数据库系统开发实验

第七章实验、数据库编程

- 班级：07111606
- 学号：1120161881
- 实验人：张启洋

- 实验日期：2018/09/26 - 2018/09/28

目录

实验二、订单管理

一、实验方法

（一）开发环境

- 操作系统：Windows 10 Home China
- 服务器和实例名称：DWQETHINKPAD
- 数据库服务器：Microsoft SQL Server 2017 Developer Edition
- 数据库客户端：SQL Server Management Studio v17.8.1
- 编程工具：Microsoft Visual Studio Community 2017 Version 15.8.2
- 编程语言：C# (.NET Framework 4.6.1)

（二）问题回答

1. 使用哪种数据提供程序?

   SQL Server数据提供程序（System.Data.SqlClient）

2. 使用的数据连接对象是哪一个？连接对象是如何建立的？最后生成的连接对象中的连接字符串是什么？代表什么含义?
   - 数据连接对象： sqlConnection
   - 建立方法： 创建数据适配器是自动建立
   - 连接字符串：

     ```
     Data Source=DwqeThinkPad;Initial Catalog=AdventureWorks2016;Integrated Security=True
     ```

   - 字符串含义： 连接服务器是DwqeThinkPad默认实例

     数据库是AdventureWorks2016

     使用Windows认证登录

3. 使用的数据适配器对象是什么？其中的查询或更新语句是什么？如果有参数，则参数是如何处理的?

   1) 数据适配器：headerDataAdapter
   - 查询语句：

     ```
     SELECT   Sales.SalesOrderHeader.*
     FROM      Sales.SalesOrderHeader
     ```

   - 更新、插入、删除语句：由Visual Studio自动生成
   - 参数也由Visual Studio自动生成

2) 数据适配器：detailDataAdapter

- 查询语句

```
SELECT   Sales.SalesOrderDetail.*
FROM      Sales.SalesOrderDetail
```

- 更新、插入、删除语句：由Visual Studio自动生成
- 参数也由Visual Studio自动生成

4. 使用的数据集对象是什么？数据集中有哪些数据表？数据表是由哪些适配器对象生成的？（或采用其它方法）
   - 数据集对象：dataset
   - 包含数据表：SalesOrderHeader（适配器：headerDataAdapter），SalesOrderDetail（适配器：detailDataAdapter）

（三）添加代码及说明

1. 加载数据集

```
private void Form1_Load(object sender, EventArgs e)
{
    //调用两个数据适配器填充数据集
    headerDataAdapter.Fill(dataset, "SalesOrderHeader");
    detailDataAdapter.Fill(dataset, "SalesOrderDetail");
    //调用数据绑定，在窗口控件中加载数据
    detailBindingSource.DataSource = headerBindingSource;
    detailBindingSource.DataMember = "SalesOrderHeader_SalesOrderDetail";
}
```

2. 删除按钮

```
private void btnDelete_Click(object sender, EventArgs e)
{
    //当订单细节不为空，读取当前选中行索引，不是最后的空行，就执行删除操作
    if (this.dgvDetail.SelectedRows.Count > 0 && this.dgvDetail.SelectedRows[0].Index != this.dgvDetail.Rows.Count - 1)
    {
        //使用系统函数，删除选中行数据
        this.dgvDetail.Rows.RemoveAt(this.dgvDetail.SelectedRows[0].Index);
    }
}
```

3. 保存按钮

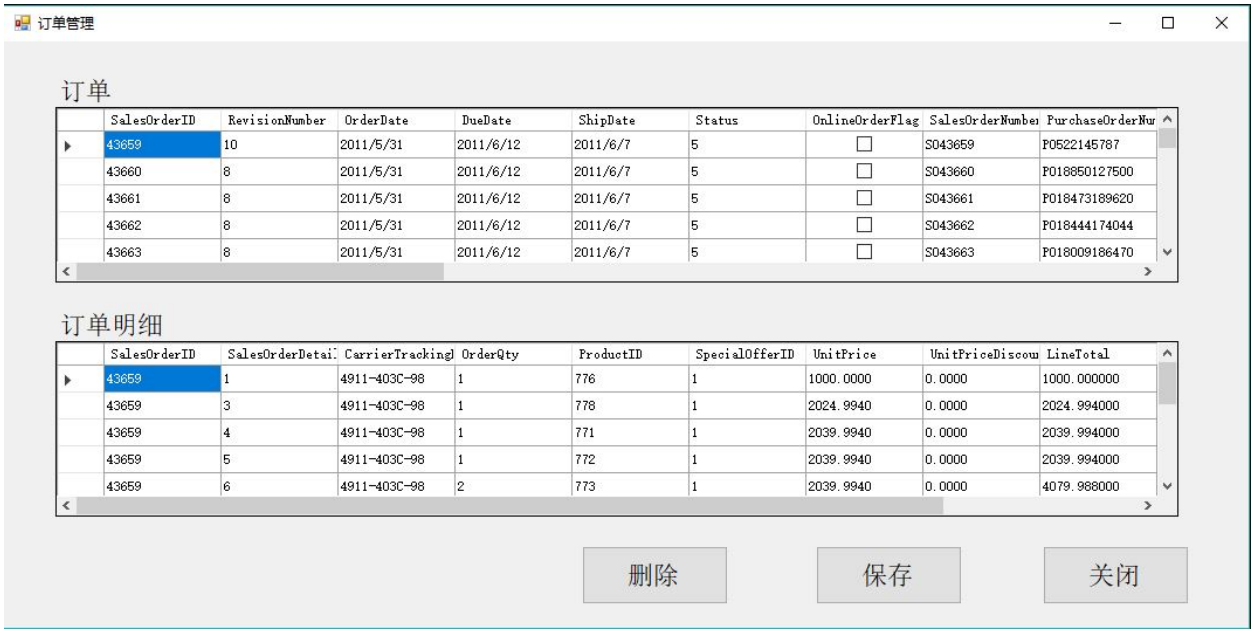
```
private void btnSave_Click(object sender, EventArgs e)
{
    //数据集未被修改，直接结束
    if (!dataset.HasChanges())
        return;
    //如果数据集被修改，执行修改操作
    try
    {
        //遍历每一行数据，如果被数据修改，修改数据库中数据
        foreach (DataRow dataRow in this.dataset.SalesOrderDetail.Rows)
            if (dataRow.RowState == DataRowState.Added)
                if (dataRow["rowguid"].Equals(System.DBNull.Value))
                    dataRow["rowguid"] = Guid.NewGuid();
        //调用数据适配器更新数据集
        headerDataAdapter.Update(dataset);
        detailDataAdapter.Update(dataset);
        MessageBox.Show("保存成功！", "保存");
    }
    //抓取错误信息，失败返回错误信息
    catch (Exception ex)
    {
        MessageBox.Show("保存失败！" + ex.Message, "保存");
    }
}
```

4. 关闭按钮

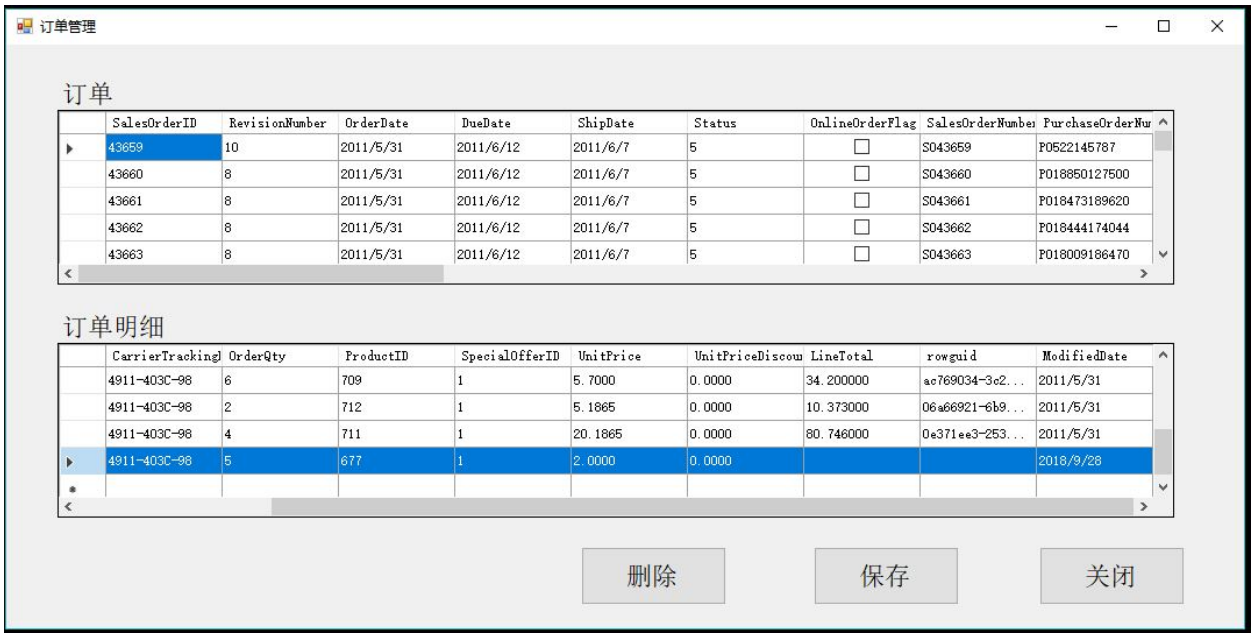
```
private void btnClose_Click(object sender, EventArgs e)
{
    //数据集未被修改，退出当前程序
    if (!dataset.HasChanges())
        Application.Exit();
    //弹出对话框，用户选择是否保存
    DialogResult result = MessageBox.Show(
        "此次修改保存，是否保存将最新修改保存至数据库？", "未保存",
        MessageBoxButtons.YesNo,
        MessageBoxIcon.Warning
    );
    //如果选择确认按钮，保存当前程序并退出
    if (result == DialogResult.Yes)
        btnSave_Click(sender, e);
    Application.Exit();
}
```

二、实验结果

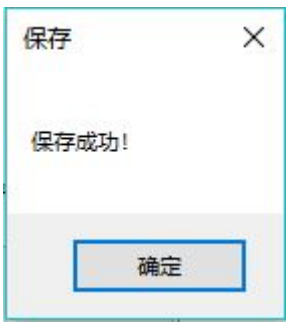
1. 订单明细查询：点击上方订单其中一行数据，在下方订单明细中查看到详细信息



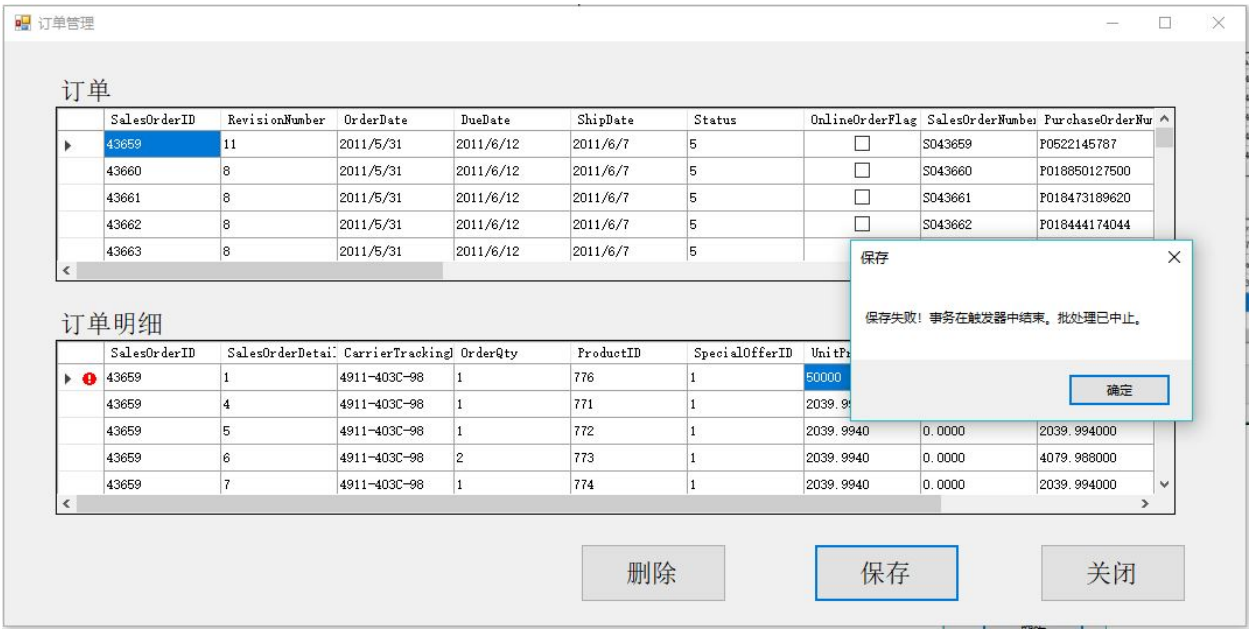
2. 订单明细插入：点击下方订单明细中最后一行，严格按照数据格式逐列，输入数据，其中LineTotal和rowguid可以为空值，完成输入后点击保存按钮



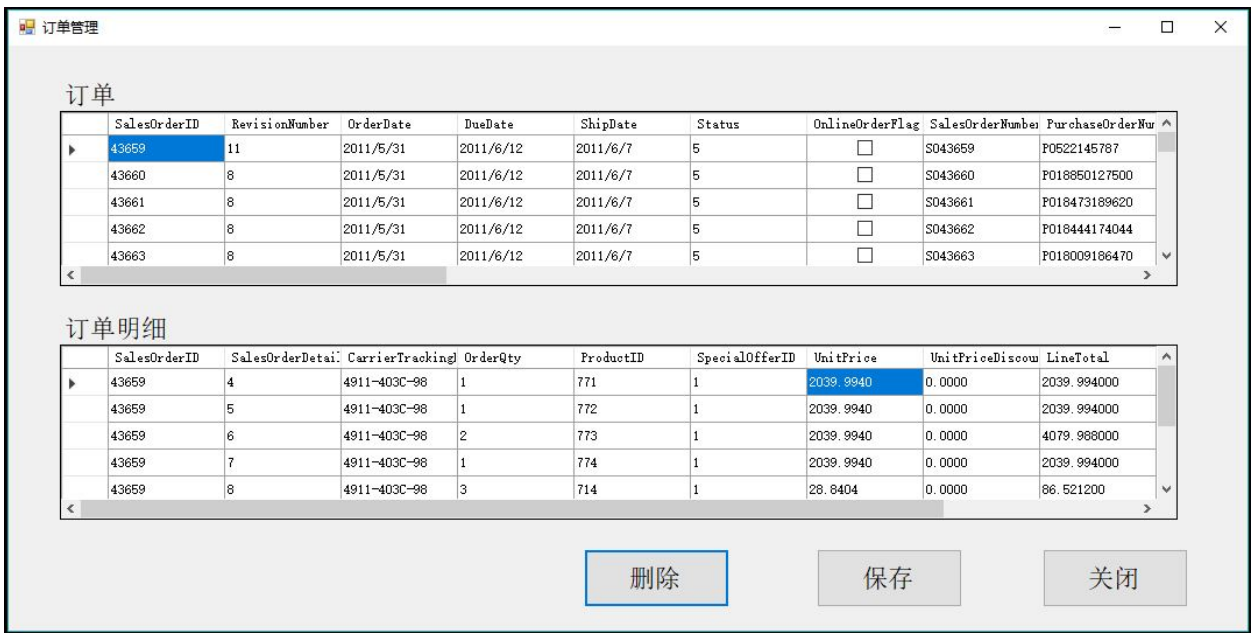
如果成功保存，弹出提示窗口



3. 订单明细更新：点击下方订单明细中需要修改的数据，输入修改后数据，完成输入后点击保存按钮，此时会触发数据库中的触发器，例如UnitPrice大于ListPrice 时，就会触发第四章实验中创建的触发器，如果没有触发器被触发，则保存成功

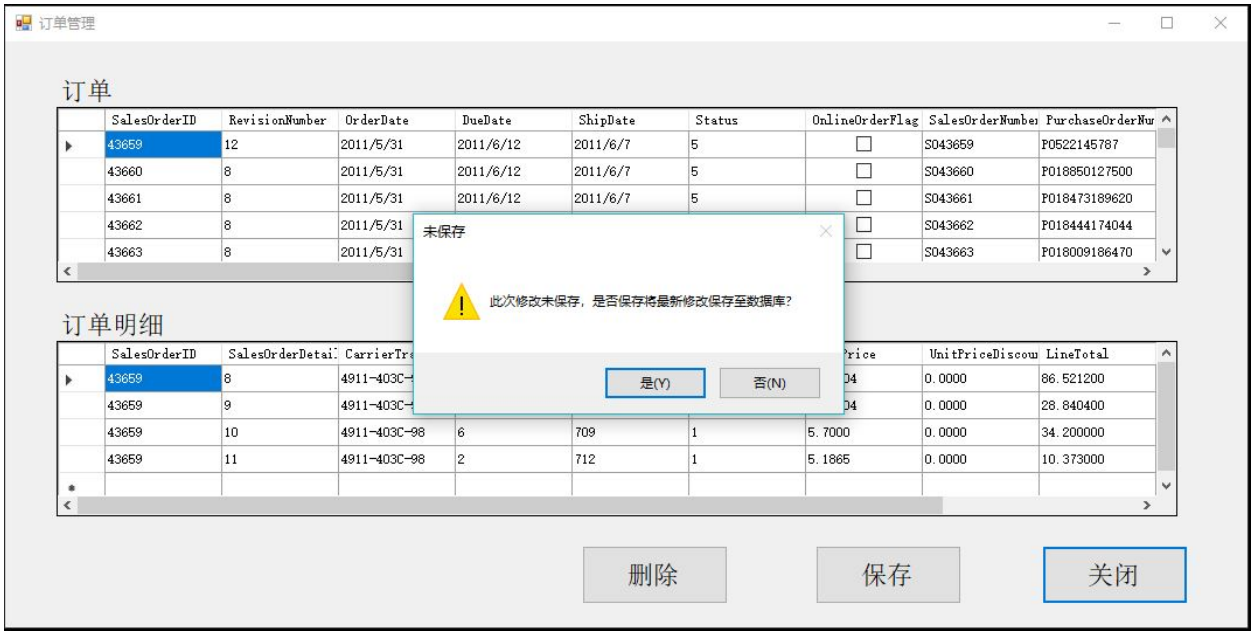


4. 订单明细删除：点击下方订单明细中需要删除的一行数据，点击删除按钮，再点击保存按钮，第一行数据成功被删除



5. 保存按钮：在对订单明细修改后，需要点击保存按钮，修改数据库，否则在关闭后再次打开时，发现修改未成功，这是因为在点击保存按钮前，只是修改了数据集，点击保存按钮后，才会修改数据库，下次打开时发现成功修改数据

6. 关闭按钮：如果对数据集做出修改后没有点击保存按钮，在点击关闭按钮时，会弹出对话框，提示用户进行保存，点击按钮是，保存并退出程序，显示保存结果，点击按钮否，直接退出程序



三、实验总结

- 在配置数据库适配器时，发现界面中的数据集未被填充，在修改无果后，重新做了一遍，发现是调用的函数名被意外修改，但编译器不会报错，所以问题较难发现
- 可以改进的地方就是应该在点击上方关闭窗口按钮时，已设置与下方关闭按钮相同的功能，更加方便用户使用；在窗口最大化后，里面控件未发生变化，可以将空间设置为自适应窗口大小，更加美观

2018/09/28 23:14　　实验人：张启洋　　学号：1120161881

实验三、购物车

一、实验方法

（一）开发环境

- 操作系统：Windows 10 Home China
- 服务器和实例名称：DWQETHINKPAD
- 数据库服务器：Microsoft SQL Server 2017 Developer Edition
- 数据库客户端：SQL Server Management Studio v17.8.1
- 编程工具：Microsoft Visual Studio Community 2017 Version 15.8.2
- 编程语言：C# (.NET Framework 4.6.1)

（二）问题回答

1. 使用哪种数据提供程序?

   SQL Server数据提供程序（System.Data.SqlClient）

2. 使用的数据连接对象是哪一个？连接对象是如何建立的？最后生成的连接对象中的连接字符串是什么？代表什么含义?
   - 数据连接对象： sqlConnection
   - 建立方法： 创建数据适配器是自动建立
   - 连接字符串：

     ```
     Data Source=DwqeThinkPad;Initial Catalog=AdventureWorks2016;Integrated Security=True
     ```

   - 字符串含义： 连接服务器是DwqeThinkPad默认实例

     数据库是AdventureWorks2016

     使用Windows认证登录

3. 使用的数据适配器对象是什么？其中的查询或更新语句是什么？如果有参数，则参数是如何处理的?

   1) 数据适配器：ShoppingDataAdapter
   - 查询语句：

     ```
     SELECT   Sales.ShoppingCartItem.*
     FROM      Sales.ShoppingCartItem
     ```

   - 更新、插入、删除语句：由Visual Studio自动生成
   - 参数也由Visual Studio自动生成

2) 数据适配器：ProductionDataAdapter

- 查询语句

```
SELECT   Production.ProductInventory.*
FROM      Production.ProductInventory
```

- 更新、插入、删除语句：由Visual Studio自动生成
- 参数也由Visual Studio自动生成

4. 使用的数据集对象是什么？数据集中有哪些数据表？数据表是由哪些适配器对象生成的？（或采用其它方法）
   - 数据集对象：dataset
   - 包含数据表：ShoppingCartItem（适配器：ShoppingDataAdapter），ProductInventory（适配器：ProductionDataAdapter）

（三）添加代码及说明

1. 创建添加商品存储过程

```
USE [AdventureWorks2016]
GO
--删除已有此存储过程
IF OBJECT_ID('[AddProduction]', 'P') IS NOT NULL
    DROP PROCEDURE [AddProduction]
GO
--重新创建存储过程
CREATE PROCEDURE [AddProduction]
    @ProductID int,
    @LocationID int,
    @Quantity int

AS
    DECLARE @TotalPrcCNT int --添加物品总量

    SELECT @TotalPrcCNT = P.Quantity
    FROM [Production].[ProductInventory] P
    WHERE P.ProductID = @ProductID AND P.LocationID = @LocationID
    --判定添加物品数量是否大于库存数量
    IF @TotalPrcCNT < @Quantity
        RETURN -1

    DECLARE @ShoppingCartID nvarchar(50) = CONVERT(nvarchar(50), @LocationID)

    --如果购物车已有该物品，更新数量，没有该物品，添加新行
    IF EXISTS (
        SELECT * FROM [Sales].[ShoppingCartItem]
        WHERE [ProductID] = @ProductID AND [ShoppingCartID] = @ShoppingCartID
```

```
        )
        UPDATE [Sales].[ShoppingCartItem]
        SET [Quantity] += @Quantity
        WHERE [ProductID] = @ProductID
    ELSE
        INSERT INTO [Sales].[ShoppingCartItem] (
            ShoppingCartID, Quantity, ProductID, DateCreated, ModifiedDate
            )
        VALUES (
            @ShoppingCartID, @Quantity, @ProductID, GETDATE(), GETDATE()
            )

    SET @TotalPrcCNT = @Quantity
    --更新仓库中数据
    UPDATE [Production].[ProductInventory]
    SET [Quantity] -= @TotalPrcCNT
    WHERE [Production].[ProductInventory].[ProductID] = @ProductID AND
        [Production].[ProductInventory].[LocationID] = @LocationID

    RETURN 1

GO
```

2. 创建删除商品存储过程

```
--删除已有此存储过程
IF OBJECT_ID('[DeleteProduction]', 'P') IS NOT NULL
    DROP PROCEDURE DeleteProduction
GO
--重新创建存储过程
CREATE PROCEDURE DeleteProduction
        @ShoppingCarItemID int,
        @ShoppingCarID int,
        @Quantity int,
        @ProductID int
AS
    --更新仓库中数据
    UPDATE [Production].[ProductInventory]
    SET Quantity += @Quantity
    WHERE [Production].[ProductInventory].[ProductID] = @ProductID AND
        [Production].[ProductInventory].[LocationID] = Convert(smallint, @ShoppingCarID)

    --删除购物车中记录
    DELETE FROM [Sales].[ShoppingCartItem]
    WHERE [Sales].[ShoppingCartItem].[ShoppingCartItemID] = @ShoppingCarItemID

GO
```

3. 加载数据集

```
private void Form1_Load(object sender, EventArgs e)
{
    ProductionDataAdapter.Fill(dataset, "ProductInventory");
    ShoppingDataAdapter.Fill(dataset, "ShoppingCartItem");
}
```

4. 添加按钮

```
private void btnAdd_Click(object sender, EventArgs e)
{
    //读取数据
    int ProductID = Int32.Parse(dgvProduction.CurrentRow.Cells[0].Value.ToString());
    int LocationID = Int32.Parse(dgvProduction.CurrentRow.Cells[1].Value.ToString());
    int Quantity = Int32.Parse(ProNum.Text);
    //判定是否输入
    if (ProNum.Text.Trim() == String.Empty)
    {
        MessageBox.Show("请输入要加入购物车的商品数量!", "提示",
            MessageBoxButtons.OK, MessageBoxIcon.Warning
            );
        return;
    }
    //添加调用存储过程命令
    SqlCommand cmd_AddProduction = new SqlCommand("AddProduction", this.sqlConnection);
    cmd_AddProduction.CommandType = CommandType.StoredProcedure;
    //传输数据
    cmd_AddProduction.Parameters.AddWithValue("@ProductID", SqlDbType.Int);
    cmd_AddProduction.Parameters["@ProductID"].Value = ProductID;
    cmd_AddProduction.Parameters.AddWithValue("@LocationID", SqlDbType.Int);
    cmd_AddProduction.Parameters["@LocationID"].Value = LocationID;
    cmd_AddProduction.Parameters.AddWithValue("@Quantity", SqlDbType.Int);
    cmd_AddProduction.Parameters["@Quantity"].Value = Quantity;
    //打开数据连接
    this.sqlConnection.Open();
    int return_value = cmd_AddProduction.ExecuteNonQuery();//读取存储过程返回值
    this.sqlConnection.Close();

    //根据返回值提示信息
    if (return_value == -1)
    {
        MessageBox.Show("加入购物车失败！加入购物车数量不得超过库存数量", "提示",
            MessageBoxButtons.OK, MessageBoxIcon.Information
            );
        return;
    }
    else
    {
        MessageBox.Show("您已将商品添加入购物车！ ", "提示",
            MessageBoxButtons.OK, MessageBoxIcon.Information
            );
```

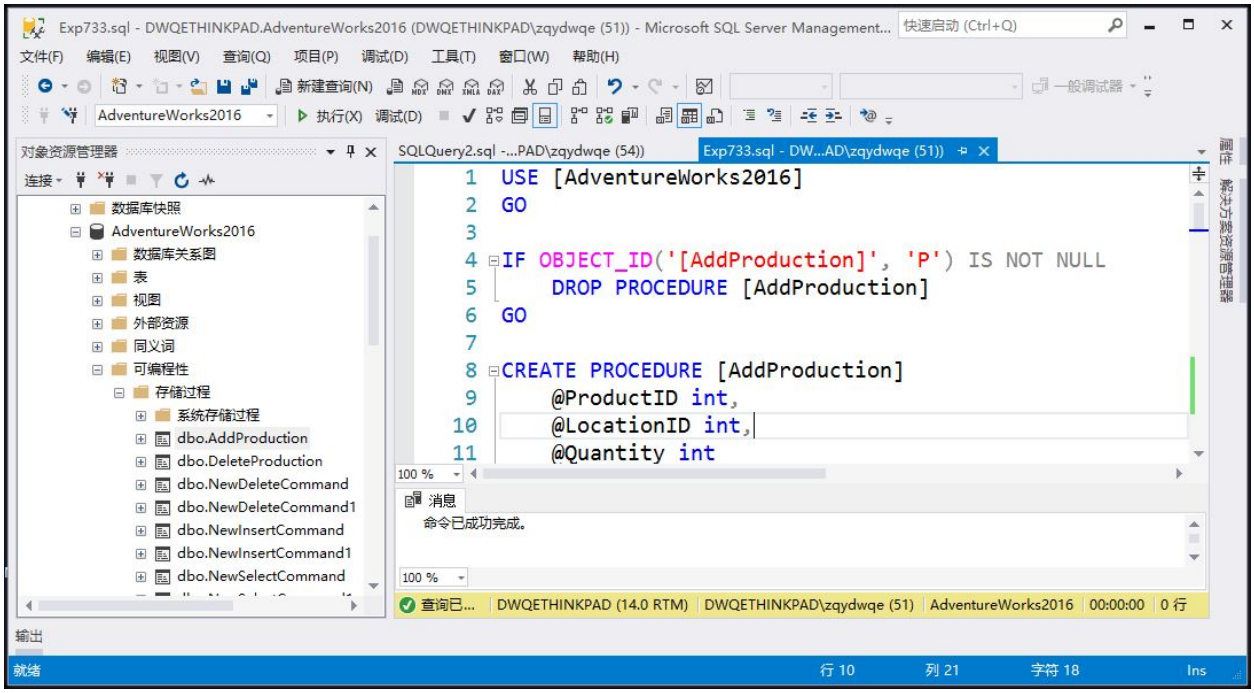
```
        this.ProductionDataAdapter.Fill(dataset);
        this.ShoppingDataAdapter.Fill(dataset);
        return;
    }
}
```

5. 删除按钮

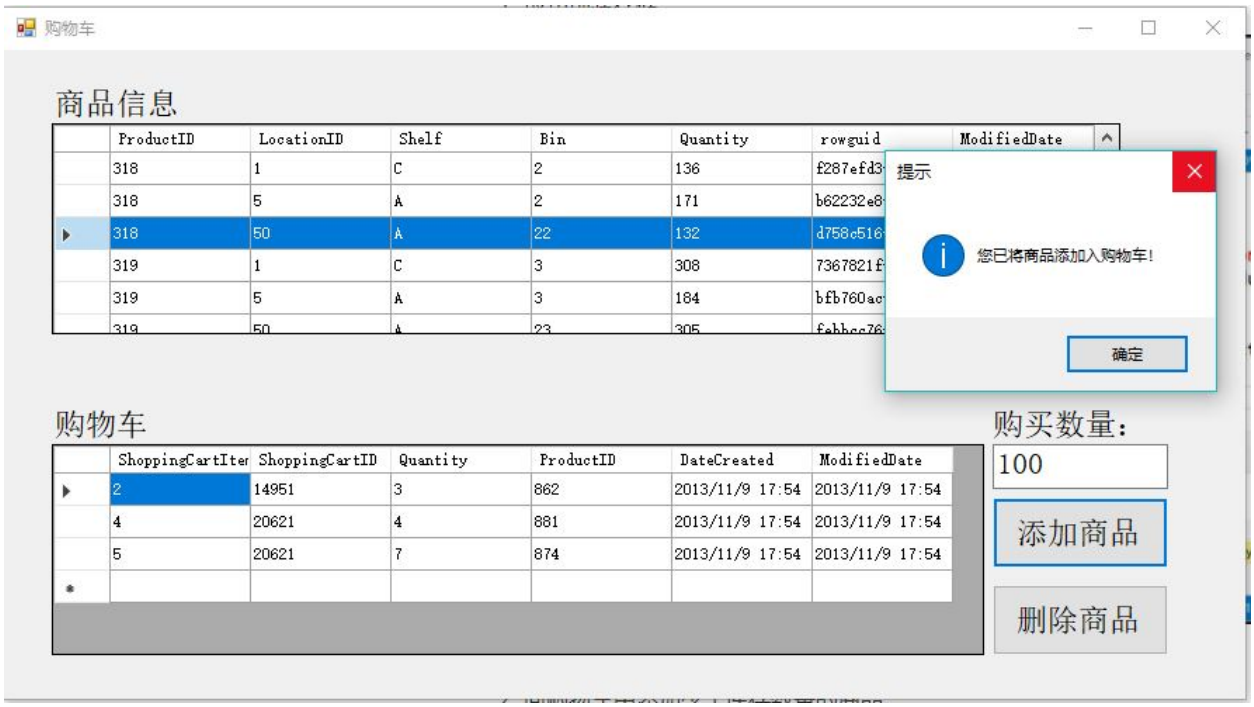
```
private void btnDelete_Click(object sender, EventArgs e)
{
    //读取数据
    int ShoppingCartItemID =
Int32.Parse(dgvShopping.CurrentRow.Cells[0].Value.ToString());
    int ShoppingCartID = Int32.Parse(dgvShopping.CurrentRow.Cells[1].Value.ToString());
    int Quantity = Int32.Parse(dgvShopping.CurrentRow.Cells[2].Value.ToString());
    int ProductID = Int32.Parse(dgvShopping.CurrentRow.Cells[3].Value.ToString());
    //添加调用存储过程命令
    SqlCommand cmd_DeleteProduction = new SqlCommand("DeleteProduction",
this.sqlConnection);
    cmd_DeleteProduction.CommandType = CommandType.StoredProcedure;
    //传输数据
    cmd_DeleteProduction.Parameters.AddWithValue("@ShoppingCarItemID", SqlDbType.Int);
    cmd_DeleteProduction.Parameters["@ShoppingCarItemID"].Value = ShoppingCartItemID;
    cmd_DeleteProduction.Parameters.AddWithValue("@ShoppingCarID", SqlDbType.Int);
    cmd_DeleteProduction.Parameters["@ShoppingCarID"].Value = ShoppingCartID;
    cmd_DeleteProduction.Parameters.AddWithValue("@Quantity", SqlDbType.Int);
    cmd_DeleteProduction.Parameters["@Quantity"].Value = Quantity;
    cmd_DeleteProduction.Parameters.AddWithValue("@ProductID", SqlDbType.Int);
    cmd_DeleteProduction.Parameters["@ProductID"].Value = ProductID;
    //打开数据连接
    this.sqlConnection.Open();
    cmd_DeleteProduction.ExecuteNonQuery();
    this.sqlConnection.Close();
    //重新填充数据库
    this.dataset.Clear();
    this.ProductionDataAdapter.Fill(dataset);
    this.ShoppingDataAdapter.Fill(dataset);
    //返回提示信息
    MessageBox.Show("删除成功！", "提示",
        MessageBoxButtons.OK, MessageBoxIcon.Information
        );
}
```

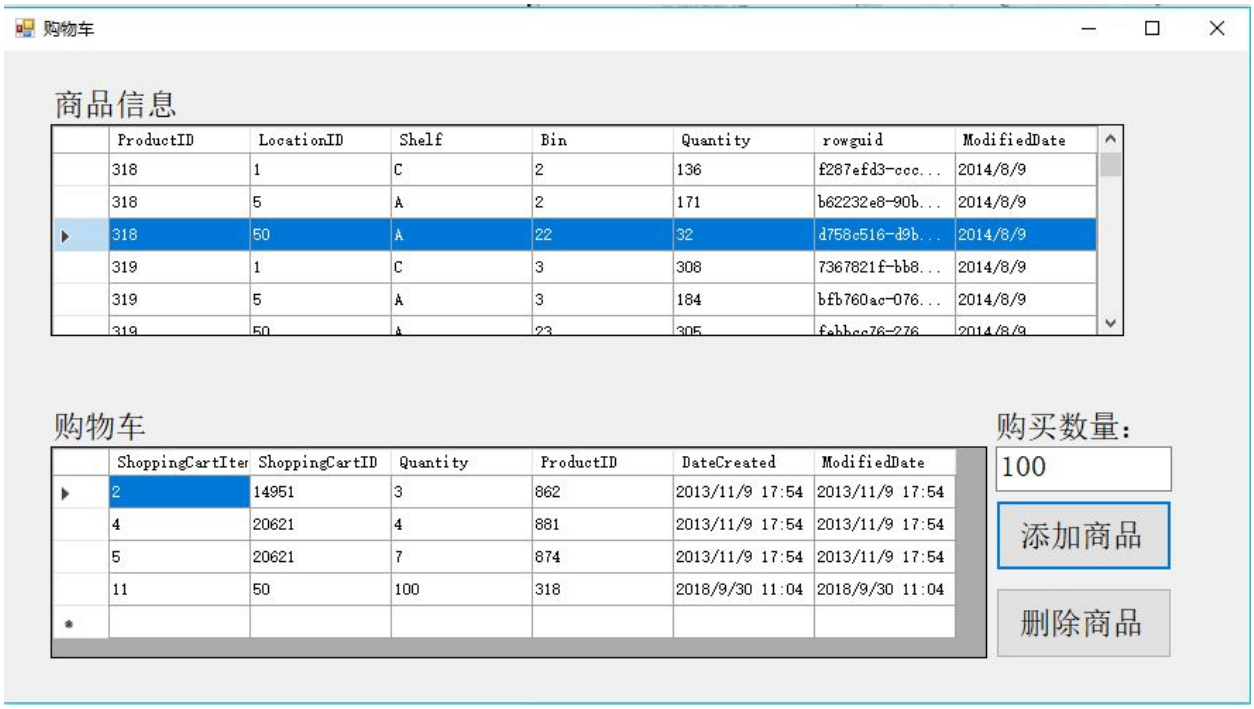
二、实验结果

1. 成功创建存储

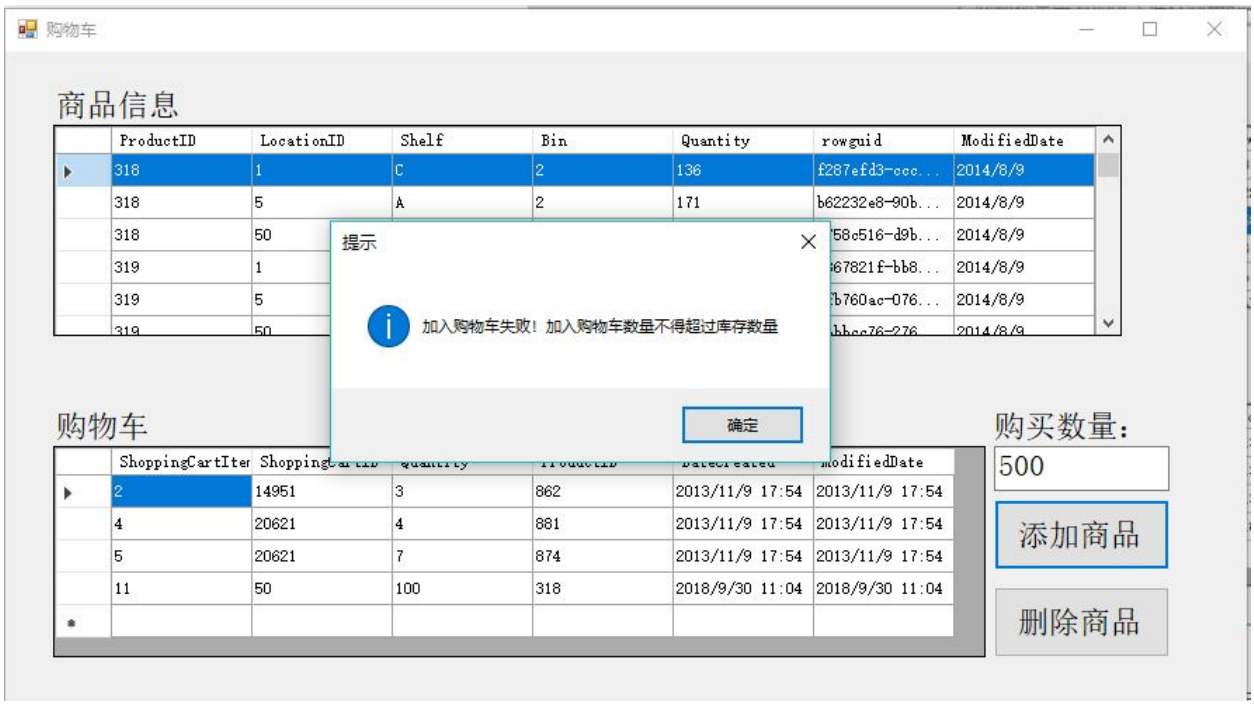


2. 向购物车中添加少于库存数量的商品：在商品信息中点击需要商品，输入商品数量，点击添加商品按钮，弹出提示信息，点击确定，发现购物车中出现新的物品

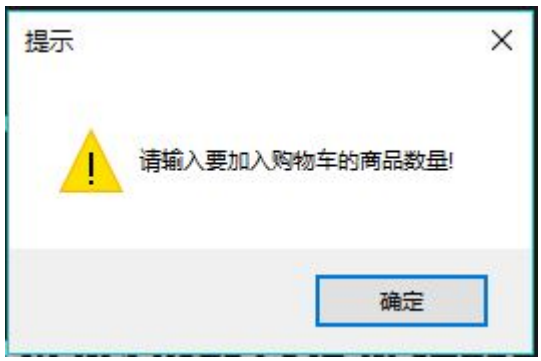




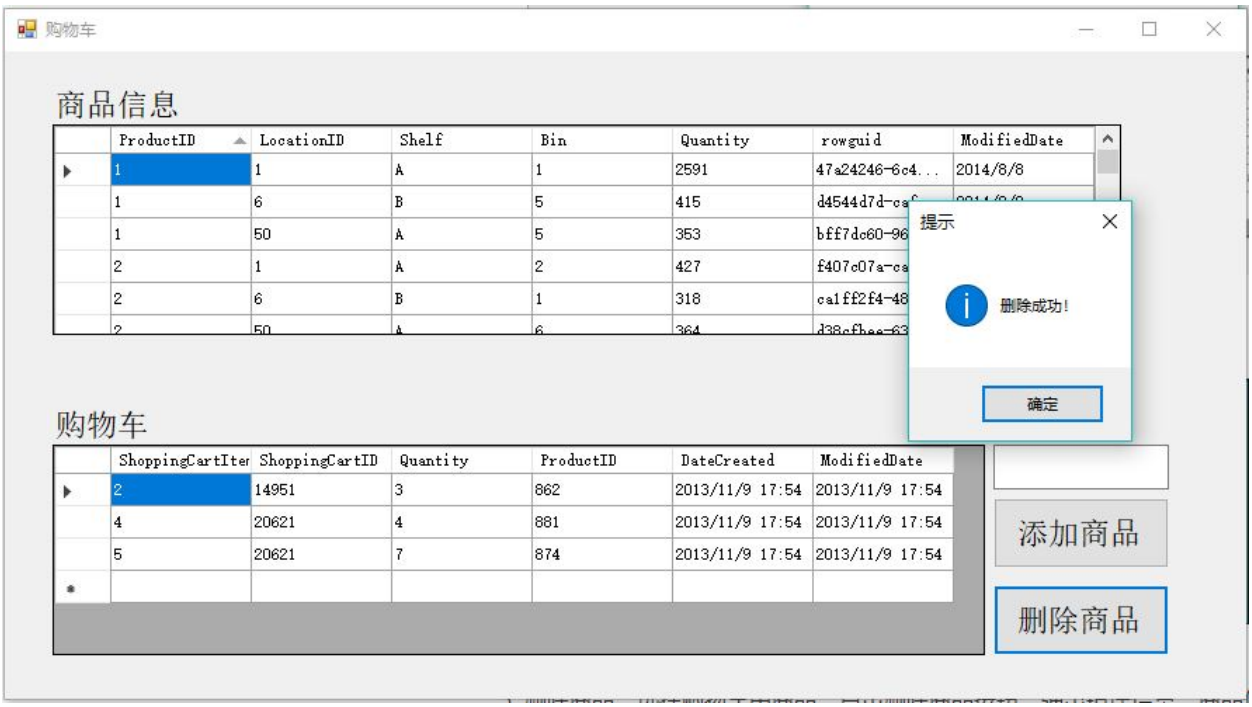
3. 向购物车中添加大于库存数量的商品：与上一步操作相同，但返回错误提示信息

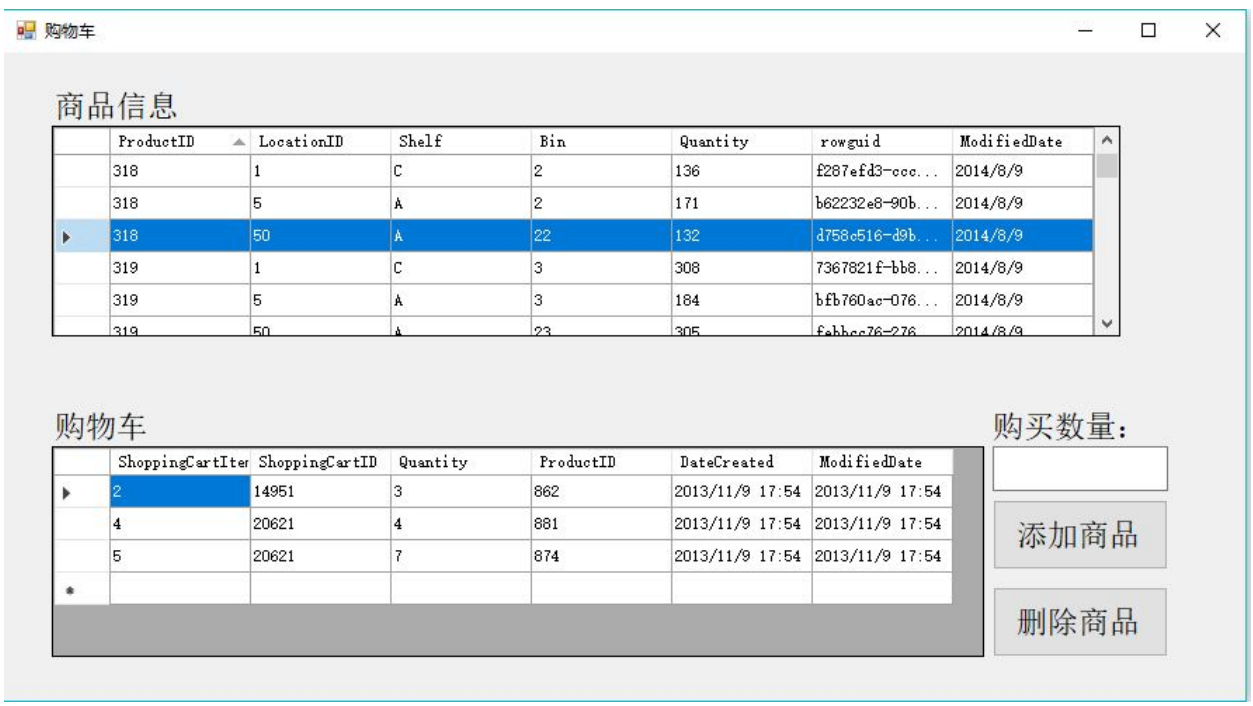


4. 未输入购买数量就点击添加商品按钮：返回错误信息，提示输入数量



5. 删除商品：选择购物车中商品，点击删除商品按钮，弹出错误信息，商品信息中商品数量增加





三、实验总结

- 刚开始点击添加按钮时，数据未被修改，经检查发现在调用存储过程，传参数是出现问题，修改参数后，成功解决
- 在删除商品时，发现商品信息中商品数量已经恢复，但是购物车中物品为被删除，重写存储过程无果后，上网查找资料后发现，需要在数据集填充前，添加dataset.Clear()函数
- 程序没有设置删除时选择产品数量的功能，可以在之后添加此功能

2018/09/29 19:18　　实验人：张启洋　　学号：1120161881