

CIS 3309 Section 4 Comprehensive Lab Project

The primary task is to complete an order entry system for MMA Books. The application needs to query customers and products data from a database and commit invoices to the database, while maintaining product inventory levels. The image below shows the form for the project. The form and most of the form code are completed for you. You still need to add classes and database code, and the form code to use them. The form is divided into 3 sections:

1. The Customer section either gets a customer from or adds a customer to the database
2. The Products section shows all the MMA Books products and allows a user to select products and add a quantity of them to the cart
3. The Cart section shows the products in the care, their sum of unit prices, shipping charge, sales tax and invoice total, edit or delete products from the cart, clear the cart and submit the invoice to the database

MMA Books Order Entry System

Customer

Customer ID:

Name:

Address:

City:

State:

Zipcode:

Find Customer By ID

Add New Customer

Clear Customer

Products

Product Code	Description	Unit Price
ZJST	Murach's JavaScript (2nd Edition)	\$54.50 5937
ZGRY	Murach's jQuery (2nd Edition)	\$54.50 677
A46C	Murach's ASP.NET 4.6 Web Programming with C# 2015	\$57.50 4632
A46V	Murach's ASP.NET 4.6 Web Programming with VB 2015	\$57.50 3974
ADC4	Murach's ADO.NET 4 with C# 2010	\$56.50 3756
ADV4	Murach's ADO.NET 4 with VB 2010	\$56.50 4538
CRFC	Murach's CICS Desk Reference	\$50.00 1865
CS15	Murach's C# 2015	\$56.50 5011
DB1R	DB2 for the COBOL Programmer, Part 1 (2nd Edition)	\$42.00 4825
DB2R	DB2 for the COBOL Programmer, Part 2 (2nd Edition)	\$45.00 621
JAVP	Murach's Java Programming	\$56.50 3455
JSP2	Murach's Java Servlets and JSP (3rd Edition)	\$57.50 4999
MCCB	Murach's Structured COBOL	\$62.50 2386
MCCP	Murach's CICS for the COBOL Programmer	\$54.00 2368
MMag	Mad Magazine	\$5.00 500
SQL2	Murach's SQL Server 2012	\$57.50 2465

Product Code:

Description:

Unit Price:

On Hand Quantity:

Quantity To Purchase:

Add To Cart

Cart

Product Total: Sales Tax:

Shipping: Invoice Total:

Edit Item Delete Item Clear Cart Submit Order

There is an additional Form, frmEditItem, with most of the code included that allows the UnitPrice and Quantity to be changed in the cart. The Form below shows that the Unit Price and Quantity can be changed but Product Code and Item Total cannot. The Item Total is automatically calculated when either the Unit Price or Quantity are changed.

Edit Item...

Product Code: Minimize

Unit Price:

Quantity:

Item Total:

OK Cancel

Note that all the numeric data in the MMABooks database are either integer or decimal.

MMABooksDB class

This class has the `GetSqlConnection` to get a `SqlConnection` to the database and a method called `ExecuteNonQuery` to execute inserts, updates and deletes from the database. You will need the following methods.

`GetSqlConnection` – Gets a `SqlConnection` to the database using the connection string

`ExecuteNonQuery` – Accepts a `SqlCommand` object and a `SqlConnection` object to call the `SqlCommand`'s `ExecuteNonQuery` method. This will save dozens of lines of code in your insert, update and delete methods. I suggest coding your `InsertCustomer` method in the `CustomerDB` class to check if it works and cut all code from try-catch-finally to the last return at the end of the method and paste it into `ExecuteNonQuery`. Then replace the cut code in `InsertCustomer` with a call to `ExecuteNonQuery` in `MMABooksDB`.

Customer classes and Customer Form section

You will need to code 2 classes for a Customer:

1. `Customer` – Contains the data and methods for a customer
2. `CustomerDB` – Static class that contains methods to get data from or put data into a database

Customer class

The `Customer` class needs data to contain the fields from the `Customers` table in the `MMABooks` database. Look at the database in Visual Studio C#. You can do this by using the drop and drag approach used in chapters 18 and 19. Make sure to add all tables from the database to see the fields in each table. They will tell you the data type and name of each field. I strongly suggest that you name the data auto-implemented properties in your class with the same names as the `Customer`'s field names. This will make coding you `CustomerDB` class much easier by eliminating any ambiguity of data. Ignore and foreign keys associated with this table. The `Invoices` table is not necessary for any `Customer` database operations for our problem. After completing your `Customer` class auto-implemented properties, code the following:

1. A default constructor
2. A parameterized constructor that accepts and set all `Customer` data
3. A `ToString` method to return a string with all the `Customer` data on one line

CustomerDB class

The `CustomerDB` class contains methods that allow your application to query a `Customer` from the database, insert a `Customer` into the database and query the max `CustomerID` from the database. You will need the following methods.

1. A method `GetCustomer` that accepts a `CustomerID` and returns a `Customer` if the query is successful and returns a null if the query was not successful.
2. A method `InsertCustomer` that accepts the data fields for a `Customer` and inserts the `Customer` data into the database. It returns a integer greater than zero if the insert is successful (will return a 1 because 1 record in the database was affected by the insert, which will be returned by `ExecuteNonQuery`), a zero if the record was not inserted into the database (`ExecuteNonQuery`

will return the zero) and a -1 if an exception is thrown by the insert. You should write a wrapper method to accept a Customer object and, in the method, breaks the Customer into fields and calls the InsertCustomer method that accepts the data fields.

3. A method GetMaxID that queries the database and returns the max CustomerID from the database if the query is successful and -1 if it throws an exception.

Customer Section on Form

After completing the Customer and CustomerDB classes, you should add the code to frmMain Customer section to test them. Look at the comments in the Customer Section for frmMain. You will see comments like:

```
//*****Customer 1. Add code here
```

This shows the comment for step 1 below. Follow these steps:

1. Enter code in to query a Customer from the database by CustomerID in btnFindCustomerBy_Click
2. Add condition: If a Customer was found in DB set TextBoxes, Buttons and copy data to TextBoxes.
3. Add code to copy the data from the Customer to the TextBoxes in the Customer section in btnFindCustomerBy_Click
4. Add code to instantiate a Customer that uses the parameterized constructor with the data in the Customer section TextBoxes in btnAddNewCustomer_Click. Put -1 in for CustomerID. The correct CustomerID will be added after the insert because this field is auto-numbered in the database
5. Add code to insert the Customer object into the database by calling the wrapper method InsertCustomer in CustomerDB in the btnAddNewCustomer_Click method
6. Add code to query the max CustomerID from the database in btnAddNewCustomer_Click

Test you Customer section code by querying and adding Customers. Check in your test app that you used to look at your database to see the Customer added to the Customers table.

Product classes and the Products Form section

You will need 3 classes for Products.

1. Product – Contains the data and methods for a product
2. ProductList – To contain multiple products
3. ProductDB – Static class that contains methods to get data from or put data into a database

Product class

The Product class needs data to contain the fields from the Products table in the MMABooks database. Look at the database in Visual Studio C# again. Again, I strongly suggest that you name the data fields in your class with the same names as the Product's field names. Ignore and foreign keys associated with this table. The InvoiceLineItems table is not necessary for any Product database operations for our problem. After completing your Product class auto-implemented properties, code the following:

1. A default constructor

2. A parameterized constructor that accepts and set all Product data
3. A ToString method to return a string with all the Product data on one line

ProductList class

The ProductList class inherits a List of Product and only contains a ToString method.

1. Add the ToString in the ProductList method prints the List of products by calling the Product ToString method in a loop

ProductDB class

The ProductDB class contains methods that allow your application to query a Product from the database, insert a Product into the database and query the max CustomerID from the database. You will need the following methods.

1. A method GetProduct that accepts a ProductCode and returns a Product if the query is successful and returns a null if the query was not successful.
2. A method GetAllProducts that returns all Products in the Products table in a ProductList if the query is successful and returns a null if the query was not successful.
3. A method UpdateProduct that accepts the fields of a Product and update them in the appropriate record in the Products table
4. Add code to insert the Customer object into the database by calling the wrapper method UpdateProduct in ProductDB
5. A method UpdateOnHandQuantity to update the Products table OnHandQuantity after a purchase has been made. It accepts a ProductCode (string) and an integer that represents the number of the product purchased. A negative integer decreases OnHandQuantity and a positive integer increases it.

Products Section on Form

After completing the Product classes, you should add the code to frmMain Products section to test them. Look at the comments in the Products Section for frmMain. When the application is run, the Products in the ListBox should be visible as shown in the frmMain above.

Next, follow these steps:

1. Enter code to load all Products from the database into your application in GetAllProducts.
2. Enter code to copy Products from the List returned in step 1.
3. Enter code to get the selected item in lstProducts and copy it to the class level Product p.
4. Enter code to copy data from selected item to Product TextBoxes and set txtQuantityToPurchase to empty string

The code for the btnAddToCart_Click event handler will be added later.

OrderOptions classes

The OrderOptions class contains data necessary to complete an order. It contains the sales tax rate, fist book shipping charge and the charge for additional books. Please look at the OrderOptions table in the database for field names. You will need two classes:

1. OrderOptions – Contains the three fields described above
2. OrderOptionsDB – Contains code to get the data for OrderOptions from the database

OrderOptions

The OrderOptions class has 3 fields from the OrderOptions table in the database. Add the necessary auto-implemented properties to the class to contain the data from the table. After completing your OrderOptions class auto-implemented properties, code the following:

1. A default constructor
2. A parameterized constructor that accepts and set all OrderOptions data
3. A ToString method to return a string with all the OrderOptions data on one line

OrderOptionsDB

The OrderOptions class contains methods that allow your application to query OrderOptions from the database. You will need the following method.

GetOrderOptions – Query OrderOptions from OrderOptions table in database

Invoice classes and the Cart Form section

This is the most challenging task. You create all the code in this class to manage an Invoice, its InvoiceLineItems and perform database operations on Invoice data. The database operations include writing a one-to-many relation into the database. The Invoice classes include:

1. InvoiceLineItem – Contains data about a Product added to the Invoice
2. Invoice – Contains the Invoice data described below and a list of InvoiceLineItem
3. InvoiceDB – Static class that contains methods to get data from or put data into a database

Add code to the frmEditItem below. See comments on form for more information.

1. Add code to frmEditItem at: `*****InvoiceLineItem 1. Add code here`
2. Add code to frmEditItem at: `*****InvoiceLineItem 2. Add code here`
3. Add code to frmEditItem at: `*****InvoiceLineItem 3. Add code here`
4. Add code to frmMain at: `*****InvoiceLineItem 4. Add code here`
5. Add code to frmMain at: `*****InvoiceLineItem 5. Add code here`

InvoiceLineItem

The InvoiceLineItem class needs data to contain the 5 fields from the InvoiceLineItems table in the MMABooks database. Look at the database in Visual Studio C# again. Again, I strongly suggest that you name the data fields in your class with the same names as the InvoiceLineItem's field names. After completing your InvoiceLineItems class auto-implemented properties, code the following:

1. A default constructor
2. A parameterized constructor that accepts and set all InvoiceLineItem data
3. A ToString method to return a string with all the InvoiceLineItem data on one line

Invoice

The Invoice class needs data to contain the 7 fields from the Invoices table in the MMABooks database. You will also need to add a List of InvoiceLineItems called invoiceLineItems and an OrderOptions

reference called `oo` to this data. Look at the database in Visual Studio C# again. Again, I strongly suggest that you name the data fields in your class with the same names as the Invoice's field names. After completing your Invoice class auto-implemented properties, code the following:

1. A method called `SetDate` that sets the `DateTime` object in the Invoice to `Now`
2. A default constructor that instantiates the List of `InvoiceLineItems`, gets the `OrderOptions` from the database and calls the `SetDate` method
3. A parameterized constructor that accepts and set all `InvoiceLineItem` data, instantiates the List of `InvoiceLineItems`, gets the `OrderOptions` from the database and calls the `SetDate` method
4. A `ToString` method to return a string with all the `InvoiceLineItem` data on one line
5. A property called `InvoiceDate` that facilitates the setting or getting of the date as a string from outside the class
6. A read-only property called `InvoiceLineItems` that gets `invoiceLineItems`
7. An indexer called `InvoiceLineItem` to set or get specific elements from `invoiceLineItems`
8. A method called `CalcProductTotal` that uses a loop to calculate the sum of `UnitPrices` in all the `InvoiceLineItems` in the List and assigns it to the `ProductTotal` field
9. A method called `CalcShipping` that calculates the shipping charge. Remember that there is a different cost for the first book and each additional book shipped and assigns it to the `Shipping` field
10. A method called `CalcSalesTax` that calculates the tax from the sum of `ProductTotal` and `Shipping` fields
11. A method called `CalcInvoiceTotal` that calculates the sum of the `ProductTotal`, `Shipping` and `SalesTax` fields and assigns it to `InvoiceTotal`
12. A method called `CalcAll` that calls the 4 previous methods in steps 8 to 11 to set all their fields
13. A method called `UpdateInvoiceLineItem` that accepts the index of the element in `invoiceLineItems` List that you want to update, a decimal `unitPrice` to update the `InvoiceLineItem`'s `UnitPrice` and an integer quantity to update the `InvoiceLineItem`'s `Quantity`, and calls the `CalcAll` to update the requisite fields
14. A method called `Add` to add an `InvoiceLineItem` to the `invoiceLineItems` List and calls the `CalcAll` to update the requisite fields
15. A method called `RemoveAt` to remove an `InvoiceLineItem` from the `invoiceLineItems` List by index and calls the `CalcAll` to update the requisite fields
16. A method called `Clear` to clear all `InvoiceLineItems` from the `invoiceLineItems` List and calls the `CalcAll` to update the requisite fields

InvoiceDB

The `InvoiceDB` class contains methods that allow your application to get the max `InvoiceID` from the database, insert an Invoice into the database, and insert the `InvoiceLineItems` into the database. You will need the following methods.

1. `InsertInvoiceLineItem` – Insert a `InsertInvoiceLineItem` into the `InsertInvoiceLineItems` table in the database
2. `GetMaxID` – Gets the max `InvoiceID` from the database, which is the `InvoiceID` of the last inserted record

3. InsertInvoice – Inserts an invoice into the database. This requires the Invoice data to be inserted into the database, close the connection, get the max InvoiceID from the Invoices table, and loops to insert all InvoiceLineItem into the InsertInvoiceLineItems table in the database by repeatedly calling the InsertInvoiceLineItem method

Cart section on Form

Please add the following code to frmMain.

1. Add code to instantiate the reference for Invoice i in class level data.
2. Add code to assign -1 to the Invoice i in class level data to record that there currently is no Customer in reference c from class level data
3. Add code to clear Invoice i.
4. Add code to copy CustomerID from Customer c to Invoice i
5. Add code to copy Invoice i CustomerID from maxID
6. Add code to set Invoice i CustomerID to -1 to record that there currently is no Customer in reference c
7. Add code to loop through the items in IstCart to see if the currently selected Product p is in the cart
8. Add condition to check if previous conversion succeeded, cart item does not have more books than are in stock and Product is not already in Cart
9. Add code to calculate cost of all items purchased for this Product (Get UnitPrice from p and qty from TryParse)
10. Add code to instantiate class reference InvoiceLineItem li with the selected Product from the Products section. See comment for what data to pass to the constructor.
11. Add code to add InvoiceLineItem li to Invoice i
12. Add code to add InvoiceLineItem li to IstCart
13. Add code to call UpdateInvoiceLineItem in i to pass new values for IstCart selected index
14. Add code to delete Product from Invoice i by using IstCart SelectedIndex. The List of InvoiceLineItem and IstCart Items are parallel arrays.
15. Add code to clear Invoice i
16. Add code to check if a Customer has been found or added. If CustomerID is less than zero, there is no Customer found or added.
17. Add code to insert an Invoice i into the database
18. Add code to copy the data from the ProductTotal, Shipping, SalesTax and InvoiceTotal from Invoice i to the TextBoxes in the Cart section