details form.

Home button that takes the user back to the main menu.

Search button that runs the SQL query based on the edit box.

New Order Details Form: using this form the user can create new orders for their required customer. They do this by either searching for a product, using the products name, or by selecting the product form the drop down ‘DBComboLookUp’. Then when the user clicks add they will see the selected product be added to the ‘Items Grid’ with the quantity that they entered and the ‘Unit Price’. Th form also calculates a total price in which it adds up all the prices of the products based on the quantity enter by the user, this is displayed in an ‘Edit Box’ which is read only, meaning the user can’t edit the prices of products, this can only be done with access to the database ‘Products’ table. From this the user can click the ‘Save’ button, this will save the order and put the required entries in to the right columns, in the right tables in the data base. Whilst it does this it creates a unique ‘OrderID’ which can only be used once and only for one order. After this has been seen all the way to the end it displays the final price which the user can then tell the customer. After the order has been saved a ‘Print’ button will appear, this button creates a html file which can be displayed though any browser, as they can interpret html and turn it in to a readable page. This readable page consists of the customer’s: ‘CompanyName’, ‘Company Address’, ‘Company Phone Number’, the date that the order was taken, and finally the products that the customer has ordered, with the ‘Product Description’, quantity and price, also in the Order File is the total price and the VAT values, VAT being 20%.



This is where the date and customer details are displayed, so the user knows who they are filling it out for and can check that they have selected the right customer.

This is where the user can search for a ‘ProductName’ and it will display the relevant based on the results from the SQL query, 'SELECT \* FROM Products WHERE ProductName Like "%' +ProductSearchEdit.text + '%"', they can then be added to the grid via the ‘Add’ button and the quantity entered, but auto set to 1

This is the grid where the chosen products are displayed, with the quantity and price. And below is the discount edit box, where the user can enter a discount if a code is redeemed or a special offer.

This is where the save and print button is, the print button is invisible until the save has been carried out in full.

This is where the total price is displayed and is automatically updated when a product is either added or removed.

Edit Customer Details Search Form: This form is quite similar to the ‘New Order Customer Search’ where the user can search the data base for the required customer and choosing from a drop down menu they can select the customer and click the ‘Edit Customer Button’ this then takes them to the customer edit page where the can see all the customers details, these are all displayed in ‘Edit Box’s. The search of the data based is done with a simple SQL query, where it selects the data that it needs by looking for ‘ContactNames’ that are like the data entered by the user.

begin

with DMain.DataMain.CustomerSet do

begin

Close;

//query must be closed before changing parameter value

CommandText := 'SELECT \* FROM Customers WHERE ContactName Like "%'+CustomerSearchEdit.text+'%"';

ShowMessage(CommandText);

open; //open (run) the query

ShowMessage('Query Done!');

end;

end;

Edit Customer Form: This form take the selected customers detail and then these are all displayed in ‘Edit Box’s that the user can edit by typing in the new data that the customer have provided, once the user has entered all of the required information they can click the ‘Update’ button, which runs a procedure I have created to update and insert all the new data in to the correct cells in the data base. After this the user can go back to the ‘Main Menu’ to carry out a new task. This is done by finding the field value in the database and adding rewriting the data which is currently stored there, this is not an SQL query, with update, it is using a ‘Edit’ and ‘Post’, which opens the database for editing and then closes it after all editing has taken place.

begin

with DMain.DataMain.CustomerSet do

begin

Edit;

FieldValues['CompanyName'] := DBCompanyNameEdit.Text;

FieldValues['ContactName'] := DBCustomerNameEdit.Text;

FieldValues['Address'] := DBCustomerAddressLine1Edit.Text;

FieldValues['City'] := DBCustomerAddressLine2Edit.Text;

FieldValues['Postcode'] := DBCustomerPostcodeEdit.Text;

FieldValues['County'] := DBCustomerAddressLine3Edit.Text;

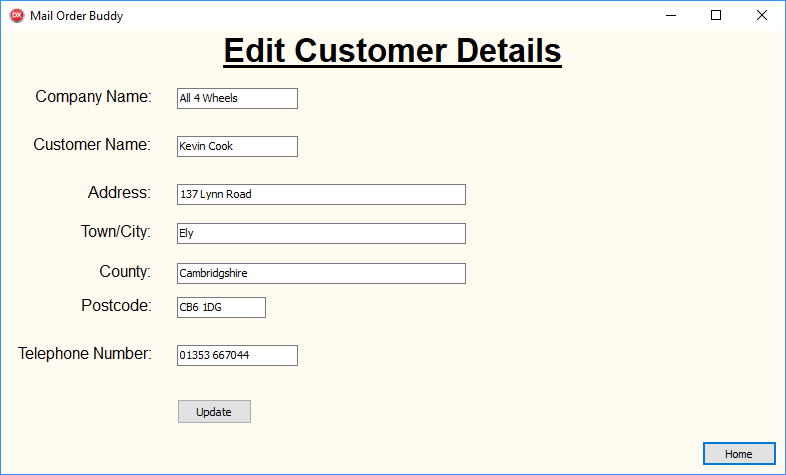
FieldValues['Phone'] := DBCustomerTelephoneEdit.Text;

Post; // save the new record

end;

ShowMessage('Update Complete');

end;



Labels that tell the user what data each edit box should contain.

Edit boxes that display the current details of the selected customer, which the user can click on and retype.

Update button that the user can click once they have finished update all the data that is needed, saves the new data in the database using the code above.

New Customer Details Form: The ‘New Customer Details Form’ is the form where the user would enter a new customer and all their details. It has a very similar layout to the ‘Edit Customer Details Form’ however the main difference is that it does not have any customer details preloaded for editing, it just has empty ‘Edit Boxes’ which the user types in the correct data, based on the ‘label’ next to the box. When all of the data has been added then the ‘Update’ button can be clicked, and the program will run a procedure that is similar to the code above; the only changes being ‘Edit;’ is ‘Append;’ and the table has to be opened before any adding is done, this is with the code ‘Open;’ Also it clear the ‘Edit Boxes’ after the new customer has been saved, this allows the user to easily add a new customer straight away.

begin;

with DMain.DataMain.CustomerSet do

begin

Open;

Append;

FieldValues['CompanyName'] := NewCompanyEdit.Text;

FieldValues['ContactName'] := NewCustomerEdit.Text;

FieldValues['Address'] := NewAddress1Edit.Text;

FieldValues['City'] := NewAddress2Edit.Text;

FieldValues['Postcode'] := NewPostcodeEdit.Text;

FieldValues['County'] := NewAddress3Edit.Text;

FieldValues['Phone'] := NewPhoneNumberEdit.Text;

Post;

end;

NewCompanyEdit.Clear;

NewCustomerEdit.Clear;

NewAddress1Edit.Clear;

NewAddress2Edit.Clear;

NewAddress3Edit.Clear;

NewPostcodeEdit.Clear;

NewPhoneNumberEdit.Clear;

ShowMessage('New Customer Created');

end;

Open Orders Form: This is a very simple form with and ‘Edit Box’ where the user types in the order number that they wish to view, this being the ‘OrderID’, and then clicking the ‘Open Order’ button. The system then will run a small piece of code which takes the number that has been typed in and finds the corresponding file.

begin

FileInteger:='G:\Computer Science Project\Win32\Debug\Order '+(OpenOrdersEdit.Text)+'.html';

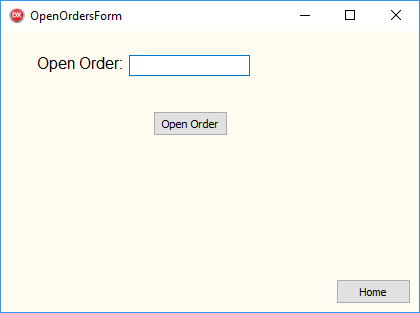
ShellExecute(Handle, 'open',

'C:\Program Files (x86)\Internet Explorer\iexplore.exe',

PWideChar(FileInteger), nil, SW\_SHOWNORMAL);

end;

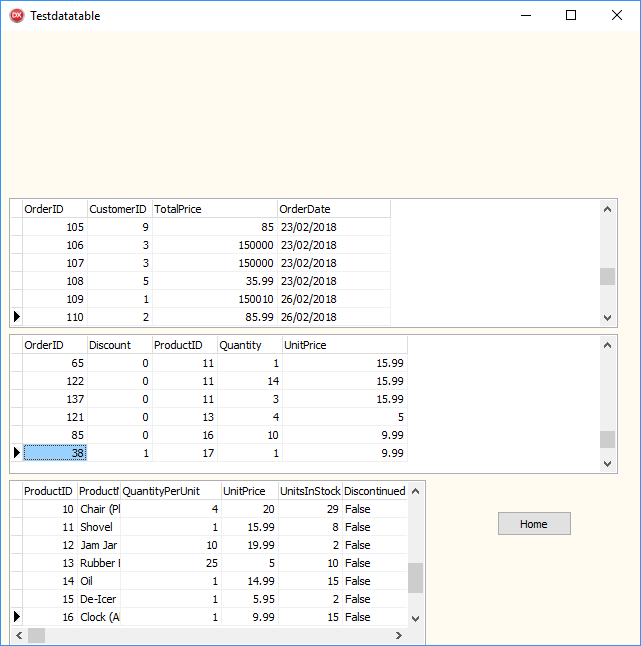
This code takes in the text and concatenates it in to the string that has been set up in the code, based on where the order files are being saved, assigns the whole string to a variable called ‘FileInteger’. Which is then inserted in to the ‘ShellExecute’ procedure; this procedure establishes what it need to carry out by the variables that have been put in to it, from this one we can see that it is being told to open the file that it will be given, ‘open’, using the program that is given as a path address, then after this required file path address is given and from this the system can open it. This procedure is also carried out in the ‘New Order Form’ after the new file has been created, this makes the process of printing quicker has it is automatically opened, and the user doesn’t have to go and find the file to print it.



Edit box where you type in the required order number and then click ‘Open Order’ and it automatically finds the file and opens it in ‘Internet Explorer’.

Home button that takes the user back to the main menu.

View Products/Orders Form: This from show the user the tables from the database, from this they can get order numbers of old orders and they can also view the ‘UnitsInStock’ of each product; meaning they can know which products they need to order in, as they are low on or out of stock. It is set up with ‘DBGrid’s that are linked to each data source that is linked to a table in the database. It is a very simple form, but it helps as then the user does not have to go directly in to the database and potentially risk changing some data that caused cause errors and they didn’t mean to change.



‘Orders’ table that displays the ‘OrderID’ and ‘CustomerID’ along with the price and order date.

‘OrderDetails’ table that displays the ‘OrderID’ and ‘ProductID’ as a composite key, along with the ‘UnitPrice’, quantity and discount.

‘Products’ table that shows all the products and their name’s, along with the ‘UnitPrice’, ‘UnitsInStock’ and their ‘QuantityPerUnit’.

Entity Relationship Diagram

This is the entity relationship diagram for the system that I have created. It shows how each of the tables in the database interact with the other tables that the program uses; how they are related with the ability to have certain values repeating throughout the tables whilst not causing any problems with writing errors or having two conflicting primary keys. It also helps show the database normalisation and how the tables store all different pieces, yet related, data about the whole system.

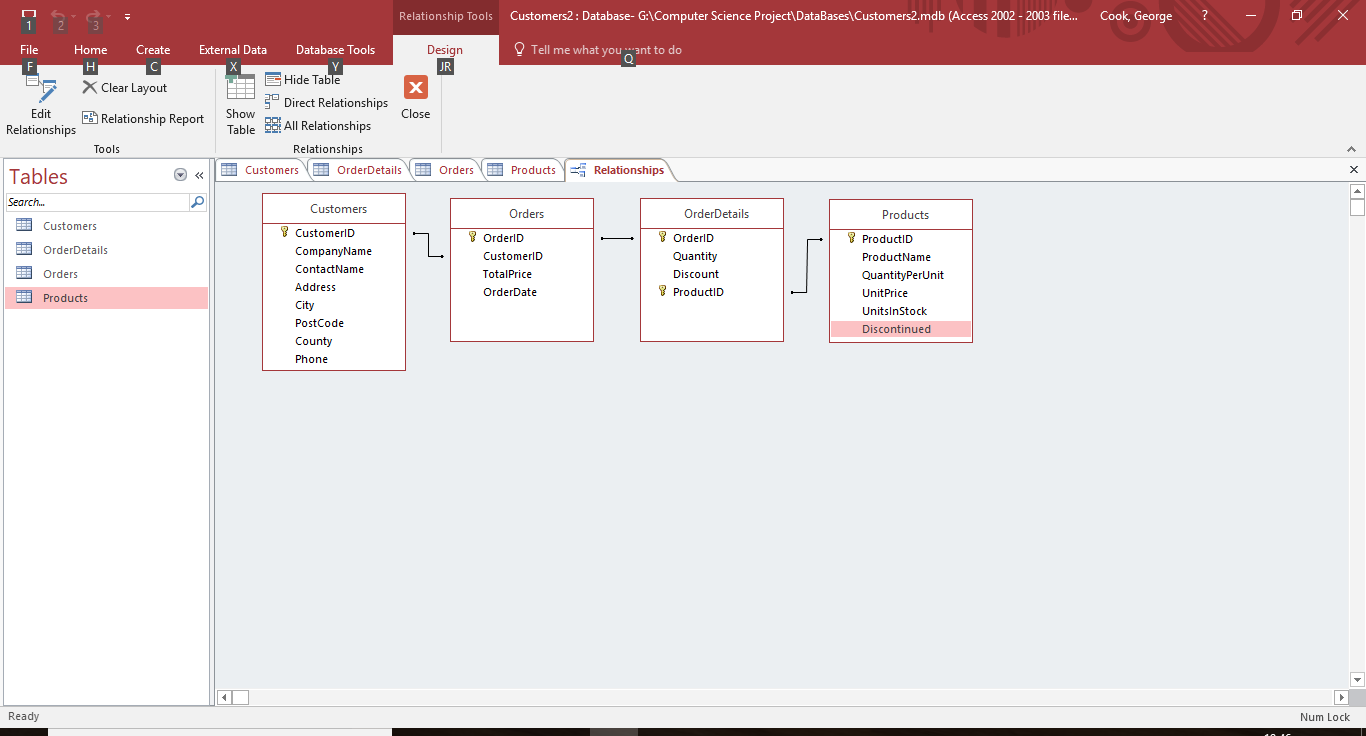
OrderDetails

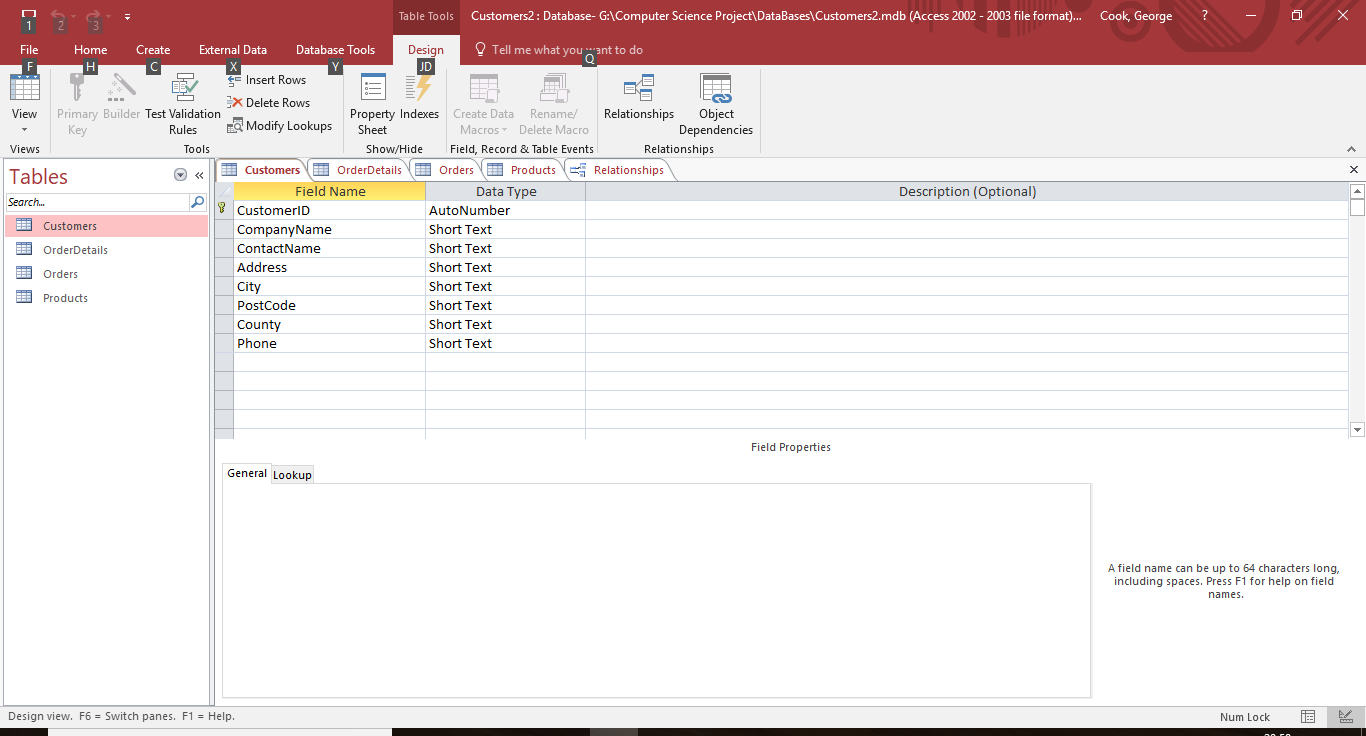
Products

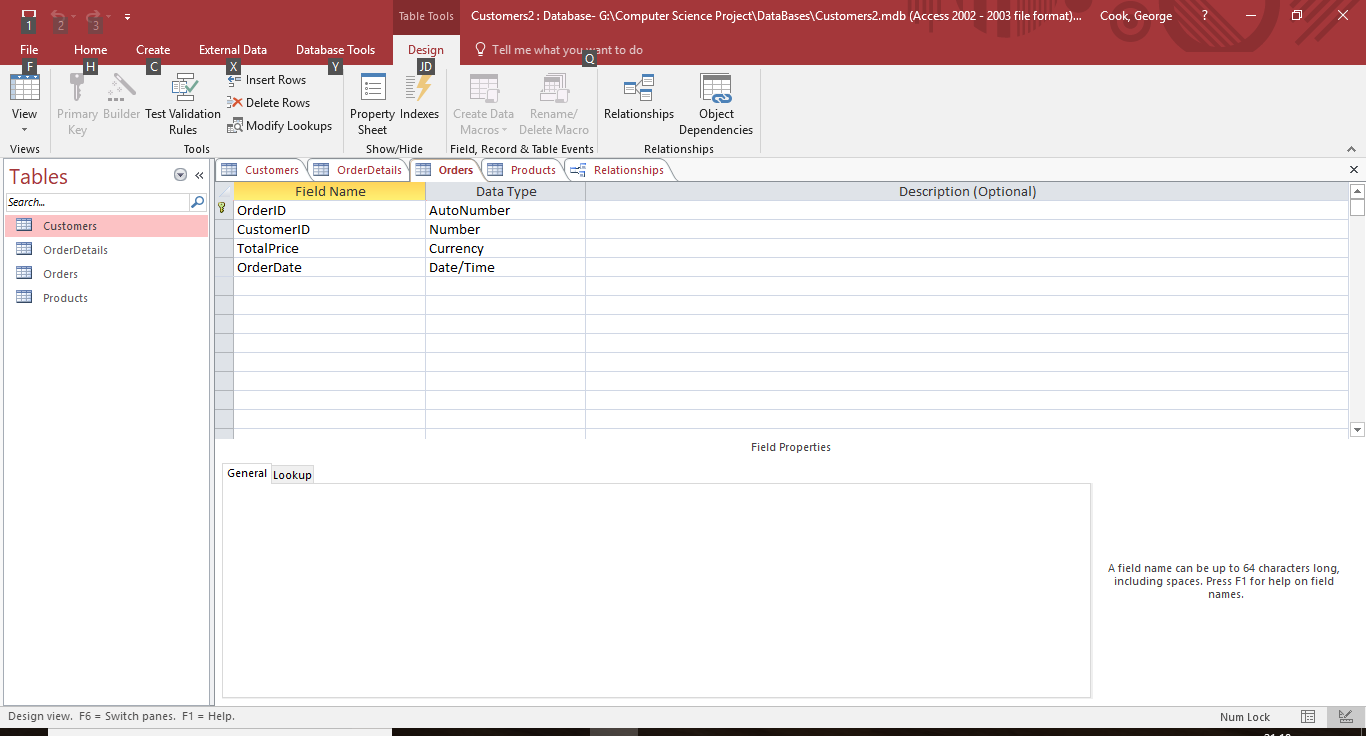
Orders

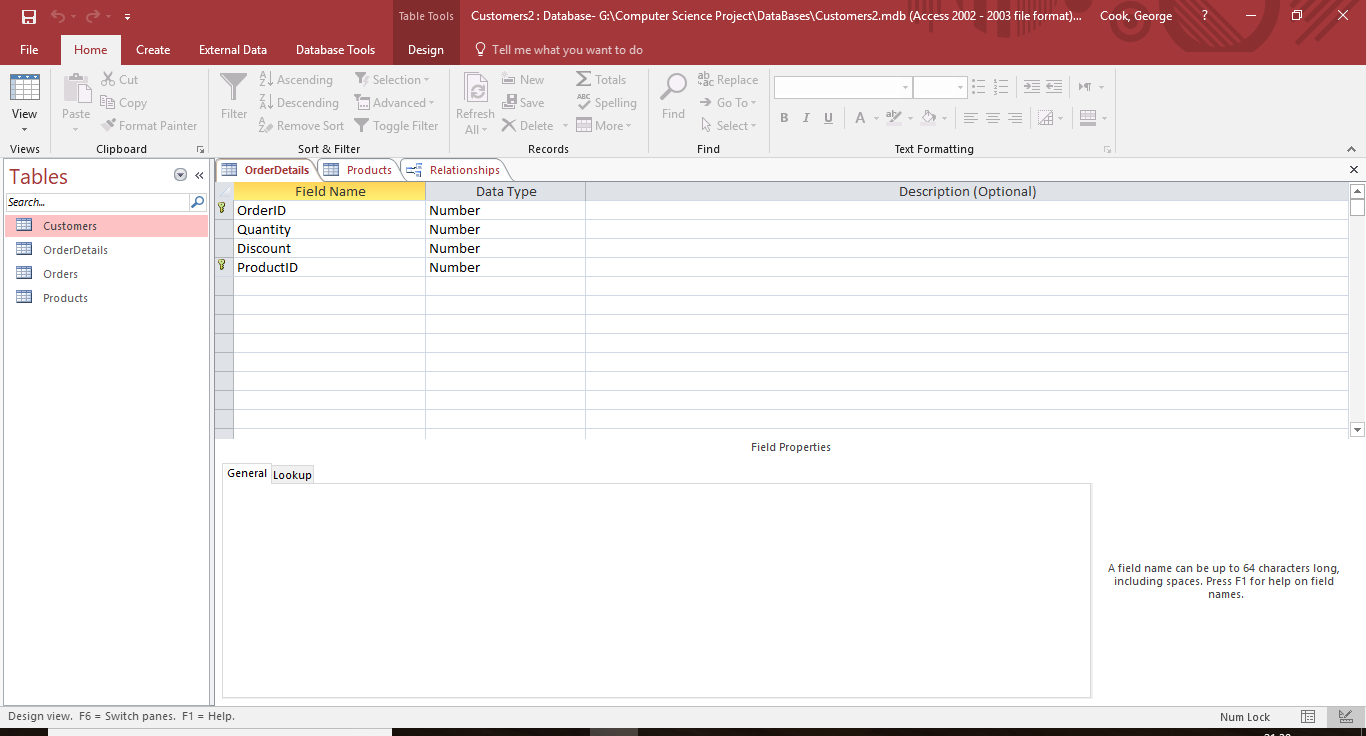
Customers

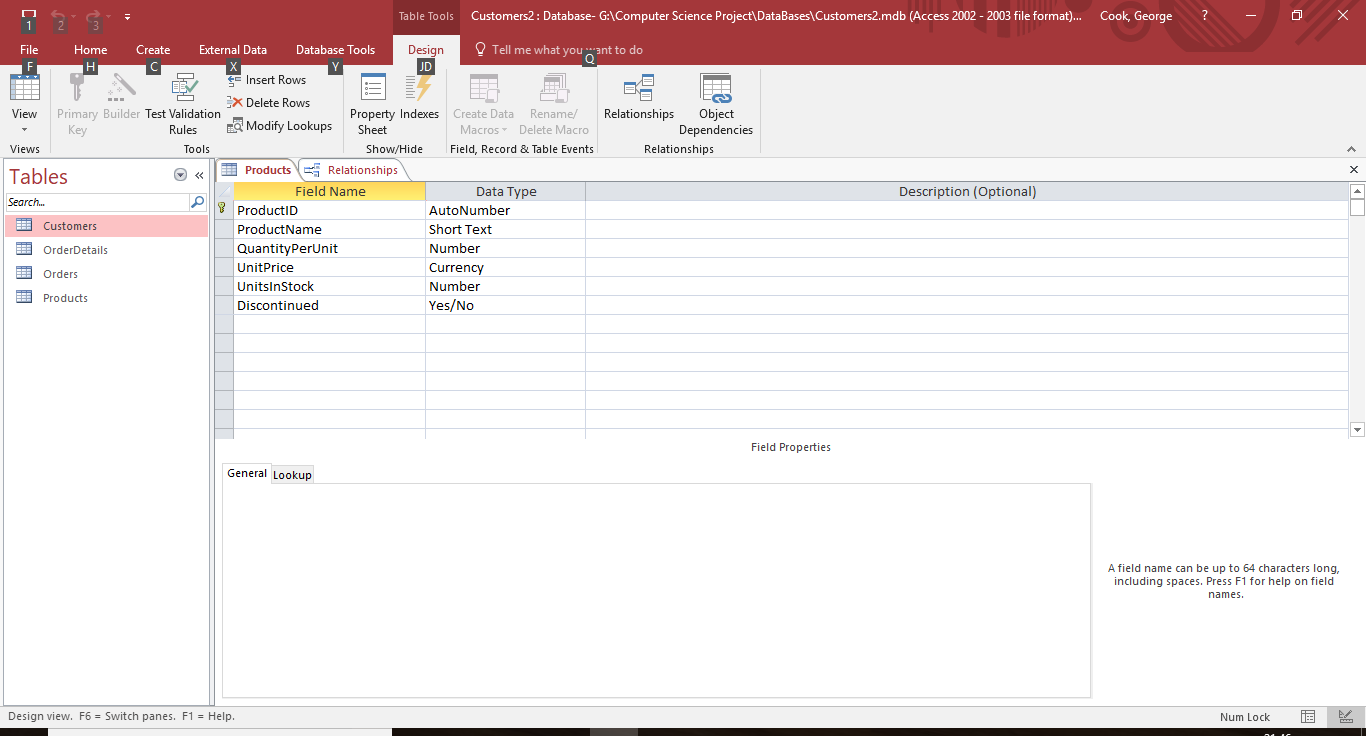
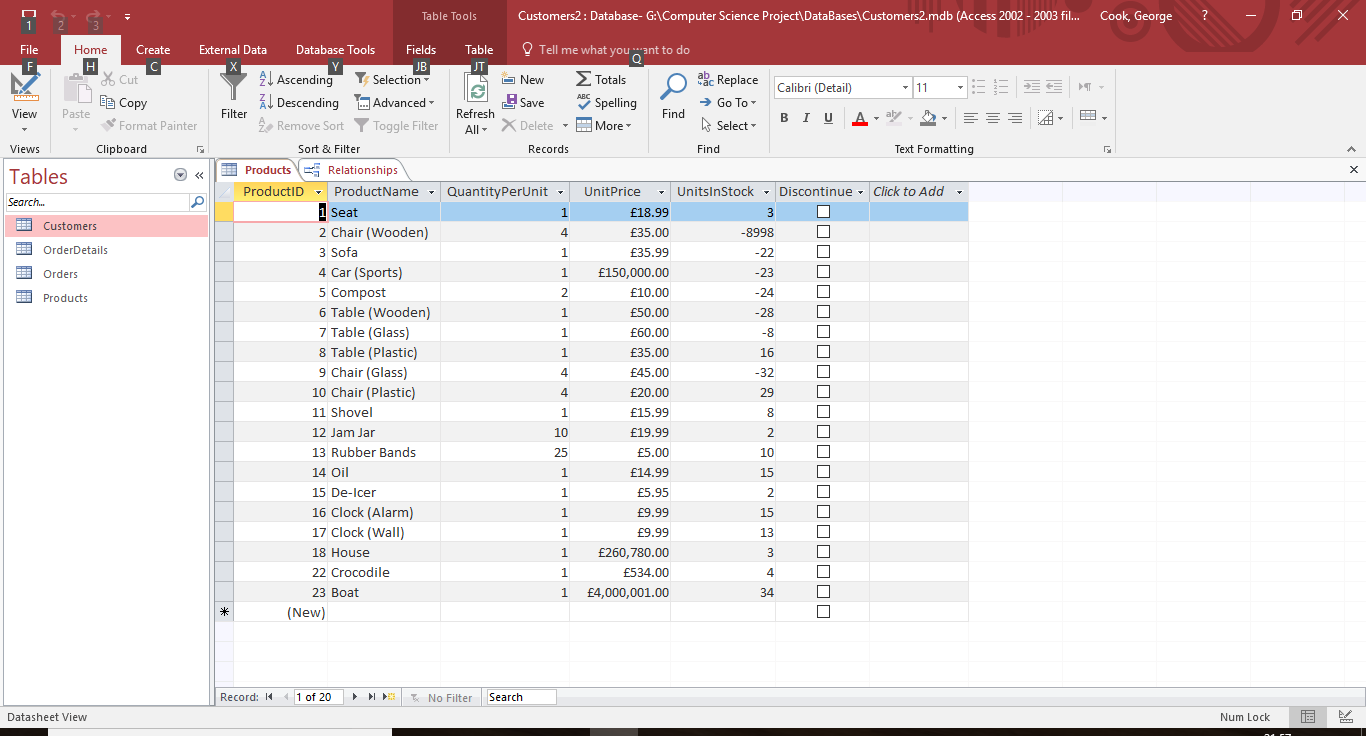
Database Relationships

This diagram shows how the database tables are all linked, it also shows the primary keys for each of the table, and one composite key for the ‘OrderDetails’ table. It also shows that the database is in third normal form, with no unnecessary repeating data, and all the entities depend on the key, the whole key and nothing but the key, meaning that they are all related to the key and there is not data that has not got anything to do with the table, in the table.

Customers table: this table has been designed and created to hold all the relevant information about the customers, this includes: Company Name, Contact Name, Address, City, Postcode, County, Phone, and a unique Auto Number primary key. From this table we can see all the data types that are used. The short text has been used because long strings of text do not need to be stored in this table, thus saving space and allowing for smaller and faster movement of data between the system and the database.

Orders table: this is the table that links all the orders placed with the customers that have placed the order. The linking factors is the ‘CustomerID’ which is not part of a composite key in this table as it doesn’t need to be and the ‘OrderID’ does the job as it will only appear once in the table; unlike in the ‘OrderDetails’ table. From this table we can see the different types of the data, with the ‘Date/Time’ being formatted to ‘DD/MM/YYYY’ and gets its value straight from the program’s ‘Now’ function which returns the current date of the computer. The ‘OrderID’ is set as the primary key for this table as it is never repeated, so therefore can be used as the primary key without causing any issues inside the database and when the database is queried.

OrderDetails table: this table holds all the information about the orders placed, which products the order includes, this is done with the use of the ‘ProductID’, which link that links the two tables together, the quantity of a product and how much discount the order has, this all coming under ‘OrderID’s that make up the composite key with the ‘ProductID’s. The reason that I am using a composite key is to stop the chance of errors and misinterpretations, when the table is queried, as the ‘OrderID’ appears many times, unlike in the ‘Orders’ table, it appears on each line of a different ‘ProductID’, that is in the same order. From the table we can see the different data types that have been used, you will notice the ‘OrderID’ and ‘ProductID’ although part of a composite key, they themselves are not ‘AutoNumber’s, the reason for this being that they get their value from the other tables when the orders are being created.

Products table: this table holds all of the information about the different products that can be ordered through the system, such as: ‘ProductID’, ‘ProductName’, ‘QuantityPerUnit’; which is how many of a product you get in a “packet”, ‘UnitPrice’, ‘UnitsInStock’ and ‘Discontinued’; which is just a Boolean value that tells the user whether the product is still available or is no longer made or stocked. The primary key in this table is the ‘ProductID’ it is an ‘AutoNumber’ which is generated when a user enters the details for a new product in to the database, there is only a need for the one primary key and not a composite key as the ‘ProductID’ only has the potential to appear once; unlike in the ‘OrderDetails’ table. From the table we can see the different types of data used, the ‘ProductName’ uses a ‘ShortText’ data type as a long description of the product is not need it is just the name and maybe a key feature in brackets. An example of this can be seen with the chairs.

Key variables and arrays

|  |  |  |
| --- | --- | --- |
| Variable Name | Access | Array description |
| OrderArray | Private (to the New Order Form) | This is used to store the quantity and the product ids of an order when it is being saved.  type  TOrderRecord = Record  Product: String;  Quantity: Integer;  End;  OrderArray: array of TOrderRecord;  This is the code of the TOrderRecord that holds three field values, them being used for the ‘ProductID’ and the quantity needed of each product. And in the second part we can see ‘OrderArray’ being initialized as an array of TOrderRecord, creating a 3-dimensional array. |
| htmlfile | Private (to the New Order Form) | This is a ‘text’ variable that stores and point to the address of the order file that is being created after the user clicks the save button. It allows the program to do ‘WriteLn’s straight in to the file that then creates a html file to be opened later. |
| ArrayCount | Private (to the New Order Form) | ArrayCount is a integer variable that stores how many spaces are filled in the ‘OrderArray’, this allows for the program to run through the array to see if it is full, it is also used in the creation of the html file as it is sorted through and lets the system to know where it needs to stop so a ‘null’ error is not thrown up and the html file crashes. |
| Column, Row | Private (to the New Order Form) | Column and row are integer variables that point to the required position in the ‘ItemsGrid’ this is used when things are being added to the grid and then being removed, it allows the system to clear it as well and stops the confusion of a procedure thinking a different column or row is selected. |
| PriceT | Private (to the New Order Form) | Is a currency variable with its main use being to update and automatically when an item is added or taken away. It is a global variable as it is needed by many procedures and saves time with creating a new variable in each procedure and possibly getting the wrong number coming out if they don’t all hold the same value. |

I don’t have any other key global variables as the rests of my variables are all local to the procedures and can’t be accessed by other parts of the system.

File Organisation

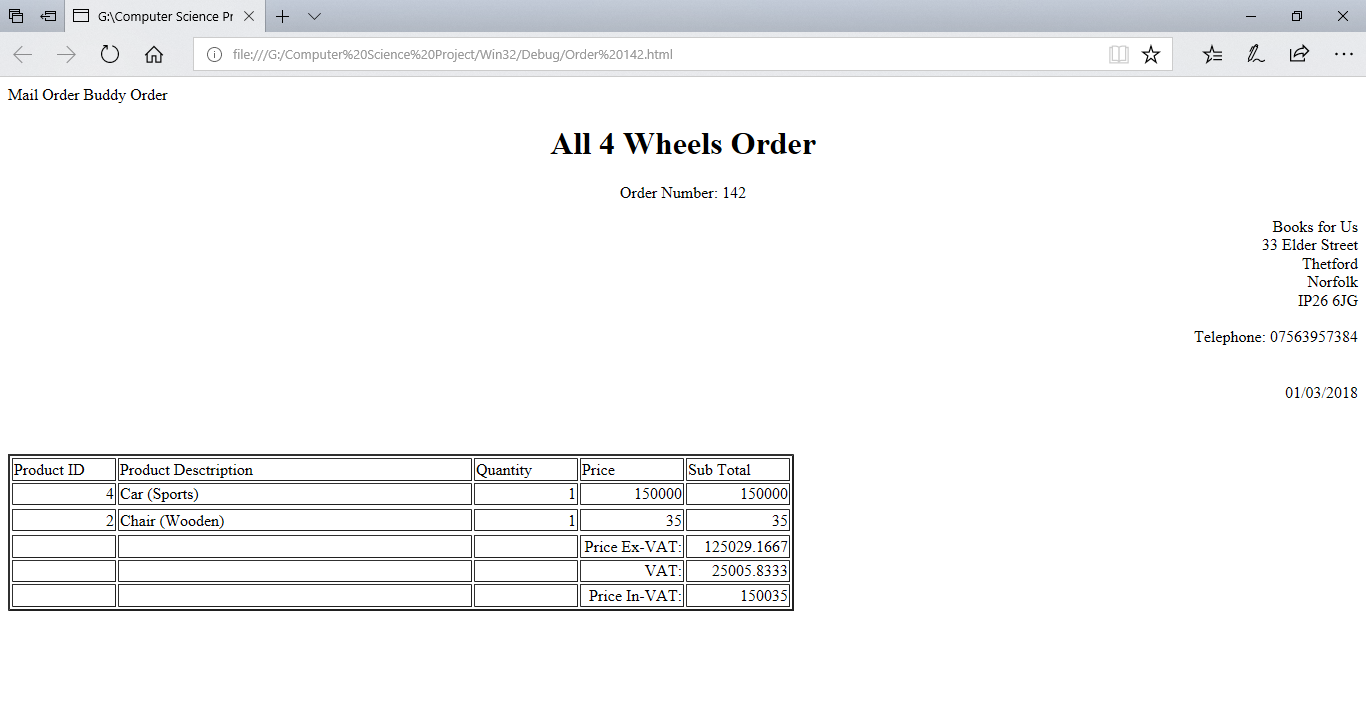
The system creates and outputs many files, they are all stored in the next to the executable file, later I might edit the program so that they are stored in other folder, which will allow for better organisation and easier navigation of the program folders. However, the files can be all opened and saved from inside of the system, so there is really no reason for the user to have to go through the files and interact with them directly. The files are save as ‘.html’ the reason I have done this is because most, if not all, computers that run windows have a browser, Internet Explorer, that interprets and displays html files. I chose to do this over a Word file as it does not limit the usability of the system and means that the target computer does not have to have word installed, which would be an extra cost for the user.

The file is split in to two parts, the customers details and then below that the information about the products that have been order, with all the prices in the of each product, the total price and VAT price, with the total price excluding VAT.

Later I will add the card details in to the system and put these on the end of the order form, however to make this work I will have to encrypt the database to keep the details for security reasons.

Algorithms

One of the main big algorithms that the system uses is the procedure that creates the html file to be saved and opened. It takes the products that are in the ‘OrderArray’ and through writing through a html file, it fills in the spaces in the table through added new lines and cycling through the ‘OrderArray’ whilst using the ‘ArrayCount’ to know when it must stop, thus meaning there are no unnecessary empty lines in the table or errors that could occur with null values.



This is created by using the ‘<tr>’ tags for each line and the ‘<td>’ to sort out the different cells with the code that needs to go in there, the code that does the table creation is:

with DataMain.ProductsSet do

begin

for i := 0 to (length(OrderArray) - (length(OrderArray) -

(ArrayCount - 1))) do

begin

LineTotal := 0;

Close;

WriteLn(htmlfile, '<tr>');

WriteLn(htmlfile, '<td align="right">' + OrderArray[i].Product + '</td>');

CommandText := 'SELECT \* FROM Products WHERE ProductID Like "' +

OrderArray[i].Product + '"';

open;

WriteLn(htmlfile, '<td>' + FieldValues['ProductName'] + '</td>');

WriteLn(htmlfile, '<td align="right">' + intToStr(OrderArray[i].Quantity)

+ '</td>');

UnitPrices := FieldValues['UnitPrice'];

ShowMessage(currToStr(UnitPrices));

WriteLn(htmlfile, '<td align="right">' +

currToStr(FieldValues['UnitPrice']) + '</td>');

LineTotal := ((OrderArray[i].Quantity) \* (UnitPrices));

ShowMessage(currToStr(LineTotal));

WriteLn(htmlfile, '<td align="right">' + currToStr(LineTotal) + '</td>');

WriteLn(htmlfile, '<tr>');

end;

end;

This is a rather simple piece of code with a loop that just goes through the filled spaces of the ‘OrderArray’ and adds them to a new line, it also does an SQL query that searches for the relevant data that it needs.

The pseudo code for this is:

With the products table

While i less than or equal to Length(OrderArray)

Start at the first line of the table

Close the data base

Write ‘<tr>’ to the new file

Write ‘<td>’ followed by the current product ID from the ‘OrderArray’ ‘</td>’

Search the database for the product ID and return all data

Open the database to run the SQL

Write ‘<td>’ followed by the product name from the products table ‘</td>’

Write ‘<td>’ followed by the quantity of items from the ‘OrderArray’ ‘</td>’

Write ‘<td>’ followed by the unit price from the products table ‘</td>’

Write ‘<td>’ followed by the quantity from ‘OrderArray’ multiplied by the ‘UnitPrice’ from the products table ‘</td>’(this is the line total)

End

The other main algorithm is the adding to the ‘OrderArray’ this is done with a loop that goes through the ‘ItemsGrid’ on the ‘NewOrderForm’:

This code runs through and adds the products and their quantities to the ‘OrderArray’, this code also works if the user has entered two lines of the same ‘ProductID’ it will spot that and still just have the one ‘ProductID’ with the increased quantity. It uses to loops, with one inside the other to allow it to search the pre-existing array, before it decides to add the new ‘ProductID’ to the end and move on to the next line in the ‘ItemsGrid’

for i := 1 to ItemsGrid.RowCount - 1 do

begin

j := 0;

while (j < length(OrderArray) - 1) And

(OrderArray[j].Product <> (ItemsGrid.Cells[Column, i])) do

begin

Inc(j);

End;

if OrderArray[j].Product = (ItemsGrid.Cells[Column, i]) then

Begin

Inc(OrderArray[j].Quantity);

Inc(j);

end

else

begin

OrderArray[ArrayCount].Product := (ItemsGrid.Cells[Column, i]);

OrderArray[ArrayCount].Quantity :=

strToInt(ItemsGrid.Cells[Column + 1, i]);

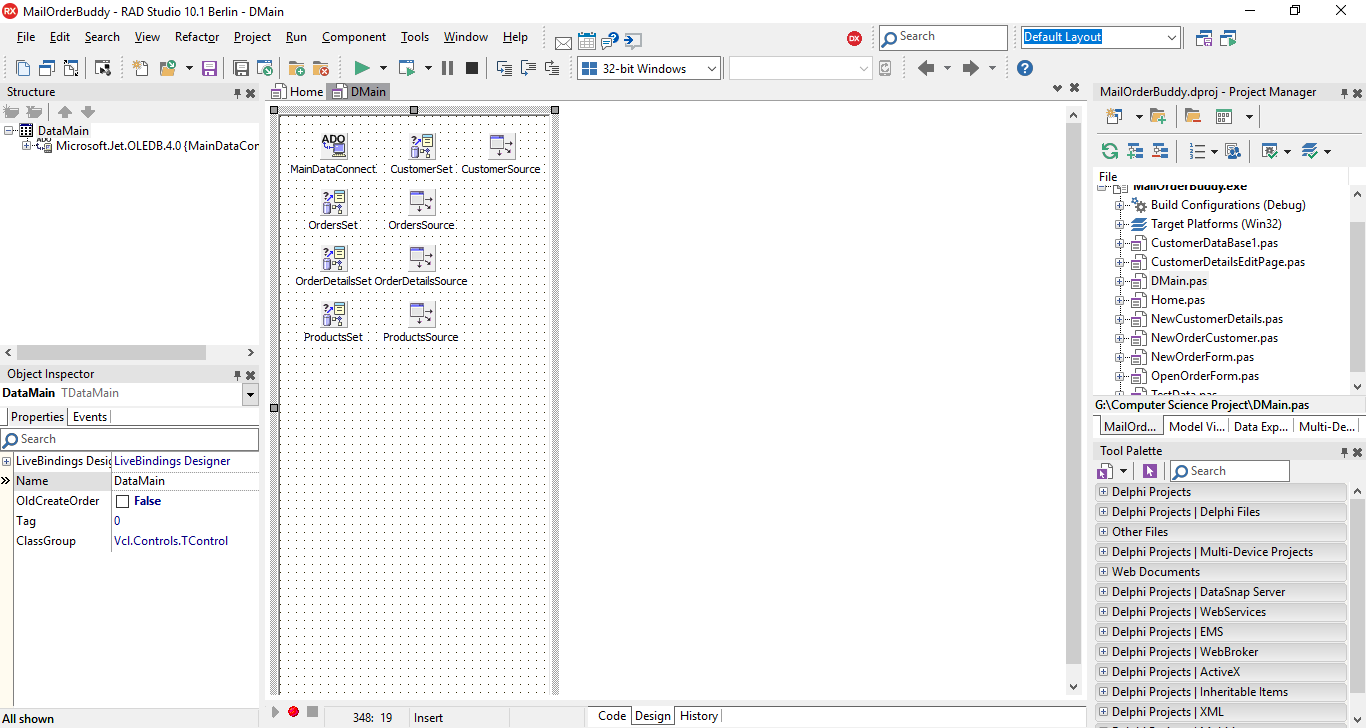
Inc(ArrayCount)

end;

end;

SQL

This is all of the SQL queries that the system uses, however it does not use much SQL, mainly just ‘SELECT’s based on certain varying parameters that the user has entered in to view the required record or records from a table, or like in the ‘Print Order procedure’ where the program changes the ‘CommandText’ based on what it needs to see and enter in to the html file, based on the current selected ‘ProductID’ from the ‘OrderArray’. This is the data module that the lets the system interact with the database, this is where all the main SQL takes place, it does take place throughout the program, but all the updating of tables is done with code, changing field values when the required tables are open.



select OrderID, Discount, OrderDetails.ProductID, Quantity, UnitPrice from OrderDetails, Products

Where Products.ProductID = OrderDetails.ProductID

select \* from Orders

select \* from Products

select \* from Customers