Oracle AppWizard for Microsoft Visual C++ User's Guide
Release 8.1.6
A73028-01

4 Tutorial

This chapter guides you through the creation and customization of an application generated by Oracle AppWizard for Microsoft Visual C++. All of the customization lessons presented in this tutorial are based on information described in Chapter 3, "Understanding Your Application's Code" This tutorial introduces each lesson with an explanation of what you will learn upon completion.

By working through all the lessons of this tutorial, you will learn how to create and customize an Oracle AppWizard for Microsoft Visual C++ starter application.

Specific topics discussed are:

Introduction

A store owner must keep accurate records of all customers and their purchases. A database application is perfect for performing this job.

In this tutorial, you develop a custom purchase order application for Nicole's Sporting Goods called "Order." This business must track information about sales orders, the items sold, and to which of the store's customers. You create and examine the structure of the database tables, then create and customize the application to manage that information. The application you develop allows Nicole and her employees to create, update, and view purchase orders for her sporting goods store.

Each main section of the tutorial corresponds to a version number for Order, with each successive lesson building on what was learned in the last one. When you finish working through the final lesson, you will have programmed a completely customized purchase order system for Nichole's Sporting Goods.

You can access the files for the tutorial in the following directory:

*ORACLE_BASE**ORACLE_HOME*\APPWIZARD\VC++\TUTORIAL\ORDER[1-5]

These files show you what you should have built by the end of each lesson. The Order1 file corresponds to Lesson 1, Order2 corresponds to Lesson 2, and so on. Therefore, to see the set of completed files for Lesson 1, access:

*ORACLE_BASE**ORACLE_HOME*\APPWIZARD\VC++\TUTORIAL\ORDER1

Before You Start

Before you can create any application, you must set up your database tables, so that the application has information to process. Because Nicole needs to keep track of information about sales orders, the items sold, the store's customers, and the products sold, you must structure the tables to reflect those needs.

To create this purchase order system, you must set up the following database tables to interact with each other:

- Sales_Order
- Item
- Customer
- Product

If you need to create the user DEMO and the above tables, search for DEMO.SQL in the *ORACLE_BASE**ORACLE_HOME*\RDBMS\ADMIN directory and execute it within Server Manager or SQL*Plus. The DEMO.SQL file creates user DEMO and calls the SQL script BDEMOBLD.SQL to create the above tables for you.

The following illustrates the data models for the tables you will construct:

Sales_Order table:

```
Name                           Null?    Type
------------------------------ -------- ----
ORDER_ID                       NOT NULL NUMBER(4)
ORDER_DATE                              DATE
CUSTOMER_ID                             NUMBER(6)
SHIP_DATE                               DATE
```

TOTAL NUMBER(8,2)

Item table:

```
Name                           Null?    Type
------------------------------ -------- ----
ORDER_ID                       NOT NULL NUMBER(4)
ITEM_ID                        NOT NULL NUMBER(4)
PRODUCT_ID                              NUMBER(6)
ACTUAL_PRICE                            NUMBER(8,2)
QUANTITY                                NUMBER(8)
TOTAL                                   NUMBER(8,2)
```

Customer table:

```
Name                           Null?    Type
------------------------------ -------- ----
CUSTOMER_ID                    NOT NULL NUMBER(6)
NAME                                    VARCHAR2(45)
ADDRESS                                 VARCHAR2(40)
CITY                                    VARCHAR2(30)
STATE                                   VARCHAR2(2)
ZIP_CODE                                VARCHAR2(9)
AREA_CODE                               NUMBER(3)
PHONE_NUMBER                            NUMBER(7)
```

```
SALESPERSON_ID                          NUMBER(4)
CREDIT_LIMIT                            NUMBER(9,2)
COMMENTS                                LONG
```

Product table:

```
Name                           Null?    Type
------------------------------ -------- ----
PRODUCT_ID                     NOT NULL NUMBER(6)
DESCRIPTION                             VARCHAR2(30)
```

Lesson 1: Creating the Starter Application

In Lesson 1, you form the basis of the purchase order application for Nicole's Sporting Goods. In this lesson, you learn how to quickly complete a starter application as explained in Chapter 2, "Creating a Starter Application" The application you create using Oracle AppWizard for Microsoft Visual C++ is then ready for customization.

When you are finished with this section, you will understand how to use Oracle AppWizard for Microsoft Visual C++ to create a starter application.

Lesson 1 consists of the following parts:

- Part 1: Working with Oracle AppWizard for Microsoft Visual C++
- Part 2: Exploring Generated Classes and Files
- Part 3: Viewing the ReadMe.txt for the Generated Project
- Part 4: Building and Running the Application

Part 1: Working with Oracle AppWizard for Microsoft Visual C++

Starting the Oracle AppWizard for Microsoft Visual C++

1. Start Microsoft Visual C++.
2. Choose New from the File menu.

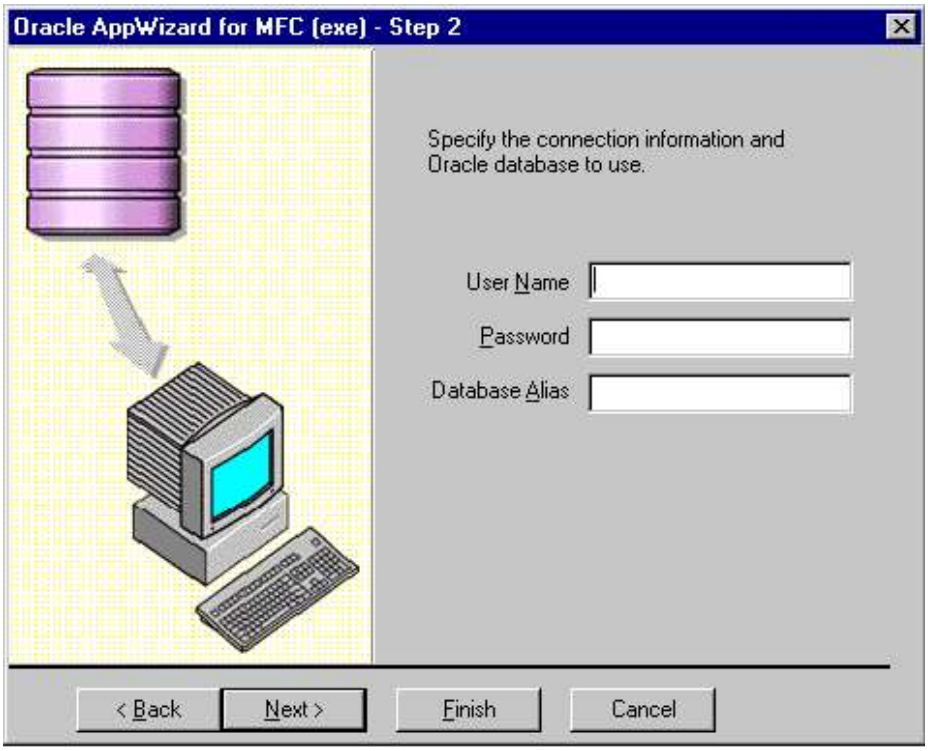
   The *New* dialog box appears.
3. Click the Projects tab.
4. Select Oracle AppWizard for MFC (exe) from the list of project types.
5. Enter the path from which you want to locate the application.
6. Enter "Order" in the Project Name box. This is the name of the application that you will create.
7. Make sure that "Create New Workspace" is selected.
8. Click OK.

   Oracle AppWizard for Microsoft Visual C++ starts.

   The *Welcome* window appears.
9. Click Next to continue.

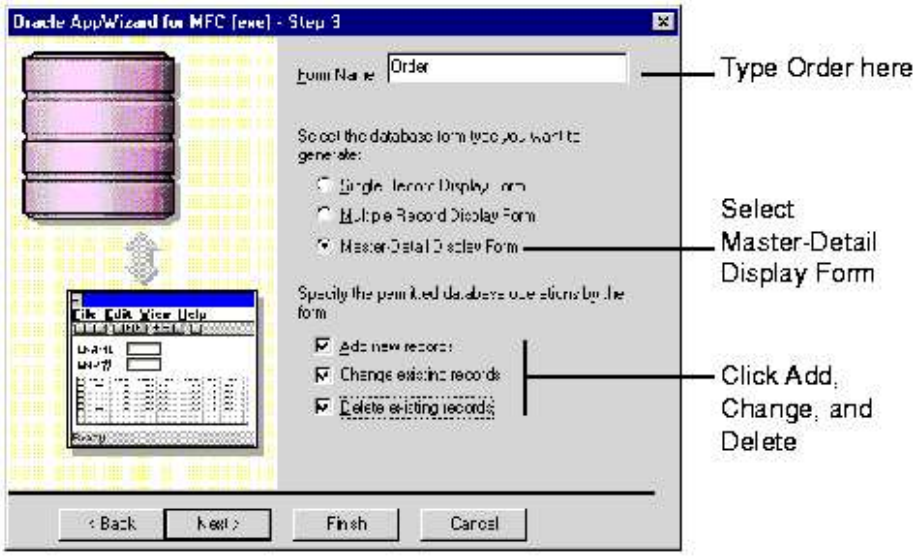
Connecting to the Oracle Database

1. Enter DEMO in the User Name text field.
2. Enter DEMO in the Password text field.
3. If connecting to a remote database, type the database alias in the Database Alias box. Otherwise, leave this field empty.
4. When you are done, click Next.



Naming and Specifying the Type of Form

1. Enter the name of the form: Order.
2. Select Master-Detail Display Form.
3. Click the Add, Change, and Delete checkboxes.
4. Click Next.

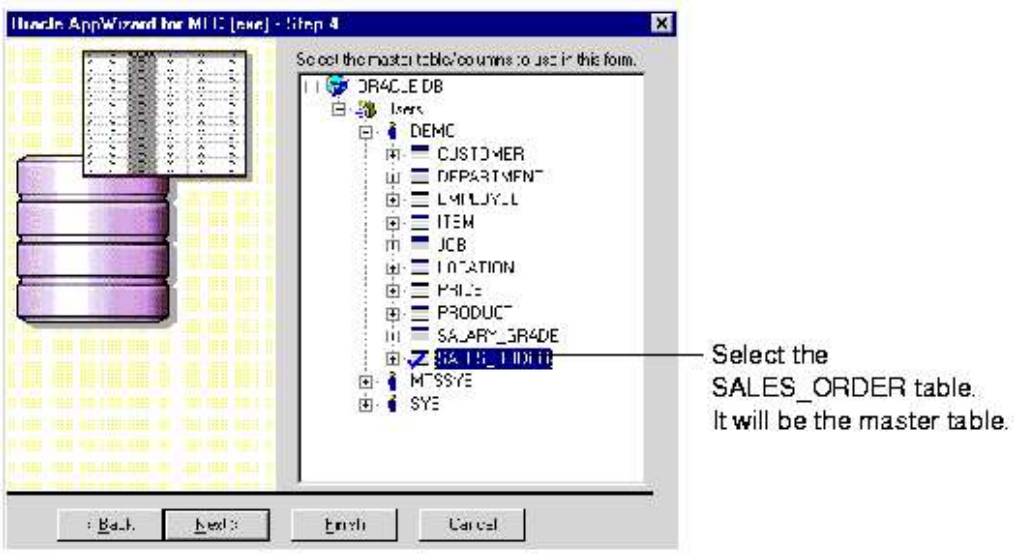


Selecting Master Tables and Columns

1. Select the SALES_ORDER table under the DEMO user in the list. This becomes the master table.

   By default, all columns under SALES_ORDER are selected.

2. Accept the default and click Next.

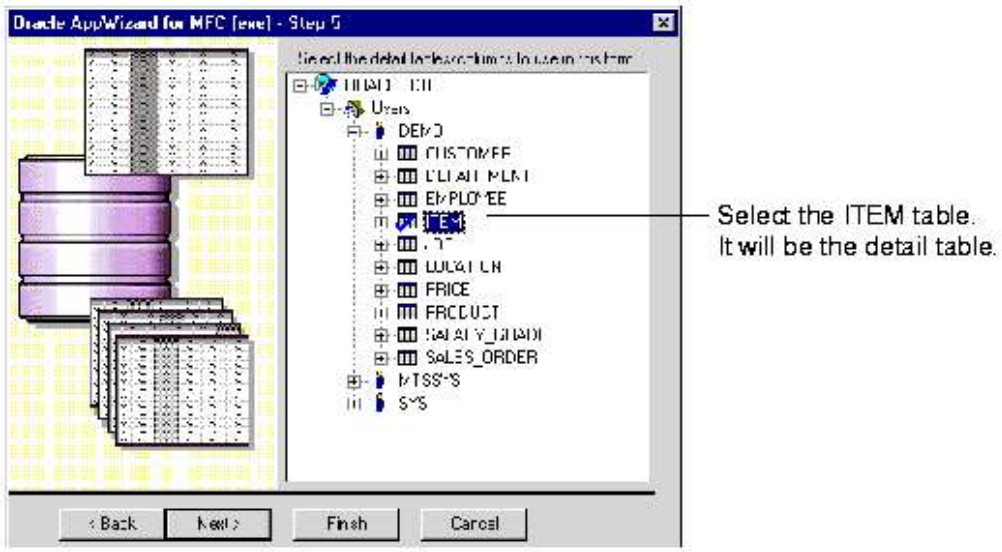


Selecting Detail Tables and Columns

1. Select the ITEM table under the DEMO user in the list. This will be the detail table.

   By default, all columns under ITEM are selected.

2. Accept the default and click Next.

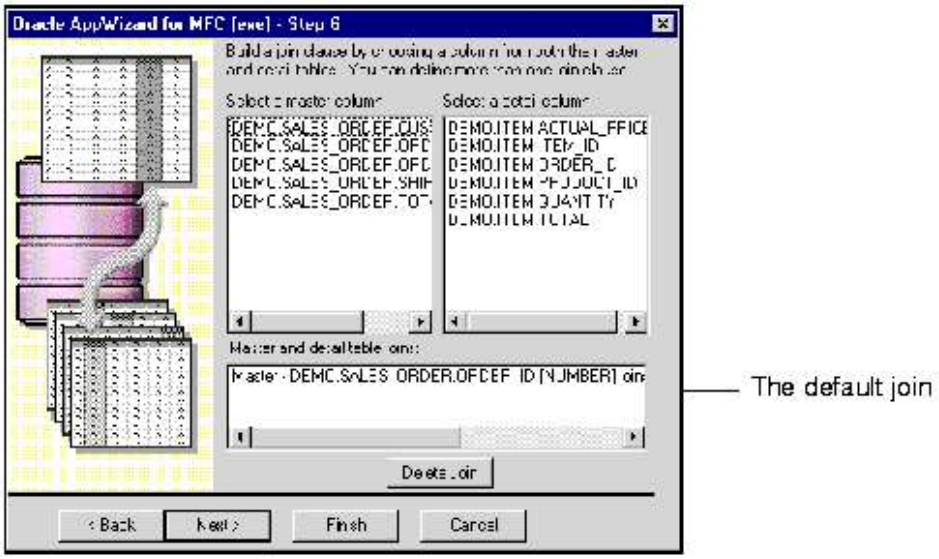


Building Join Clauses Between Tables

The next window displays the columns from the master and the detail tables you have selected and the default join clause that Oracle AppWizard has created. Oracle AppWizard creates the default join it bases on the primary and foreign keys of the master and detail tables.

In our example, Oracle AppWizard has created a default join, based on the primary and foreign keys:

```
Sales.Order.Order_ID joined with Item.Order_ID
```



3. Click Next to use the default join.

Oracle AppWizard prompts you:

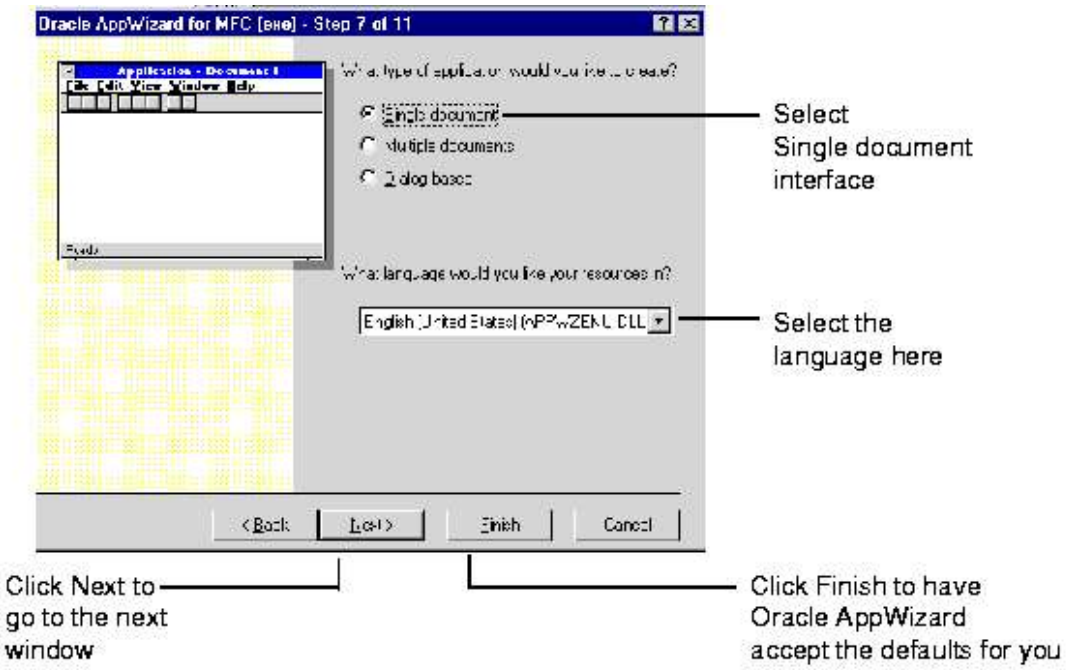
Would you like to create another form?

4. Click No.

Specifying the Application Type and User Language

1. Select Single document.

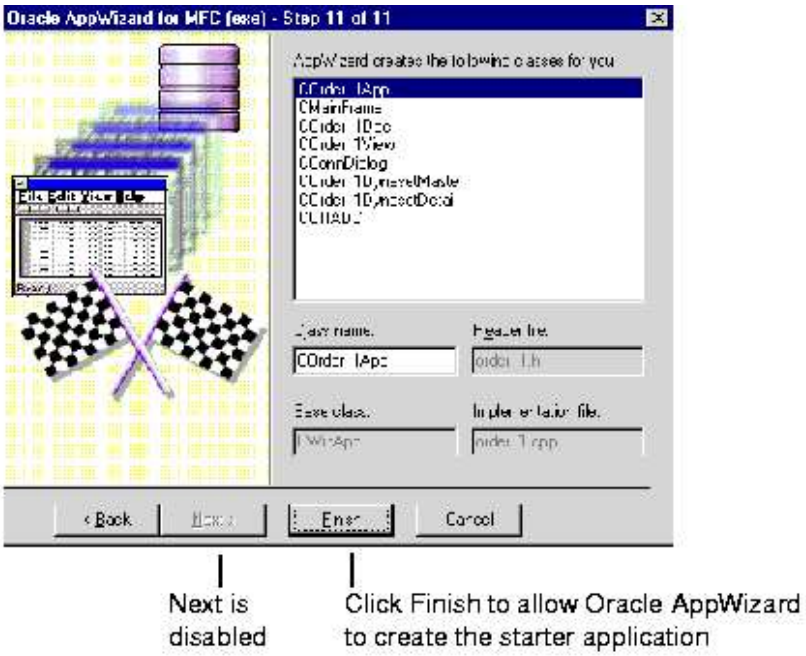
2. Accept the default for Document/View Architecture Support.
3. Accept the current default language. In this tutorial, we use English [United States].
4. Click Next.



Completing the Remaining Steps

1. Accept the default options for the remaining steps by clicking Next on each window. You can also click Finish and Oracle AppWizard for Microsoft Visual C++ will accept the defaults for you.

   In the last window, the Next button is disabled, as shown in the following diagram.

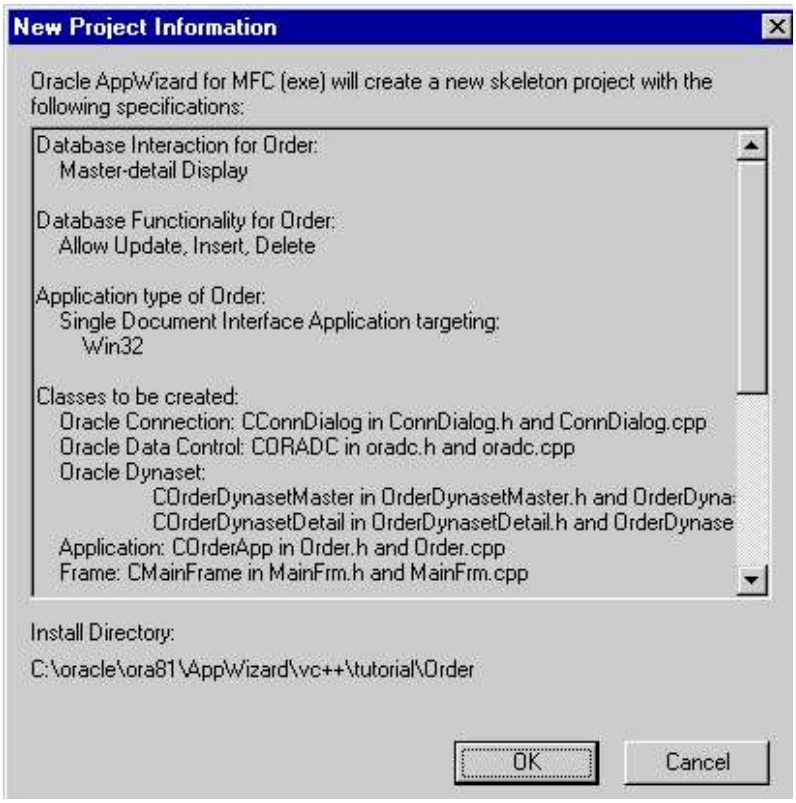


In this window, Oracle AppWizard for Microsoft Visual C++ displays the classes it creates for your application.

2. Click Finish to allow Oracle AppWizard for Microsoft Visual C++ to create the starter application.

   The *New Project Information* window appears. This window shows you the specifications that Oracle AppWizard used to create the Order application.

3. Examine the information in the *New Project Information* window for accuracy.



4. When done, click OK.

Oracle AppWizard for Microsoft Visual C++ creates the source, header, and resource files for your Order application and takes you automatically to the Microsoft Developer Studio with Workspace "Order" open.

Part 2: Exploring Generated Classes and Files

Now that you have created the Order application using Oracle AppWizard for Microsoft Visual C++, you can customize it to better suit your needs. To customize this application you need to know about the classes and files created for this purpose, as shown in the following table:

Files	Class	Description	Derived From...
AboutDlg.cpp AboutDlg.h	CAboutDlg	dialog class for *About Order* dialog box	CDialog
ConnDialog.cpp ConnDialog.h	CConnDialog	dialog class for *Connect to Oracle* dialog box	CDialog
MainFrame.cpp MainFrame.h	CMainFrame	frame window class	CFrameWnd
ORADC.cpp ORADC.h	CORADC	data control class	CWnd

Files	Class	Description	Derived From...
OrderApp.cpp OrderApp.h	COrderApp	application class	CWinApp
OrderDynasetDetail.cpp OrderDynasetDetail.h	COrderDynasetDetail	dynaset class for Sales Order table	ODynaset
OrderDynasetMaster.cpp OrderDynasetMaster.h	COrderDynasetMaster	dynaset class for Item table	ODynaset
OrderView.cpp OrderView.h	COrderView	form view class	CFormView
OrderDoc.cpp OrderDoc.h	COrderDoc	document class	CDocument
OrderUtil.cpp OrderUtil.h	Not applicable	stand alone functions: Process OO4OError(), DDX_Field Text(), ...	

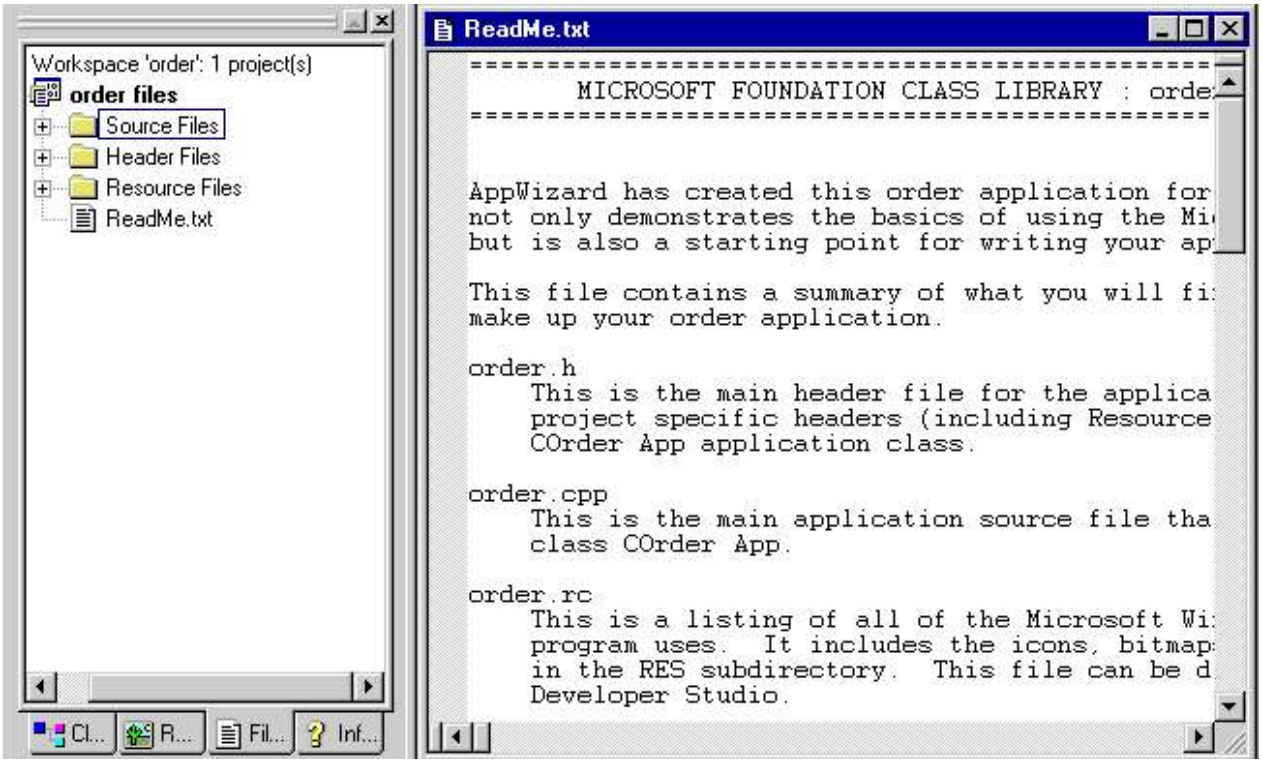
Part 3: Viewing the ReadMe.txt for the Generated Project

Oracle AppWizard for Microsoft Visual C++ creates a ReadMe.txt file for the generated Order project. This ReadMe.txt file describes the source, header, and resource files that Oracle AppWizard has created.

To view the contents of the ReadMe.txt file:

1. Select the FileView tab.
2. Click the name of your project, Order, to open it in the Workspace of the dialog box.
3. Click the ReadMe.txt file to open it.

   The following diagram shows what the ReadMe.txt file for Order should look like:



Part 4: Building and Running the Application

In this section you use Microsoft Developer Studio to build and run the executable files created by Oracle AppWizard for Microsoft Visual C++.

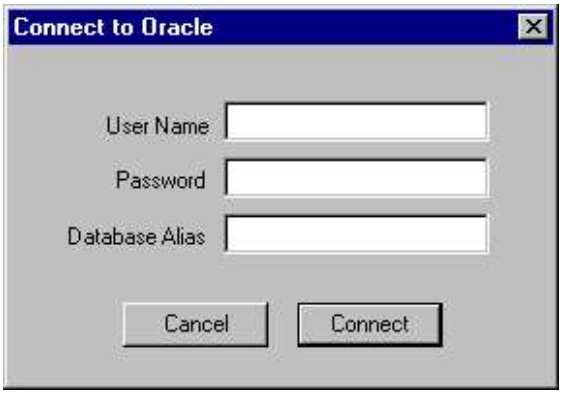
Note:

By default, the active project configuration is the debug version. To change the active project configuration, choose Set Active Configuration from the Build menu.

---

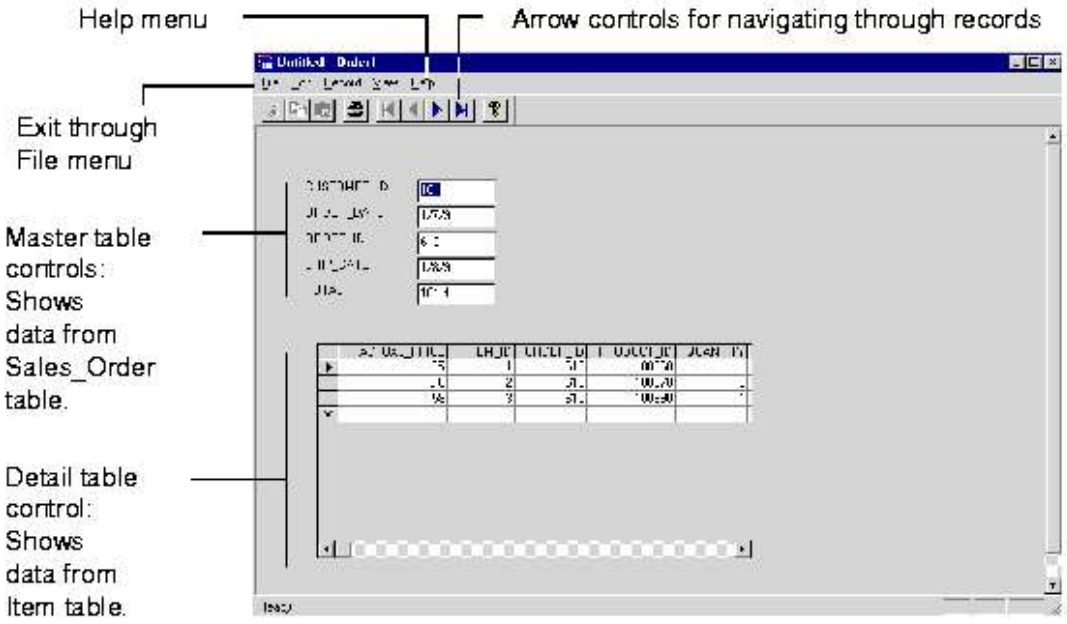
1. Choose Build Order.exe from the Build menu to build the application.
2. Choose Execute Order.exe from the Build menu to run the application.

   The *Connect to Oracle* dialog box appears.



3. Enter DEMO in the User Name text field and DEMO in the Password text field. If connecting to a remote database, type the database alias in the Database Alias text field.
4. Click Connect.

   The Order application appears.



Notice and try using the following features:

- Arrow controls

  These controls on the toolbar let you easily navigate through the records.

- Record menu

  Another method for navigating through the records.

- View menu

  Shows or hides the toolbar or status bar.

- Help > About Order

  This displays the *About Order* window. (Clicking OK closes the window.)

- File > Exit

  This exits the application.

Lesson 2: Adding Customer Information to a Purchase Order

In this first customization task, you learn how to add specific, detailed information about customers that should appear on any purchase order for the Order application. To accomplish this task, complete the following parts of this lesson:

- Part 1: Creating a Dynaset Class for the Customer Table
- Part 2: Adding Customer Information to a Purchase Order
- Part 3: Displaying Customer Information for a Purchase Order

To see the set of completed files for Lesson 2, access:

*ORACLE_BASE**ORACLE_HOME*\APPWIZARD\VC++\TUTORIAL\ORDER2.

Note:

Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Creating a Dynaset Class for the Customer Table

To display customer information on your purchase order form for an active order, you must first create an ODynaset class to represent the CUSTOMER table. To do this, create a new generic class called COrderCustomerDynaset, which is derived from the ODynaset class.

The ODynaset class is a class in the Oracle Objects for OLE C++ class library. It creates, manages, and accesses data records from the database.

To create a dynaset for the CUSTOMER table:

1. Right-click Order class and select New Class in the Class view of Microsoft Developer Studio.

The *New Class* dialog box appears.

2. Select Generic Class from the Class Type list.

3. Enter the following in the Name text box under Class Information:

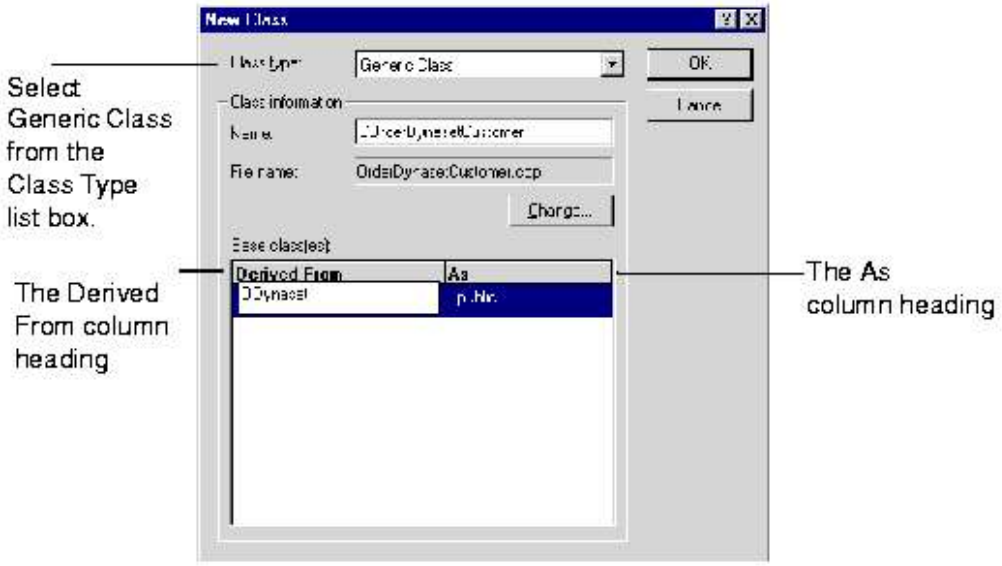
```
COrderDynasetCustomer
```

4. Enter the following under the column heading Derived From in the Base Class(es) list:

```
ODynaset
```

5. Accept the default entry, "public" under the As column heading.

   This is how the *New Class* dialog box appears before you click OK.



6. Click OK.

   Microsoft Developer Studio displays the following message:

   The New Class Wizard could not find the appropriate header file(s) to include for the base class(es) ODynaset...

7. Click OK.

   This message occurs because the definition of the ODynaset class is not included in the generated file. Therefore, you must manually include the header file that defines the ODynaset class, ORACL.H, in the OrderDynaSetCustomer.h file. You can locate the header file in:

   *ORACLE_BASE**ORACLE_HOME*\0040\cpp\include

   a *ORACLE_HOME*\0040\cpp\include

   Oracle AppWizard for Microsoft Visual C++ generates both an OrderDynasetCustomer.cpp file and OrderDynasetCustomer.h file for the COrderDynasetCustomer class with the following contents:

   The OrderDynasetCustomer.cpp file contains:

```
//OrderDynasetCustomer.cpp: implmentation of the COrderDynasetCustomer
class.
//
//////////////////////////////////////////////////////
#include "stdafcx.h"
```

```
#include "Order.h"
#include "OrderDynasetCustomer.h"
COrderDynasetCustomer:COrderDynasetCustomer()
{

}
COrderDynasetCustomer:~COrderDynasetCustomer()
{

}
```

The OrderDynasetCustomer.h file contains:

```
//OrderDynasetCustomer.h: interface for the COrderDynasetCustomer Class//
//////////////////////////////////////////////////////////
class COrderDynasetCustomer : public ODynaset
{
public:
COrderDynasetCustomer();
};
```

Adding Member Variables to the COrderDynasetCustomer Class

COrderDynasetCustomer class is derived from the ODynaset class. This class contains member variables that store information about a SQL statement to query the database. This class also stores information to represent columns in the USER.DEMO.CUSTOMER table. These are shown below:

Member Variables	Type	Type Description
m_strSQLQuery	CString	Query to be sent to the database
m_strSQLSelect	CString	Select clause of the SQL statement
m_strSQLFilter	CString	Where clause of the SQL statement
m_strSQLSort	CString	Order clause of the SQL statement
m_Column1	OField	CUSTOMER_ID column of CUSTOMER table
m_NAME	OField	NAME column of CUSTOMER table
m_ADDRESS	OField	ADDRESS column of CUSTOMER table
m_CITY	OField	CITY column of CUSTOMER table
m_STATE	OField	STATE column of CUSTOMER table
m_ZIP_CODE	OField	ZIP column of CUSTOMER table
m_AREA_CODE;	OField	AREA_CODE column of CUSTOMER table
m_PHONE_NUMBER	OField	TELEPHONE_NUMBER column of CUSTOMER table
m_SALESPERSON_ID	OField	SALESPERSON_ID column of CUSTOMER table
m_CREDIT_LIMIT	OField	CREDIT_LIMIT column of CUSTOMER table
m_COMMENTS	OField	COMMENTS column of CUSTOMER table

To add member variables for the COrderDynasetCustomer class:

1. Enter the following code in the COrderDynasetCustomer.h file:

```
class COrderDynasetCustomer : public ODynaset
{
...
public:
    COrderDynasetCustomer();
    virtual ~COrderDynasetCustomer();
```

```
public:
// strings needed for creating the queries
   CString m_strSQLQuery;  // the query to be sent to the database
   CString m_strSQLSelect; // the select portion
   CString m_strSQLFilter; // the where portion
   CString m_strSQLSort;   // the order by portion

// Field/Param Data



   OField m_CUSTOMER_ID;
   OField m_NAME;
   OField m_ADDRESS;
   OField m_CITY;
   OField m_STATE;
   OField m_ZIP_CODE;
   OField m_AREA_CODE;
   OField m_PHONE_NUMBER;
   OField m_SALESPERSON_ID;
   OField m_CREDIT_LIMIT;
   OField m_COMMENTS;



...
};
```

Adding Member Functions and Implementation Details to the COrderDynasetCustomer Class

Adding the member functions shown below to the COrderDynaset customer class enables you to create a query statement, have the database process it, then retrieve and store information.

Member Functions	Descriptions
OpenQuery	Starts a query and retrieves the information from the database.
CreateSQLSelect	Creates the default query (SELECT) statement.
AddFilter	Adds a condition to the query statement.
ResetToDefaultFilter	Resets the condition clause for the query.
RefreshQuery	Refreshes the query to be executed.

To declare member functions for the COrderDynasetCustomer class:

1. Add the following code to the OrderDynasetCustomer.h file:

```
class COrderDynasetCustomer : public ODynaset
{
...

    public:
            COrderDynasetCustomer();
            virtual ~COrderDynasetCustomer();
```

```
// Operations
public:
void OpenQuery(ODatabase theDB);
void CreateSQLSelect();
void AddFilter(CString strFilter);
void ResetToDefaultFilter();
void RefreshQuery();
 ...


  };
```

2. Add the following code to the constructor method of COrderDynasetCustomer Class in the OrderDynasetCustomer.cpp file to initialize the value in the member variables:

```
COrderDynasetCustomer::COrderDynasetCustomer()
{

m_strSQLQuery.Empty();
m_strSQLSelect.Empty();
m_strSQLFilter.Empty();
m_strSQLSort.Empty();



// Create the default select clause of the

statement m_strSQLSelect


  CreateSQLSelect();

}
```

3. Add the following code for the COrderDynasetCustomer class::*CreateSQLSelect()* method at the end of the OrderDynasetCustomer.cpp file to create the default SQL statement:

```
// Creates the default select
void COrderDynasetCustomer::CreateSQLSelect()

{

m_strSQLSelect = "select CUSTOMER_ID, NAME,
    ADDRESS, CITY, STATE, \
    ZIP_CODE, AREA_CODE, PHONE_NUMBER, \
    SALESPERSON_ID, CREDIT_LIMIT, COMMENTS \
    from DEMO.CUSTOMER";

}
```

4. Add the following code for the COrderDynasetCustomer class::*OpenQuery()* method in the OrderDynasetCustomer.cpp file to:

- Create a query statement
- Process the query
- Retrieve information from the database

```
// opens the query
void COrderDynasetCustomer::OpenQuery(ODatabase theDB)
{
```

```
oresult dbresult;
// create a query statement

m_strSQLQuery = m_strSQLSelect;
if (m_strSQLFilter)
      m_strSQLQuery += m_strSQLFilter;
if (m_strSQLSort)
      m_strSQLQuery += m_strSQLSort;


// query the database

dbresult = Open(theDB, m_strSQLQuery);


// retreive/store information from the database

m_CUSTOMER_ID = GetField("CUSTOMER_ID");
m_NAME = GetField("NAME");
m_ADDRESS = GetField("ADDRESS");
m_CITY = GetField("CITY");
m_STATE = GetField("STATE");
m_ZIP_CODE = GetField("ZIP_CODE");
m_AREA_CODE = GetField("AREA_CODE");
m_PHONE_NUMBER = GetField("PHONE_NUMBER");
m_SALESPERSON_ID = GetField("SALESPERSON_ID");
m_CREDIT_LIMIT = GetField("CREDIT_LIMIT");
m_COMMENTS = GetField("COMMENTS");


// display the first record from the Customer table

dbresult = MoveFirst();

}
```

5. Add the following code for the COrderDynasetCustomer class::*AddFilter()* method in the OrderDynasetCustomer.cpp file to add a conditional clause for the query:

```
// adds a condition to the where clause

void COrderDynasetCustomer::AddFilter(CString strFilter)
{

m_strSQLFilter += m_strSQLFilter.IsEmpty() ? "
    WHERE " : " AND ";
m_strSQLFilter += strFilter;

}
```

6. Add the following code for the COrderDynasetCustomer class::*ResetToDefaultFilter()* method in the OrderDynasetCustomer.cpp file to reset the conditional clause for the query:

```
// resets the filter to the default value
void COrderDynasetCustomer::ResetToDefaultFilter()
{

  m_strSQLFilter.Empty();

}
```

7. Add the following code for COrderDynasetCustomer class::*RefreshQuery()* method in the OrderDynasetCustomer.cpp file to refresh the query to be executed:

```
void COrderDynasetCustomer::RefreshQuery()
{
```

```
oresult dbresult;


// create a query statement

m_strSQLQuery = m_strSQLSelect;
if (m_strSQLFilter)
    m_strSQLQuery += m_strSQLFilter;
if (m_strSQLSort)
    m_strSQLQuery += m_strSQLSort;


// set the query statement to be used

dbresult = SetSQL(m_strSQLQuery);
dbresult = Refresh();

}
```

Part 2: Adding Customer Information to a Purchase Order

To display customer information for a purchase order, add the customer dynaset to the COrderView class and COrderDoc class.

To add customer information to a purchase order:

1. Add a member variable called m_pDynasetCustomer to the COrderView class in the COrderView.h header file. This member variable is a pointer to the type COrderCustomerDynaset class.

```
class COrderView : public CFormView
{

...
COrderDynasetMaster *m_pDynasetMaster;
COrderDynasetDetail *m_pDynasetDetail;
COrderDynasetCustomer *m_pDynasetCustomer;
...

}
```

2. Initialize the variable, m_pDynasetCustomer, to NULL in the constructor method of COrderView class in the the COrderView.cpp file.

```
COrderView::COrderView()

: CFormView(COrderView::IDD)

{

...
m_pDynasetMaster = NULL;
m_pDynasetDetail = NULL;
m_pDynasetCustomer = NULL;
...

}
```

3. Add a member variable called m_OrderDynasetCustomer to the COrderDoc class in the COrderCustomerDynaset.cpp file. This is an object of COrderCustomerDynaset class.

```
class COrderDoc : public CDocument
{
```

```
...
COrderDynasetMaster m_OrderDynasetMaster;
COrderDynasetDetail m_OrderDynasetDetail;
COrderDynasetCustomer m_OrderDynasetCustomer;
...

}
```

4. Include the OrderDynasetCustomer.h header file in Order.cpp, OrderDoc.h, and OrderView.h to define the COrderDynasetCustomer class.

Part 3: Displaying Customer Information for a Purchase Order

This section demonstrates how to display customer information for each purchase order.

1. Add 10 edit controls to display customer information in the IDD_ORDER_FORM dialog box with the Control IDs as shown in the following table:

Table 4-1 Control IDs with Member Variables

Control ID	Member Variables for COrderDynasetCustomer Class
IDC_CUST_NAME	m_NAME
IDC_CUST_ADDRESS	m_ADDRESS
IDC_CUST_CITY	m_CITY
IDC_CUST_STATE	m_STATE
IDC_CUST_ZIP_CODE	m_ZIP_CODE
IDC_CUST_AREA_CODE	m_AREA_CODE
IDC_CUST_PHONE_NUMBER	m_PHONE_NUMBER
IDC_CUST_SALESPERSON_ID	m_SALESPERSON_ID
IDC_CUST_CREDIT_LIMIT	m_CREDIT_LIMIT
IDC_CUST_COMMENTS	m_COMMENTS

After you add these edit controls, the IDD_ORDER_FORM dialog box appears similar to the one shown below. This illustration serves only as a sample. Yours may look slightly different.

To bind these edit controls with the OField members in the COrderDynasetCustomer class, call the *DDX_FieldText()* method for each customer edit control in the COrderView class::*DoDataExchange()* method,

as described in the next step.

The OField members represent the columns in the Customer table. Bind these with a corresponding OField member variable of the COrderDynasetCustomer class shown in Table 4-1.

2. Enter the following code in COrderView:DoData Exchange in the OrderView.cpp file to bind the customer edit controls with the OField members in the COrderDyaset Customer class:

```
void COrderView::DoDataExchange(CDataExchange* pDX)
{

...
CFormView::DoDataExchange(pDX);
...
// for customer dynaset
DDX_FieldText(pDX, IDC_CUST_NAME,

m_pDynasetCustomer->m_NAME, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_ADDRESS,

m_pDynasetCustomer->m_ADDRESS, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_CITY, m_pDynasetCustomer->m_CITY,

m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_STATE,

m_pDynasetCustomer->m_STATE, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_ZIP_CODE,

m_pDynasetCustomer->m_ZIP_CODE, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_AREA_CODE,

m_pDynasetCustomer->m_AREA_CODE, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_PHONE_NUMBER,

m_pDynasetCustomer->m_PHONE_NUMBER, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_CREDIT_LIMIT,

m_pDynasetCustomer->m_CREDIT_LIMIT, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_SALESPERSON_ID,

m_pDynasetCustomer->m_SALESPERSON_ID, m_pDynasetCustomer);

DDX_FieldText(pDX, IDC_CUST_COMMENTS,

m_pDynasetCustomer->m_COMMENTS, m_pDynasetCustomer);

...

}
```

3. Add the following code to the COrderView::*OnInitialUpdate()* method in OrderView.cpp to initialize the customer dynaset when the view is attached to the document:

```
void COrderView::OnInitialUpdate()
{

CString strJoin;
...
m_pDynasetMaster = &GetDocument()->m_OrderDynasetMaster;
m_pDynasetDetail = &GetDocument()->m_OrderDynasetDetail;
```

```
m_pDynasetCustomer = &GetDocument()->m_OrderDynasetCustomer;

m_pDynasetMaster->OpenQuery(GetDocument()->m_database);

}
```

4. Add the following code to the COrderView::*OnInitialUpdate()* method in OrderView.cpp to perform a join operation between the CUSTOMER_ID column of the SALES_ORDER table and the CUSTOMER_ID column of the ITEM table to retrieve customer information for a purchase order:

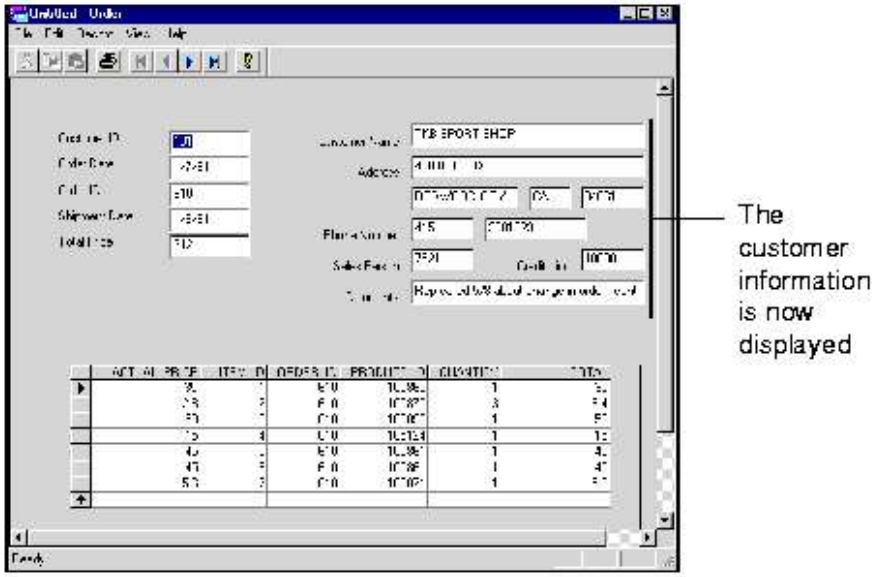
```
...
m_pDynasetMaster->OpenQuery(GetDocument()->m_database);

// create a join based on CUSTOMER_ID
// between the Customer table and
// the Sales.Order table.
m_pDynasetCustomer->ResetToDefaultFilter();
strJoin = "CUSTOMER_ID = " +

(CString)m_pDynasetMaster->m_Column1;

m_pDynasetCustomer->AddFilter(strJoin);
// create, process the query
m_pDynasetCustomer->OpenQuery(GetDocument()->m_database);
...

CFormView::OnInitialUpdate();

m_dataControl.SetRecordset((LPDISPATCH)(m_pDynasetDetail->Internal()));
...

}
```

5. Enter the following code to the *PerformMove()* method of the COrderView class in OrderView.cpp to update the customer dynaset after a move operation has been performed:

```
void COrderView::PerformMove(int nCommand)
{

...
// Update Customer information
m_pDynasetCustomer->ResetToDefaultFilter();
strJoin = "CUSTOMER_ID = " +

(CString)m_pDynasetMaster->m_Column1;

m_pDynasetCustomer->AddFilter(strJoin);
m_pDynasetCustomer->RefreshQuery();


UpdateData(FALSE);

}
```

6. Build and run the application.

This is how the application should look when you are done:



Lesson 3: Enabling Users to Add Products to a Purchase Order

Nicole's employees must add products to a purchase order. This lesson demonstrates how to customize your application to:

- Display available products from the PRODUCT table
- Add product items to a purchase order
- Update the changes in the ITEM table and the detail table control when product items have been added to a purchase order
- Update the changes in the SALES_ORDER table and the master table controls when product items have been added to a purchase order

Lesson 3 contains the following parts:

- Part 1: Displaying a List of Items from the PRODUCT Table
- Part 2: Adding the Selected Products to the Purchase Order

To see the set of completed files for Lesson 3, access:

ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER3.

Note:

Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Displaying a List of Items from the PRODUCT Table

Nicole's Sporting Goods carries a wide variety of products. Many of them have similar names and descriptions. To insure data integrity, display a list of the product items in the purchase order application window. This enables Nicole's employees to select the items that customers have purchased from the list.

In this part, you will add a list box that displays the available products in the *IDD_ORDER_FORM* dialog box.

To add a list box:

1. Add a list box with a control ID = IDC_PRODUCTLIST that allows multiple selections. This enables users to select multiple purchased items, which can then be added to a purchase order, all at one time.

2. Create a new member variable, m_prodList, for the list box in COrderView class. The variable m_prodList is a control handler of type CListBox. It is used to store the list of product items from the PRODUCT table. Bind the IDC_PRODUCTLIST list box control with this new member variable.

3. Add the following code to COrderView::*OnInitialUpdate()* method in the OrderView.cpp file to display a list of available products from the PRODUCT table in the list box:

```
void COrderView::OnInitialUpdate()
{

...
CFormView::OnInitialUpdate()

...

// Put list of sales items into the listbox
ODynaset oProdList;
oProdList.Open(GetDocument()->m_database,
     "SELECT PRODUCT_ID,DESCRIPTION
     FROM DEMO.PRODUCT ORDER BY PRODUCT_ID");
int index = 0;
while (!oProdList.IsEOF())

{

   m_prodList.InsertString(index,
       (CString)oProdList.GetField(0) + " - " +
        oProdList.GetField(1));



m_prodList.SetItemData(index,
       (long)oProdList.GetField(0));
oProdList.MoveNext();
index++;

}

}
```

Part 2: Adding the Selected Products to the Purchase Order

When Nicole's employees add a purchased item from the product list box to a purchase order, a new record for the purchased item is inserted into the ITEM table. Nicole wants the TOTAL column in the SALES_ORDER table to be re-calculated to reflect the amount added from the purchased item to the purchase order. Content in the detail table control and the master table control should also be updated accordingly.

1. To have the TOTAL column in the SALES_ORDER table updated when a purchased item is added to a purchase order, the SALES_ORDER table should be updated. To allow the updating of the SALES_ORDER table, remove the third parameter, ODYNASET_READONLY, from the Open() function call in the COrderDynasetMaster::*OpenQuery()* method:

Change:

```
dbresult = Open(theDB, m_strSQLQuery, ODYNASET_READONLY);
```

To:

```
dbresult = Open(theDB, m_strSQLQuery);
```

2. In the IDD_ORDER_FORM dialog, create a button that handles the event of adding a purchase item. The button requires the following properties and events:

Button Control ID	Button Caption	Event	Event Handler Function
IDC_ADDITEMS	Add to Order	BN_Clicked	OnAdditems

3. Add the following code to the COrderView::*OnAddItems()* method in the OrderView.cpp file to process the updates in the SALES _ORDER table, the ITEM table, the master table controls, and the detail table control when purchase items are added to a purchase order:

```
void COrderView::OnAdditems()
{

int *pAddIndices;
int nSelCount;
oresult r;
nSelCount = m_prodList.GetSelCount();
pAddIndices = new int[nSelCount];
m_prodList.GetSelItems(nSelCount, pAddIndices);
for (int i = 0; i < nSelCount; i++)
{

char szProdID[8];
char szItemID[8];
itoa(m_prodList.GetItemData(pAddIndices[i]), szProdID,
     10);
itoa(m_pDynasetDetail->GetRecordCount() + 1, szItemID,
     10);
(CString strInsert = "INSERT INTO DEMO.ITEM (ORDER_ID,
     PRODUCT_ID, ITEM_ID, QUANTITY,
     ACTUAL_PRICE, TOTAL) SELECT " +
       CString)m_pDynasetMaster->m_Column3 + ", "
     + szProdID +", " + szItemID +
       ", 1, LIST_PRICE, LIST_PRICE FROM
     DEMO.PRICE WHERE PRODUCT_ID = " +
     szProdID + " AND ((SYSDATE BETWEEN
     START_DATE AND END_DATE) OR " +
   "(SYSDATE > START_DATE AND END_DATE IS NULL))";
r = GetDocument()->m_database.ExecuteSQL(strInsert);
if (r == OFAILURE)
ProcessOO4OError(&GetDocument()->m_database);
m_pDynasetDetail->Refresh();

}
TotalOrder();
m_prodList.SelItemRange(FALSE, 0, m_prodList.GetCount());
delete pAddIndices;
}
```

The function TotalOrder() is being called in the COrderView::*OnAdditems()* method. It is a member function for COrderView class. It re-calculates the total amount for a purchase order and updates the TOTAL column in the SALES_ORDER table.

4. Enter the following code in the COrderView class of the OrderView.h file to declare the TotalOrder() in COrderView class:

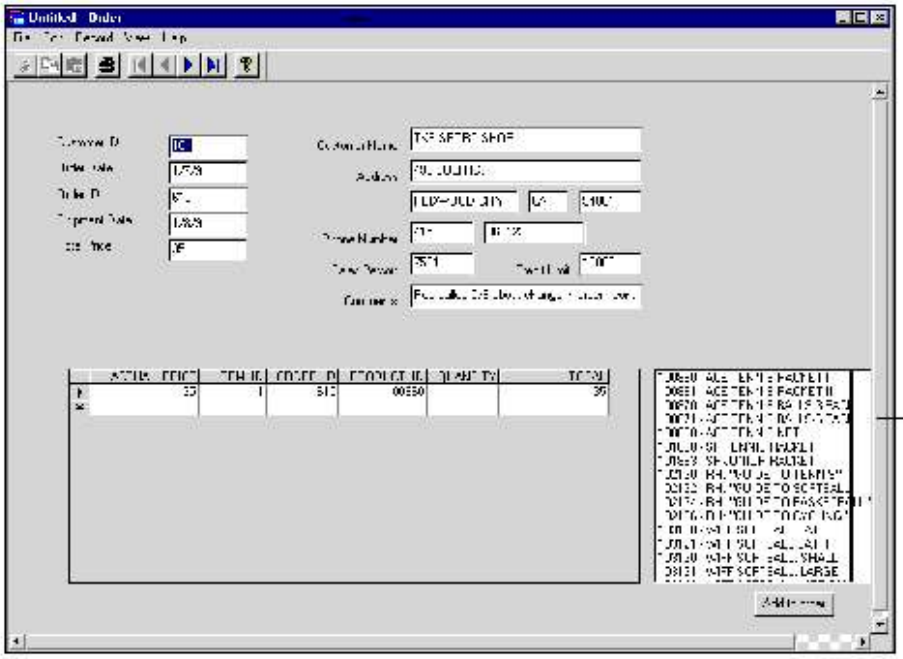
```
public:

void TotalOrder();
```

5. Add the following code for the COrderView::*TotalOrder()* method in the OrderView.cpp file:

```
void COrderView::TotalOrder()
{

ODynaset oTotal;
oTotal.Open(GetDocument()->m_database, "SELECT SUM(TOTAL) FROM

DEMO.ITEM WHERE ORDER_ID = " +
(CString)m_pDynasetMaster->m_Column3);

m_pDynasetMaster->StartEdit();
m_pDynasetMaster->

m_Column5.SetValue((double)oTotal.GetField(0));

m_pDynasetMaster->Update();
UpdateData(FALSE);

}
```

6. Build and run the application.

   The application will look similar to the following screen:



Lesson 4: Enabling Users to Update a Purchase Order

This section allows you to make changes in the detail table control for the Order application. When the value in the QUANTITY column or the ACTUAL_PRICE column in the detail table changes, the application should

update the appropriate changes to the ITEM table, the SALES_ORDER table, the content in the master detail controls, and the detail table control accordingly.

This section consists of the following parts:

- Part 1: Allowing the Detail Table Control to Handle Events
- Part 2: Adding Implementation Details to the Event Handler Function

To see the set of completed files for Lesson 4, access:

ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER4

.

Note:

Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Allowing the Detail Table Control to Handle Events

1. Modify the detail table control (ID = IDC_DATAGRID) in the IDD_ORDER_FORM Dialog that handles event types with specified event handler functions. These functions and handlers are listed in the following table.

Event Type	Event Handler
AfterColUpdate	OnAfterColUpdateDatagrid
AfterUpdate	OnAfterUpdateDatagrid
AfterDelete	OnAfterDeleteDatagrid

2. Create a member variable, m_datagrid, for the detail table control in COrderView class. The variable, m_datagrid, is a control handler of type CMsDgridCtrl. It represents the detail table and allows for update or retrieval of data.

Part 2: Adding Implementation Details to the Event Handler Function

When the Actual Price or the Quantity sold for a purchased item has been changed, the total price for this item should change. To have the application do this, additions must be made to the code.

1. Add the following code to the COrderView::*OnAfterColUpdateDatagrid()* method in the OrderView.cpp file to update the TOTAL column for the detail table control when the ACTUAL_PRICE column or the QUANTITY column in the detail table control changes:

```
void COrderView::OnAfterColUpdateDatagrid(short ColIndex)

// recalcuates the TOTAL column in the detail table when the
value
// in the ACTUAL_PRICE or QUANTITY column changes

{

        if (ColIndex == 4 || ColIndex == 0)
        {
```

```
                    CString strText = m_datagrid.GetText();
                    CString strTotal;
                    char *stopString;
                    char szTotal[15];
                    long quantity;
                    double price;

                    // QUANTITY column is updated
                    if (ColIndex == 4)
                    {
                      quantity = strtol(strText, &stopString, 10);
                      m_datagrid.SetCol(0);
                      strText = m_datagrid.GetText();
                      price = strtod(strText, &stopString);
                      m_datagrid.SetCol(4);
                    }
                    else // PRICE column is updated
                    {
                      price = strtod(strText, &stopString);
                      m_datagrid.SetCol(4);
                      strText = m_datagrid.GetText();
                      quantity = strtol(strText, &stopString, 10);
                      m_datagrid.SetCol(0);
                    }

                    sprintf(szTotal, "%f", quantity * price);
                    strTotal = szTotal;
                    m_datagrid.SetCol(5);
                    m_datagrid.SetText((LPCTSTR)strTotal);

            }

    }
```

2. Add the following code to the COrderView::*OnAfterUpdateDatagrid()* method in the OrderView.cpp file to update the TOTAL column for the master table controls when the ACTUAL_PRICE column or the QUANTITY column in the detail table change:

```
void COrderView::OnAfterUpdateDatagrid()
{

// Calculates the total amount for a purchase order
TotalOrder();

}
```

3. Add the following code to the COrderView::*OnAfterDeleteDatagrid()* method in the OrderView.cpp file to update the TOTAL column for the master table controls when the a record has been deleted from the detail table control:

```
void COrderView::OnAfterDeleteDatagrid()
{

// Calculates the total amount for a purchase order
TotalOrder();

}
```

Lesson 5: Enabling Users to Add, Commit, or Cancel a New Purchase Order

This lesson describes how to update the application to enable Nicole's employees to create a new purchase order for a customer. When you have completed this last customization, Nicole's employees can also commit or cancel the creation of a new purchase order.

This section consists of the following parts:

To see the set of completed files for Lesson 5, access:

ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER5

.

Note:

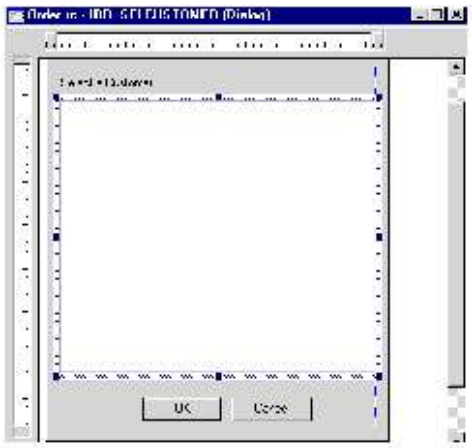
Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Creating a Customer List Dialog Box

Many of Nicole's customers have similar names. To ensure that the each customer gets the correct merchandise, the Order application will have a dialog box with a list of customers. Nicole's employees can then pick the appropriate name from the list when they create a new purchase order.

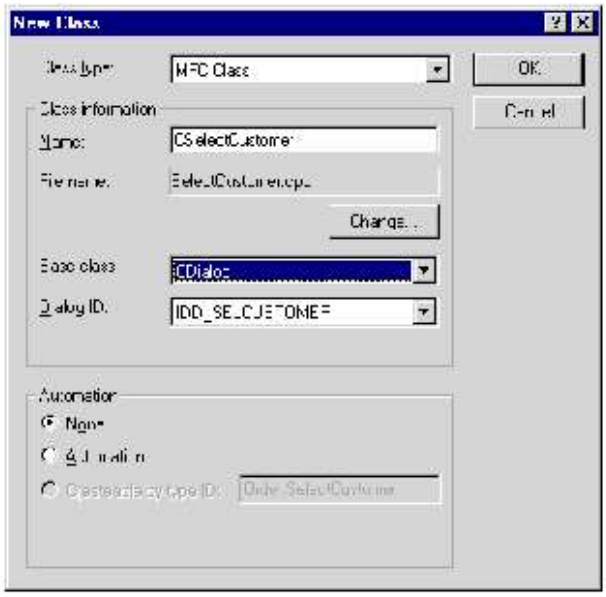
1. Create a dialog box with control ID = IDD_SELCUSTOMER. Customize the dialog box so that it contains an OK Button, Cancel Button, and List Box with control ID = IDC_CUSTLIST.

   The customer list dialog box should look similar to the one shown below. The list box displays a list of customers from the CUSTOMER table.



Part 2: Creating a New Class to Handle Events for the Customer Dialog Box

1. Create a new MFC class, CSelectCustomer, that is derived from CDialog. This class handles events occurring in the customer dialog box.



2. Allow the CSelectCustomer class to handle the events shown below:

Object ID	Event	Event Handler Function
CSelectCustomer	WM_INITDIALOG	OnInitDialog
IDOK	BN_CLICKED	OnOK

3. Create a member variable, m_CustomerList, for the customer list box, IDC_CUSTLIST, in the CSelectCustomer class. The variable, m_CustomerList, is a control handler of type CListBox. This enables users to select a customer from the customer list box for a purchase order.

4. To implement the events for the Customer dialog box, add the following member declarations to the CSelectCustomer class in SelectCustomer.h file:

```
public:
long GetSelection();              // obtain the selected customer from the
                                        // customer list box.
int DoModal(ODatabase *db);  // involve the customer dialog box
long m_nSelection;              // nth location of the selected customers
                                     // in the customer list.
ODatabase * m_pDatabase;       // the database
```

5. Include the header file oracl.h in the SelectCustomer.h file.

6. Add the following code to the CSelectCustomer::*OnInitDialog()* method in the CSelectCustomer.cpp file to display a list of customers in the customer list box:

```
BOOL CSelectCustomer::OnInitDialog()
{

...
CDialog::OnInitDialog();
ODynaset oCustList;
int index = 0;
```

```
oCustList.Open(*m_pDatabase, "SELECT CUSTOMER_ID, NAME FROM

DEMO.CUSTOMER");

while (!oCustList.IsEOF())
{

m_CustomerList.InsertString(index,
     (CString)oCustList.GetField(0) +
     " - " + oCustList.GetField(1));
m_CustomerList.SetItemData(index,
     (long)oCustList.GetField(0));
oCustList.MoveNext();
index++;

}

...
}
```

7. Add the following code for the CSelectCustomer::*DoModal()* method in the SelectCustomer.cpp file to invoke the customer dialog box:

```
int CSelectCustomer::DoModal(ODatabase *db)
{

m_pDatabase = db;
return CDialog::DoModal();

}
```

8. Add the following code for the CSelectCustomer::*GetSelection()* method in the SelectCustomer.cpp file to obtain the selected customer for the new purchase order:

```
long CSelectCustomer::GetSelection()
{

return(m_nSelection);

}
```

9. Add the following code for the OnOK() method of the CSelectCustomer class to get the selected customer from the customer list box for a new purchase order when the OK button in the customer dialog box is clicked:

```
void CSelectCustomer::OnOK()
{

// TODO: Add extra validation here
int nItem = -1;



nItem = m_CustomerList.GetCurSel();
if (nItem >=0)
{

// location of selected customer in the customer list box
m_nSelection = m_CustomerList.GetItemData(nItem);
CDialog::OnOK();

}
else
```

```
AfxMessageBox(_T("Please select a customer.\n"),
    MB_OK, 0);

}
```

Part 3: Creating "New Order", Commit Order", and "Cancel Order" Buttons

To enable Nicole's employees to easily add, commit, or cancel a new purchase order, create buttons in the IDD_ORDER_FORM dialog that enable them to perform these operations.

1. Include the header file, SelectCustomer.h, in the OrderView.h.file.

   This header file defines the CSelectCustomer class that is used when getting customer information for a new purchase order.

2. Create the buttons with the following properties shown in the Button Control ID and Event Handler Functions in the IDD_ORDER_FORM dialog.

Button Control ID	Button Caption	Event	Event Handler Function
IDC_NEWORDER	New Order	BN_CLICKED	OnNeworder
IDC_COMMITORDER	Commit Order	BN_CLICKED	OnCommitorder
IDC_CANCELORDER	Cancel Order	BN_CLICKED	OnCancelorder

These event handler functions are added to the COrderView class when the buttons specify the event handle.

Part 4: Enabling Users to Add a New Purchase Order

When a new purchase order is created, a new record is inserted into the SALES_ORDER table and the ITEM table. Content in the master table controls and detail table control is also updated to reflect the new purchase order.

1. Add the following code to the COrderView::*OnNeworder()* method in the OrderView.cpp file to add a new purchase order to the purchase order system:

```
#include "SelectCustomer.h" // define CSelectCustomer class
                            // to get customer information for
                      // a new purchase order
...
void COrderView::OnNeworder()
{

CWnd *tempWindow;
char szCUSTOMER_ID[8];
OField oORDER_ID;
CSelectCustomer selectCustomer;
oresult dbresult;

if (selectCustomer.DoModal(&GetDocument()->m_database) ==

IDCANCEL)
return;



if (GetDocument()->m_database.GetSession().

BeginTransaction() == OSUCCESS &&
m_pDynasetMaster->AddNewRecord() == OSUCCESS)

{
```

```
        UpdateData(FALSE);
        tempWindow = GetDlgItem(IDC_Column1);
        itoa(selectCustomer.GetSelection(),
             szCUSTOMER_ID, 10);
        tempWindow->SetWindowText(szCUSTOMER_ID);
        if (m_pDynasetMaster->GetEditMode() ==
             ODYNASET_EDIT_NEWRECORD)
        {

                        ODynaset oSequenceDynaset;

                        dbresult = oSequenceDynaset.Open(
                             GetDocument()->m_database,
                             "SELECT MAX(ORDER_ID) + 1 FROM
                             DEMO.SALES_ORDER");
                        oORDER_ID = oSequenceDynaset.GetField(0);
                        dbresult = oSequenceDynaset.Close();
                        tempWindow = GetDlgItem(IDC_Column3);
                        tempWindow->SetWindowText

                    ((CString)oORDER_ID);

    }
    dbresult = UpdateData();
    if (m_pDynasetMaster->Update() == OFAILURE)
    {

                  ProcessOO4OError(m_pDynasetMaster);
                  return;
    }
    m_pDynasetMaster->ResetToDefaultFilter();
    m_pDynasetMaster->AddFilter("ORDER_ID = " +
    (CString)oORDER_ID);
    m_pDynasetMaster->RefreshQuery();
    m_pDynasetCustomer->ResetToDefaultFilter();
    m_pDynasetCustomer->AddFilter("CUSTOMER_ID = " +
    (CString)m_pDynasetMaster->m_Column1);
    m_pDynasetCustomer->RefreshQuery();
    m_pDynasetDetail->ResetToDefaultFilter();
    m_pDynasetDetail->AddFilter("ORDER_ID = " +
    (CString)m_pDynasetMaster->m_Column3);
    m_pDynasetDetail->RefreshQuery();
    UpdateData(FALSE);

    }
    CButton *tempButton = (CButton *)GetDlgItem(IDC_NEWORDER);
    tempButton->EnableWindow(FALSE);
    tempButton = (CButton *)GetDlgItem(IDC_COMMITORDER);
    tempButton->EnableWindow(TRUE);
    tempButton = (CButton *)GetDlgItem(IDC_CANCELORDER);
    tempButton->EnableWindow(TRUE);

    }
```

Part 5: Enabling Users to Commit a New Purchase Order

When a new purchase order is created, users must be able to commit or cancel the change made to the purchase order system.

1. Add the following code to the COrderView::*OnCommitorder()* method in the OrderView.cpp file to commit a new purchase order:

```
void COrderView::OnCommitorder()
{
```

```
// TODO: Add your control notification handler code here
CString strORDER_DATE;
CString strSHIP_DATE;
CWnd *tempWindow;

m_pDynasetMaster->StartEdit();
tempWindow = GetDlgItem(IDC_Column2);
tempWindow->GetWindowText(strORDER_DATE);
tempWindow = GetDlgItem(IDC_Column4);
tempWindow->GetWindowText(strSHIP_DATE);
m_pDynasetMaster->m_Column2.SetValue(LPCTSTR(strORDER_DATE));
m_pDynasetMaster->m_Column4.SetValue(LPCTSTR(strSHIP_DATE));
m_pDynasetMaster->Update();

OSession oSess = GetDocument()->m_database.GetSession();
if (oSess.Commit() == OFAILURE)

ProcessOO4OError(&oSess);

m_pDynasetMaster->ResetToDefaultFilter();
m_pDynasetMaster->RefreshQuery();
UpdateData(FALSE);
OnMoveFirst();
CButton *tempButton = (CButton *)GetDlgItem(IDC_NEWORDER);
tempButton->EnableWindow(TRUE);
tempButton = (CButton *)GetDlgItem(IDC_COMMITORDER);
tempButton->EnableWindow(FALSE);
tempButton = (CButton *)GetDlgItem(IDC_CANCELORDER);
tempButton->EnableWindow(FALSE);

}
```

Part 6: Enabling Users to Cancel a New Purchase Order

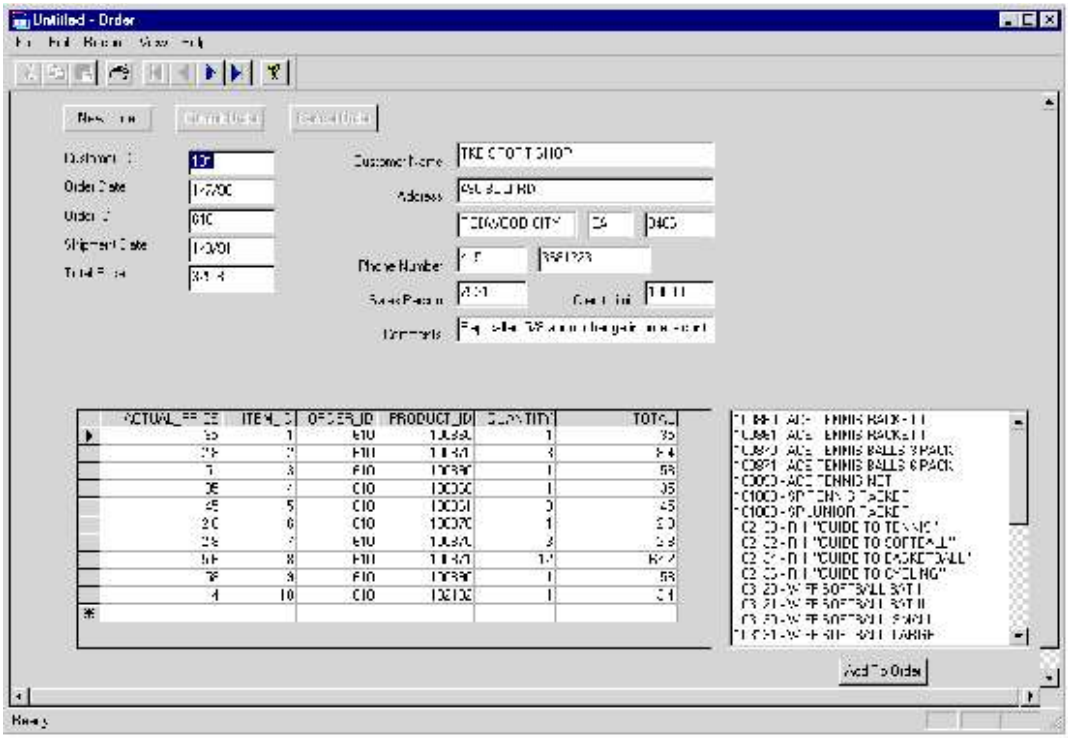
When a new purchase order is created, users must be able to commit or cancel the change made to the purchase order system.

1. Add the following code to the COrderView::*OnCancelorder()* method in the OrderView.cpp file to enable cancelling a new purchase order:

```
void COrderView::OnCancelorder()
{

// TODO: Add your control notification handler code here
GetDocument()->m_database.GetSession().Rollback();
m_pDynasetMaster->ResetToDefaultFilter();
m_pDynasetMaster->RefreshQuery();
UpdateData(FALSE);
OnMoveFirst();
CButton *tempButton = (CButton *)GetDlgItem(IDC_NEWORDER);
tempButton->EnableWindow(TRUE);
tempButton = (CButton *)GetDlgItem(IDC_COMMITORDER);
tempButton->EnableWindow(FALSE);
tempButton = (CButton *)GetDlgItem(IDC_CANCELORDER);
tempButton->EnableWindow(FALSE);

}
```

2. Build and run the application.

   It should look similar to the following screen:



Your application is now complete.