```
unit NewOrderForm;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
  System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.DBCtrls, Vcl.Mask,
  IdBaseComponent, IdDateTimeStamp, Data.DB, Vcl.Grids, Vcl.DBGrids, HTTPApp,
  DBWeb, ShellApi;

type { Creation of TOrderRecord, this is used in the running of 'Order Array'
    as it sets out the shape and diamentions of the array. }
  TOrderRecord = Record
    Product: String;
    Quantity: Integer;
    UnitPrice: Currency;
  End;

  TNewOrdersForm = class(TForm)
    Homebtn: TButton;
    DateLbl: TLabel;
    CompanyNameLbl: TLabel;
    ItemsLbl: TLabel;
    TakenByLbl: TLabel;
    DBProductList: TDBLookupComboBox;
    ProductSearchEdit: TEdit;
    LookUpBtn: TButton;
    DBCompanyNameEdit: TDBEdit;
    AddToItemsBtn: TButton;
    SaveOrderBtn: TButton;
    DBCustomerIDEdit: TDBEdit;
    Label1: TLabel;
    DiscountEdit: TEdit;
    DiscountLbl: TLabel;
    RemoveBtn: TButton;
    ItemsGrid: TStringGrid;
    QuantityEdit: TEdit;
    quantityLbl: TLabel;
    TotalPriceLbl: TLabel;
    TotalPriceEdit: TEdit;
    PrintBtn: TButton;
    TodayDateLbl: TLabel;
    procedure HomebtnClick(Sender: TObject);
    procedure LookUpBtnClick(Sender: TObject);
    procedure AddToItemsBtnClick(Sender: TObject);
    procedure SaveOrderBtnClick(Sender: TObject);
    procedure RemoveBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PrintBtnClick(Sender: TObject);
```

```pascal
  private
    { Private declarations }
    Column, Row: Integer;
    PriceT: Currency;
    htmlfile: text;
    ArrayCount: Integer;
    OrderArray: array of TOrderRecord;
  public
    { Public declarations }
  end;

var
  NewOrdersForm: TNewOrdersForm;

implementation

{$R *.dfm}

uses Home, CustomerDataBase1, CustomerDetailsEditPage, NewCustomerDetails,
  DMain,
  NewOrderCustomer, TestData;

procedure TNewOrdersForm.AddToItemsBtnClick(Sender: TObject);
{ Adds the selected item from the 'DBProductList' to the 'ItemsGrid' }
var
  i: Integer;
begin
  ItemsGrid.RowCount := ItemsGrid.RowCount + 1;
  { Increasing the number of rows in the 'ItemsGrid' so that the new item
    can be added and not over write the last item added. }
  ItemsGrid.Cells[Column, Row] := DBProductList.text;
  ItemsGrid.Cells[Column + 1, Row] := QuantityEdit.text;
  ItemsGrid.Cells[Column + 2, Row] := DataMain.ProductsSet.FieldValues
    ['UnitPrice'];
  Inc(Row); { So it starts off at the bottom of the grids when
    a new item is added. }
  PriceT := 0;
  { Setting the global price variable to 0 so that it is not added
    to the previous price and subsequently get the wrong price. }
  for i := 1 to ItemsGrid.RowCount -
    1 do { Goes through the 'ItemsGrid' adding up all the prices with
    the quantities to get a total price to display. }
  begin
    PriceT := (strToFloat(ItemsGrid.Cells[Column + 2, i]) *
      strToInt(ItemsGrid.Cells[Column + 1, i])) + PriceT;
    { Adding on the the current price to get a running total. }
  end;

  TotalPriceEdit.text := currToStr(PriceT);
  { Displaying the calculated price in a 'editbox' on the form. }
```

```pascal
end;

procedure TNewOrdersForm.FormCreate(Sender: TObject);
{ When the form is created it does some setting up of the 'ItemsGrid' and making sure the correct buttons are displayed. }
begin
  Column := 0;
  { Sets the 'Column' value of the 'ItemsGrid' to '0' to make sure it starts at the top of the grid and not in an unpredicted place. }
  Row := 0;
  { Sets the 'Row' value of the 'ItemsGrid' to '0' to make sure it starts on the left of the grid and not in an unpredicted place. }
  ItemsGrid.Cells[Column, Row] := 'ProductID:';
  { Adds the headers to the table columns, starting from the left. }
  ItemsGrid.Cells[Column + 1, Row] := 'Quantity:';
  { Second column header being set. }
  ItemsGrid.Cells[Column + 2, Row] := 'Price:';
  { Third column header begin set. }
  Inc(Row); { Incrementing the row count so that it does not rewrite the previous line and starts on a new on. }
  PrintBtn.Visible := false;
  { Sets the visiblilty of the 'Print' button to invisible, as it is not required yet. }
  SaveOrderBtn.Visible := True;
  { Sets the visibility of the 'Save' button to visible as it is required straight away. }
  TodayDateLbl.Caption := dateToStr(Now);
  { Sets the date label to the current date. }
  SaveOrderBtn.Visible := True;
  { Sets the Save button to visable as it is needed when the form opens. }
  PrintBtn.Visible := false;
  { Sets the visiblity of the Print button to invisible as it is not needed until the order has been saved. }
end;

procedure TNewOrdersForm.HomebtnClick(Sender: TObject);
{ Closes the current form and makes the 'Main' form visible again so that the user can carry out a new task. }
var
  i: Integer;
begin
  with DMain.DataMain.CustomerSet do
  begin
    Close;
    { Data Set must be closed before changing the command text parameter. }
    DiscountEdit.text := '0';
    { Clears the 'Discount' EditBox and set the value to zero. }
    CommandText := 'SELECT * FROM Customers';
    { Changes the command text to show all of the records in the DataSet. }
    Open; { Opening the Data Set will run the new Command Text and return the records that meet the query. }
    for i := 0 to ItemsGrid.RowCount -
      1 do { Start of a loop that clears the 'ItemsGrid', goes through the loop for as long as the 'ItemsGrid' is }
    begin
      ItemsGrid.RowCount := ItemsGrid.RowCount - 1;
      { Decrases the 'RowCount' for every row in the 'ItemsGrid' }
    end;
    TotalPriceEdit.text := '';
    { Sets the 'TotalPrice' EditBox to null so that is can be used from fresh the next time it is needed and does not start form the la
    st total price. }
  end;

  HomeForm.Visible := True;
  { Shows the 'Main' form so the user can carry out their next task. }
  NewOrdersForm.Close;
  { Closes the 'NewOrders' form as it is no longer needed. }
end;

procedure TNewOrdersForm.LookUpBtnClick(Sender: TObject);
{ Procedure to search the 'Products' table in the DataBase. }
begin
  with DMain.DataMain.ProductsSet do
  begin
    Close;
    { Data Set must be closed before changing the command text parameter. }
    CommandText := 'SELECT * FROM Products WHERE ProductName Like "%' +
      ProductSearchEdit.text + '%"';
    Open; { Opening the Data Set will run the new Command Text and return the records that meet the query. }
    ShowMessage('Search Complete');
    { A message box to let the user know the query has been run and a customer found. }
  end;
end;

procedure TNewOrdersForm.PrintBtnClick(Sender: TObject);
{ Procedure which handels the creation and opening of the order form file. }
var { Declaring the Variables that are needed for this procedure. }
  FileName, Address, FileString: String;
  i: Integer;
  LineTotal, UnitPrices, VATValue, VATPrice: Currency;
begin
  FileName := DataMain.OrdersSet.FieldValues['OrderID'];
  { Creating the filename, from the last 'OrderID' in the DataBase 'Orders' table. }
  assignfile(htmlfile, 'Order ' + FileName + '.html');
  { Creates the file in memory that everything is going to be written to. }
  rewrite(htmlfile); { Opens the file to receive text and editing. }

  WriteLn(htmlfile, '<!DOCTYPE HTML>');
  { Setting up the html file, by declaring its 'DOCTYPE'. }

  WriteLn(htmlfile, '<html> <head> <meta http-equiv="Content-Type"');
  { Initilise that text to follow is the main part of the page and is html. }

  WriteLn(htmlfile, '<title>Mail Order Buddy Order</title>');
  { Creates a title for the page, top left hand }

  WriteLn(htmlfile, '</head><body class="verdana">');
  { Sets the body font type and stops the heading. }
  WriteLn(htmlfile, '<h1 align=center>All 4 Wheels Order</h1>');
  { Adds the title, this is the company name. }
```

```pascal
  WriteLn(htmlfile, '<p align="center">');
  { Aligns a paragraph to the center under the title. }
  WriteLn(htmlfile, 'Order Number: ' + FileName + '');
  { Adds text 'Order Number', followed by the 'OrderID'. }

  WriteLn(htmlfile, '<p align="right">');
  { Aligns the 'Customer' to the right of the page. }
  WriteLn(htmlfile, '' + DataMain.CustomerSet.FieldValues['CompanyName']);
  { Adds the selected 'CompanyName' from the 'Customers' table. }
  WriteLn(htmlfile, '<br>');
  Address := DataMain.CustomerSet.FieldValues['Address'];
  { Adds the selected 'Address' from the 'Customers' table. }
  WriteLn(htmlfile, '' + Address);
  WriteLn(htmlfile, '<br>');
  WriteLn(htmlfile, '' + DataMain.CustomerSet.FieldValues['City']);
  { Adds the selected 'City' from the 'Customers' table. }
  WriteLn(htmlfile, '<br>');
  WriteLn(htmlfile, '' + DataMain.CustomerSet.FieldValues['County']);
  { Adds the selected 'County' from the 'Customers' table. }
  WriteLn(htmlfile, '<br>');
  WriteLn(htmlfile, '' + DataMain.CustomerSet.FieldValues['PostCode']);
  { Adds the selected 'PostCode' from the 'Customers' table. }
  WriteLn(htmlfile, '<br><br>');
  WriteLn(htmlfile, 'Telephone: ' + DataMain.CustomerSet.FieldValues['Phone']);
  { Adds the selected 'Phone Number' from the 'Customers' table. }
  WriteLn(htmlfile, '<br><br><br>');
  WriteLn(htmlfile, '' + dateToStr(Now) + '');
  { Adds the current date that is set on the computer system. }
  WriteLn(htmlfile, '</p>'); { Ends the paragraph part of the page. }

  WriteLn(htmlfile, '<br><br>'); { Drops two line of the page. }

  WriteLn(htmlfile, '<table align="left" border="2"');
  { Setting up the table aligning it to the left, like it was in the document already used. }
  WriteLn(htmlfile, '<tr>'); { Creates the first row of the table. }
  WriteLn(htmlfile, '<td width="100">');
  { Creates the first cell and sets the width to 100 pixels. }
  WriteLn(htmlfile, 'Product ID'); { Sets the string inside the cell. }
  WriteLn(htmlfile, '</td>'); { Ends the cell. }
  WriteLn(htmlfile, '<td width="350">');
  { Adds a new cell to the table and set the width  to 350 pixels, bigger than the last as more data needs to be held. }
  WriteLn(htmlfile, 'Product Desctription');
  { Sets the string inside the cell. }
  WriteLn(htmlfile, '</td>'); { Ends the cell. }
  WriteLn(htmlfile, '<td width="100">');
  { Creates another cell and sets the width to '100' as it only hold a number. }
  WriteLn(htmlfile, 'Quantity');
  { Sets the string inside the cell, this will be the header. }
  WriteLn(htmlfile, '</td>'); { Ends the cell. }
  WriteLn(htmlfile, '<td width="100">');
  { Creates a new cell that will hold the price header. }
  WriteLn(htmlfile, 'Price'); { Sets the colummn header. }
  WriteLn(htmlfile, '</td>'); { Ends the cell. }
  WriteLn(htmlfile, '<td width="100">');
  { Creates a cell that will be the column for the line total. }
  WriteLn(htmlfile, 'Sub Total'); { Column header setting. }
  WriteLn(htmlfile, '</td>'); { Ends the cell. }
  WriteLn(htmlfile, '</tr>'); { Ends the first line of the table. }

  with DataMain.ProductsSet
    do { Selects the 'Products' table from the DataBase. }
  begin
    for i := 0 to (length(OrderArray) - (length(OrderArray) - (ArrayCount - 1)))
      do { Initalisation of a loop that will go through the 'OrderArray' a record at a time. }
    begin
      LineTotal := 0;
      { Sets the line total to zero so that it is not adding to the previous line total. }
      Close; { Data Set must be closed before changing the command text parameter. }
      WriteLn(htmlfile, '<tr>'); { Starts a new row on the table in the file. }
      WriteLn(htmlfile, '<td align="right">' + OrderArray[i].Product + '</td>');
      { Enters the 'ProductID' from the 'OrderArray' at position 'i'. }
      CommandText := 'SELECT * FROM Products WHERE ProductID Like "' +
        OrderArray[i].Product + '"';
      { Searches the DataBase for the record where the 'ProductID' is the same as the 'ProductID' in the 'OrderArray' at position 'i'.
}
      Open; { Opening the Data Set will run the new Command Text and return the records that meet the query. }
      WriteLn(htmlfile, '<td>' + FieldValues['ProductName'] + '</td>');
      { Adds the 'ProductName' from the record that has been returned from the DataBase. }
      WriteLn(htmlfile, '<td align="right">' + intToStr(OrderArray[i].Quantity)
        + '</td>');
      { Adds the 'Quantity' form the 'OrderArray' at position 'i' in to the 'Quantity' column of the table. }
      UnitPrices := FieldValues['UnitPrice'];
      { Sets the variable 'UnitPrices' to the 'UnitPrice' from the 'Products' table. }
      WriteLn(htmlfile, '<td align="right">' +
        floatToStrf(FieldValues['UnitPrice'], ffcurrency, 18, 2) + '</td>');
      { Adds the 'UnitPrice' to the table from the 'Products' table in the database. }
      LineTotal := ((OrderArray[i].Quantity) * (UnitPrices));
      { Works out the line total by doing the 'OrderArray' quantity multiplied by the 'UnitPrice'. }
      WriteLn(htmlfile, '<td align="right">' + floatToStrf(LineTotal,
        ffcurrency, 18, 2) + '</td>');
      { Adds the calculated line total to the last cell of the table in the page. }
      WriteLn(htmlfile, '<tr>');
      { Adds a new line to the table so that the values already enter to the table are not over written. }
    end;
  end;

  WriteLn(htmlfile, '</tr>'); { Ends the table lines. }

  WriteLn(htmlfile, '<tr>');
  { Adds a new line to the table so that you can add the price excluding the VAT. }
  WriteLn(htmlfile, '<td></td>');
  WriteLn(htmlfile, '<td></td>');
  { Formats the cell position so that it starts on the penultimate cell. }
  WriteLn(htmlfile, '<td></td>');
  WriteLn(htmlfile, '<td align="right">Price Ex-VAT: </td>');
  { Adds the the header for the folowing cell in the same row. }
  VATPrice := ((PriceT / 120) * 100);
  { Works out the VAT, which is 20% of the original price, so you have to divide by 120. }
  WriteLn(htmlfile, '<td align="right">' + floatToStrf(VATPrice, ffcurrency, 18,
    2) + '</td>'); { Adding the calculated VAT price to the table in the file. }

  WriteLn(htmlfile, '<tr>'); { Adds a new line to the table. }
  WriteLn(htmlfile, '<td></td>');
  WriteLn(htmlfile, '<td></td>');
  { Formats the cell position so that it starts on the penultimate cell. }
  WriteLn(htmlfile, '<td></td>');
  WriteLn(htmlfile, '<td align="right">VAT: </td>');
  { Adds the header for the cell which will display the VAT which is included in the price }
  VATValue := (PriceT - VATPrice);
  { Works out how much VAT there is, does this by taking away the 'VATPrice' from the 'TotalPrice'. }
  WriteLn(htmlfile, '<td align="right">' + floatToStrf(VATValue, ffcurrency, 18,
    2) + ''); { Adding the calculated VAT value to the table in the file. }

  WriteLn(htmlfile, '<tr>'); { Adds a new line to the table. }
  WriteLn(htmlfile, '<td></td>');
  WriteLn(htmlfile, '<td></td>');
  { Formats the cell position so that it starts on the penultimate cell. }
  WriteLn(htmlfile, '<td></td>');
  WriteLn(htmlfile, '<td align="right">Price In-VAT: </td>');
  { Adds the header for the cell to follow. }
  WriteLn(htmlfile, '<td align="right">' + floatToStrf(PriceT, ffcurrency, 18,
    2) + ''); { Adding the 'TotalPrice' to the table. }

  WriteLn(htmlfile, '</tr>'); { Ending the table lines. }
  WriteLn(htmlfile, '</table>');
  { Ends the table and stops adding things to the table. }

  WriteLn(htmlfile, '</body></html>');
  { Ends the main body of the file, then ends the 'html' part of the file. }
  closefile(htmlfile);
  { Closes and saves the file, next to the aplication file. }

  FileString := 'G:\Computer Science Project\Win32\Debug\Order ' + (FileName) +
    '.html'; { Gets the 'path' for the file that has just been created and saves it in a variable. }

  ShellExecute(Handle, 'open',
    'C:\Program Files (x86)\Internet Explorer\iexplore.exe',
    PWideChar(FileString), nil, SW_SHOWNORMAL);
  { A 'ShellExecute' procedure that handles the opening of the file that has just been created. }
  { It opens the file automatically, this makes it easier for the user to use as then they don't have to go and search for the file. }

  ShowMessage('Printed');
end;

Procedure TNewOrdersForm.RemoveBtnClick(Sender: TObject);
{ Procedure that removes the selected row from the 'ItemsGrid'. }
var
  i, j: Integer;
begin
  for i := 0 to 2 do { }
  begin
    ItemsGrid.Cells[i, ItemsGrid.Row] := '';
    { Clears the selected row and puts all of the cell values to null. }
  end;
  for j := ItemsGrid.Row to ItemsGrid.RowCount do
  begin { Goes through the 'ItemsGrid' starting at the selected row and moves all of rows up one until the end of the 'ItemsGrid' }
    ItemsGrid.Cells[Column, j] := ItemsGrid.Cells[Column, j + 1];
    { Moves the first column in the table up by one row. }
    ItemsGrid.Cells[Column + 1, j] := ItemsGrid.Cells[Column + 1, j + 1];
    { Moves the second column in the table up by one row. }
    ItemsGrid.Cells[Column + 2, j] := ItemsGrid.Cells[Column + 2, j + 1];
    { Moves the second column in the table up by one row. }
  end;
  ItemsGrid.RowCount := ItemsGrid.RowCount - 1;
  { Decreases the row count by one to remove the empty space that will be left in the 'ItemsGrid'. }
  Row := ItemsGrid.RowCount;
  { Sets the variable 'Row' to the length of the 'ItemsGrid' }
  PriceT := 0;
  { Sets the price variable to zero ready to recalculate the total price. }
  for i := 1 to ItemsGrid.RowCount - 1 do
  begin
    PriceT := (strToFloat(ItemsGrid.Cells[Column + 2, i]) *
      strToInt(ItemsGrid.Cells[Column + 1, i])) + PriceT;
    { Recalculates the total price and saves it in the variable 'PriceT'. }
  end;

  TotalPriceEdit.text := currToStr(PriceT);
  { Displays the new calculated total price in the 'TotalPrice' EditBox. }

  ShowMessage('Deleted');
end;

procedure TNewOrdersForm.SaveOrderBtnClick(Sender: TObject);
{ Procedure to save the order to the database in the 'Orders' and 'OrderDetails' tables. }
var
  i, j, QuantityInStock: Integer;
  Discount: Real;
  TotalPrice: Currency;
  DateT: string;

```

```pascal
begin
  PrintBtn.Visible := True;
  { Sets the print button to visible so that it can be used after the procedure has finished. }
  SaveOrderBtn.Visible := false;
  { Sets the save button to invisible as it is no longer needed and could cause an error if clicked again with orders. }
  SetLength(OrderArray, ItemsGrid.RowCount);
  { Sets the length of the 'OrderArray' to the length of the 'ItemsGrid' so that there are enough spaces, and no overflow errors. }
  ArrayCount := 0;
  { Sets the 'OrderArray' count to zero so that the program knows how many spaces are filled, and adds from the start. }
  for i := 1 to ItemsGrid.RowCount -
    1 do { Goes through the 'ItemsGrid' missing out the first row as we don't need to add the headers to the database. }
  begin
    j := 0; { Sets 'j' variable to zero so that it searches through the whole of the 'OrderArray' each time and does not start from a random place. }
    while (j < length(OrderArray) - 1) And
      (OrderArray[j].Product <> (ItemsGrid.Cells[Column, i]))
      do { Goes through the 'OrderArray' until it find two 'ProductID's that are the same. }
    begin
      Inc(j); { Increases 'j' by one so that it will move on the next row in the 'ItemsGrid'. }
    End;
    if OrderArray[j].Product = (ItemsGrid.Cells[Column, i])
    then { Searching for the 'ProductID' from the 'ItemsGrid' that matches one in the 'OrderArray'. }
    Begin
      Inc(OrderArray[j].Quantity);
      // OrderArray[j].Quantity:= (OrderArray[j].Quantity + ItemsGrid.Cells[Column + 1, i]);
      { Adds the existing quantity of the selected 'ProductID' to the new quantity of the 'ProductID' that has been found. }
      Inc(j); { Increments 'j' by one so that it moves on to the next item in 'OrderArray'. }
    end

    else
    begin
      OrderArray[ArrayCount].Product := (ItemsGrid.Cells[Column, i]);
      { If the 'ProductID' is not found then it will add a new item to the 'OrderArray'. }
      OrderArray[ArrayCount].Quantity := strToInt(ItemsGrid.Cells[Column + 1, i]
        ); { Adds the quantity to the array that is stored in the 'ItemsGrid'. }
      Inc(ArrayCount)
      { Increments the 'ArrayCount' by one as there in now another space that has been filled in the 'OrderArray'. }
    end;
  end;

  if DiscountEdit.text <> ''
  then { Checks to see if there is a value in the 'EditBox' or if it is empty. }
  Begin
    Discount := strToInt(DiscountEdit.text);
    { If there is a value, then it converts it to an integer and then saves it in a variable space. }
    if Discount > 0
    then { Makes sure that the value entered is more than zero. }
    Begin
      TotalPrice := PriceT - (PriceT * (Discount / 100));
      { If the value is more than zero then it will calculate the new 'TotalPRice' with the discount applied. }
    End
    else { If the value entered is equal to, or less than then the 'TotalPrice' is the same. }
    begin
      TotalPrice := PriceT;
    end;
  End;
  PriceT := TotalPrice;
  { Makes sure the gobal variable has been changed for other procedures to use. }

  with DMain.DataMain.OrdersSet do
  begin
    Append; { Opens the 'Orders' table in the database so that it is ready for a new record to be added to the end. }
    FieldValues['CustomerID'] := DBCustomerIDEdit.text;
    { Adds the 'CustomerID' from the 'EditBox' on the form, in to the correct column in the 'Orders' table. }
    FieldValues['TotalPrice'] := TotalPrice;
    { Adds the total price from the 'TotalPrice' variable that has been calculated, in to the correct column in the 'Orders' table. }
    FieldValues['OrderDate'] := TodayDateLbl.Caption;
    { Adds the current date from the 'EditBox' on the form that holds the current date, in to the correct column in the 'Orders' table.
 }
    Post; { Save the new record. }
  end;

  with DMain.DataMain.OrderDetailsSet do
  begin
    for i := 0 to ArrayCount -
      1 do { Sets up a loop that goes through the 'ArrayCount', so as many times as there are spaces filled in the 'OrderArray'. }
    begin
      Append; { Opens the 'OrderDetails' table in the database so that it is ready for a new record to be added to the end. }
      FieldValues['OrderID'] := DataMain.OrdersSet.FieldValues['OrderID'];
      { Adds the 'OrderID' that has already been created, retreaves it from the 'Orders' table. }
      FieldValues['ProductID'] := strToInt(OrderArray[i].Product);
      { Adds the current 'ProductID' from the 'OrderArray'. }
      FieldValues['Quantity'] := OrderArray[i].Quantity;
      { Adds the current 'Quantity' from the 'OrderArray'. }
      FieldValues['Discount'] := Discount;
      { Adds the current value stored in the variable 'Discount'. }
      Post; { Saves the new record to the table. }
    end;
  end;

  with DMain.DataMain.ProductsSet do
  begin
    for i := 0 to ArrayCount -
      1 do { Sets up a loop that goes through the 'ArrayCount', so as many times as there are spaces filled in the 'OrderArray'. }
    begin
      Close; { Data Set must be closed before changing the command text parameter. }
      CommandText := 'SELECT * FROM Products WHERE ProductID = ' + OrderArray[i]
        .Product + '';
      Open; { Opening the Data Set will run the new Command Text and return the records that meet the query. }
      QuantityInStock := FieldValues['UnitsInStock'];
      { Gets the value from the record that has been selected based on the SQL query. }
      QuantityInStock := QuantityInStock - OrderArray[i].Quantity;
      { Works out the new quantity and stores it in the variable, 'QuantityInStock'. }
      Edit; { Sets the DataSet up ready for editing values inside of it. }
      FieldValues['UnitsInStock'] := QuantityInStock;
      { Changes the 'UnitsInStock' to the value that is in the 'QuantityInStock' variable. }
      Post; { Save the new record. }
    end;

  end;
  ShowMessage('The total price is: £' + floatToStrf(TotalPrice, ffcurrency, 18,
    2) + ''); { Tells the user the total price, formated to 2 decimal places, so they can then tell the customer and take the payment.
  }
end;

end.
```