# 1.安装SQlite链接器

## 1)新建一个c#控制台工程,取名c#SQliteDemo

|  |
| --- |
|  |

## 2)在项目文件上面点击右键->过来Nuget程序包,会出现下面的界面,注意需要点击浏览选项卡,

|  |
| --- |
|  |

## 3)在文本框里面输入System.Data.SQlite,注意有很多SQlite链接器不过功能不一样,有些笔记轻量,这个比较好

|  |
| --- |
|  |

## 4)点击右边的安装按钮,下面就开始安装

|  |
| --- |
|  |

## 5)然后会弹出一个对话框,只需要点击应用按钮

|  |
| --- |
|  |

## 6)然后弹出接受许可对话框,点击我接受

|  |
| --- |
|  |

## 7)然后程序继续安装

|  |
| --- |
|  |

## 8)安装完成的界面如下

|  |
| --- |
|  |

### 注意,它会把下面3个相关的包都一起安装了.

## 9)查看一下项目的应用,如图

|  |
| --- |
|  |

## 10)然后我们需要在程序里面导入SQlite组件

|  |
| --- |
|  |

## 11)先初始化数据库,我们编写一个InitDB函数,代码如下,然后我们在Main函数里面调用这个函数

|  |
| --- |
|  |

### 运行程序,没有报错,

|  |
| --- |
|  |

## 12)我们我们打开项目的bin目录里面的Debug文件夹,发现新建了一个school.db数据库

|  |
| --- |
|  |

## 13)使用SQlite Database Brower工具打开这个数据库,效果如下,可以看到我们创建的数据表students

|  |
| --- |
|  |

## 14)然后我们编写一个AddNew函数完数据库里面插入4条记录,代码如下

|  |
| --- |
| static void AddNew()  {  InitDB();  string sql = "insert into students values(100,'Jackline',18,'female',90)";  SQLiteCommand cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  int result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);  sql = "insert into students values(101,'LiLiana',17,'female',80)";  cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);    sql = "insert into students values(102,'Jack',19,'male',70)";  cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);  sql = "insert into students values(103,'Rosie',18,'female',95)";  cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);  } |

### 然后在Main函数里面调用这个函数

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

## 15)我们用上面的数据库浏览器工具可以看到我们添加的数据

|  |
| --- |
|  |

## 16)然,后我们添加一个ShowAll方法,根据成绩降序排序,然后在Main函数里面调用这个函数

|  |
| --- |
|  |

### 运行程序,效果如下

|  |
| --- |
|  |

## 17)我们来实现一下条件查询,例如根据年龄来查询,代码如下,然后我们在Main函数里面调用这个函数

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

## 18)我们来做修改功能,编写一个UpdateData函数,然后在Main函数里面调用

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

## 19)使用SQlite Database Brower工具查看数据库,发现Rosie的成绩被改为75了

|  |
| --- |
|  |

## 20)然后我们来做删除功能,编写一个DeleteByNum(int num)函数,然后在Main函数里面调用这个函数

|  |
| --- |
|  |

### 运行程序,效果如下

|  |
| --- |
|  |

## 21)然后我们利用可视化工具查看数据表,发现学生Jack被删除了

|  |
| --- |
|  |

# 这一节的学习到此为止,本实例的完整代码如下

## Program.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Data;  using System.Data.SQLite;  namespace c\_SQliteDemo  {  internal class Program  {  public static SQLiteConnection conn;  static void InitDB()  {  if (conn == null)  conn = new SQLiteConnection("Data Source=School.db; Version=3;");  if(conn.State != ConnectionState.Open)  conn.Open();  string sql = "create table if not exists students (no int ,name varchar(20),age int,gender varchar(6),score int)";  SQLiteCommand cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  cmd.ExecuteNonQuery();  }  static void AddNew()  {  InitDB();  string sql = "insert into students values(100,'Jackline',18,'female',90)";  SQLiteCommand cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  int result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);  sql = "insert into students values(101,'LiLiana',17,'female',80)";  cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);    sql = "insert into students values(102,'Jack',19,'male',70)";  cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);  sql = "insert into students values(103,'Rosie',18,'female',95)";  cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}",result);  }  static void ShowAll()  {  InitDB();  string sql = "select \* from students order by score desc";  SQLiteCommand cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  SQLiteDataReader reader = cmd.ExecuteReader();    while(reader.Read())  {  Console.WriteLine($"Number:{reader["no"]},Name:{reader["name"]},Age:{reader["age"]}," +  $"Gender:{reader["gender"]},Score:{reader["score"]}");  }  }  //根据年龄查询  static void QueryByAge(int age)  {  InitDB();  string sql = "select \* from students where age = @age";  SQLiteCommand cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  cmd.Parameters.AddWithValue("age", age);  SQLiteDataReader reader = cmd.ExecuteReader();  while (reader.Read())  {  Console.WriteLine($"Number:{reader["no"]},Name:{reader["name"]},Age:{reader["age"]}," +  $"Gender:{reader["gender"]},Score:{reader["score"]}");  }  }  //修改数据  static void UpdateData()  {  InitDB();  string sql = "update students set score=75 where name=@name";  SQLiteCommand cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  cmd.Parameters.AddWithValue("name", "Rosie");  int result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}", result);  }  //删除数据  static void DeleteByNum(int num)  {  InitDB();  string sql = "delete from students where no=@no";  SQLiteCommand cmd = new SQLiteCommand(sql, conn);  cmd.CommandType = CommandType.Text;  cmd.Parameters.AddWithValue("no", num);  int result = cmd.ExecuteNonQuery();  Console.WriteLine("受影响行数{0}", result);  }  static void Main(string[] args)  {  //InitDB();  //AddNew();  //ShowAll();  //QueryByAge(18);  //UpdateData();  DeleteByNum(102);  }  }  } |

# 网上找了一个sqlite帮助类，有需要的可以看下

|  |
| --- |
| **/// <summary>**  **/// SQLiteHelper is a utility class similar to "SQLHelper" in MS**  **/// Data Access Application Block and follows similar pattern.**  **/// </summary>**  **public class SQLiteHelper**  **{**  **/// <summary>**  **/// Creates a new <see cref="SQLiteHelper"/> instance. The ctor is marked private since all members are static.**  **/// </summary>**  **private SQLiteHelper()**  **{**  **}**  **/// <summary>**  **/// Creates the command.**  **/// </summary>**  **/// <param name="connection">Connection.</param>**  **/// <param name="commandText">Command text.</param>**  **/// <param name="commandParameters">Command parameters.</param>**  **/// <returns>SQLite Command</returns>**  **public static SQLiteCommand CreateCommand(SQLiteConnection connection, string commandText, params SQLiteParameter[] commandParameters)**  **{**  **SQLiteCommand cmd = new SQLiteCommand(commandText, connection);**  **if (commandParameters.Length > 0)**  **{**  **foreach (SQLiteParameter parm in commandParameters)**  **cmd.Parameters.Add(parm);**  **}**  **return cmd;**  **}**  **/// <summary>**  **/// Creates the command.**  **/// </summary>**  **/// <param name="connectionString">Connection string.</param>**  **/// <param name="commandText">Command text.</param>**  **/// <param name="commandParameters">Command parameters.</param>**  **/// <returns>SQLite Command</returns>**  **public static SQLiteCommand CreateCommand(string connectionString, string commandText, params SQLiteParameter[] commandParameters)**  **{**  **SQLiteConnection cn = new SQLiteConnection(connectionString);**  **SQLiteCommand cmd = new SQLiteCommand(commandText, cn);**  **if (commandParameters.Length > 0)**  **{**  **foreach (SQLiteParameter parm in commandParameters)**  **cmd.Parameters.Add(parm);**  **}**  **return cmd;**  **}**  **/// <summary>**  **/// Creates the parameter.**  **/// </summary>**  **/// <param name="parameterName">Name of the parameter.</param>**  **/// <param name="parameterType">Parameter type.</param>**  **/// <param name="parameterValue">Parameter value.</param>**  **/// <returns>SQLiteParameter</returns>**  **public static SQLiteParameter CreateParameter(string parameterName, System.Data.DbType parameterType, object parameterValue)**  **{**  **SQLiteParameter parameter = new SQLiteParameter();**  **parameter.DbType = parameterType;**  **parameter.ParameterName = parameterName;**  **parameter.Value = parameterValue;**  **return parameter;**  **}**  **/// <summary>**  **/// Shortcut method to execute dataset from SQL Statement and object[] arrray of parameter values**  **/// </summary>**  **/// <param name="connectionString">SQLite Connection string</param>**  **/// <param name="commandText">SQL Statement with embedded "@param" style parameter names</param>**  **/// <param name="paramList">object[] array of parameter values</param>**  **/// <returns></returns>**  **public static DataSet ExecuteDataSet(string connectionString, string commandText, object[] paramList)**  **{**  **SQLiteConnection cn = new SQLiteConnection(connectionString);**  **SQLiteCommand cmd = cn.CreateCommand();**  **cmd.CommandText = commandText;**  **if (paramList != null)**  **{**  **AttachParameters(cmd, commandText, paramList);**  **}**  **DataSet ds = new DataSet();**  **if (cn.State == ConnectionState.Closed)**  **cn.Open();**  **SQLiteDataAdapter da = new SQLiteDataAdapter(cmd);**  **da.Fill(ds);**  **da.Dispose();**  **cmd.Dispose();**  **cn.Close();**  **return ds;**  **}**  **/// <summary>**  **/// Shortcut method to execute dataset from SQL Statement and object[] arrray of parameter values**  **/// </summary>**  **/// <param name="cn">Connection.</param>**  **/// <param name="commandText">Command text.</param>**  **/// <param name="paramList">Param list.</param>**  **/// <returns></returns>**  **public static DataSet ExecuteDataSet(SQLiteConnection cn, string commandText, object[] paramList)**  **{**  **SQLiteCommand cmd = cn.CreateCommand();**  **cmd.CommandText = commandText;**  **if (paramList != null)**  **{**  **AttachParameters(cmd, commandText, paramList);**  **}**  **DataSet ds = new DataSet();**  **if (cn.State == ConnectionState.Closed)**  **cn.Open();**  **SQLiteDataAdapter da = new SQLiteDataAdapter(cmd);**  **da.Fill(ds);**  **da.Dispose();**  **cmd.Dispose();**  **cn.Close();**  **return ds;**  **}**  **/// <summary>**  **/// Executes the dataset from a populated Command object.**  **/// </summary>**  **/// <param name="cmd">Fully populated SQLiteCommand</param>**  **/// <returns>DataSet</returns>**  **public static DataSet ExecuteDataset(SQLiteCommand cmd)**  **{**  **if (cmd.Connection.State == ConnectionState.Closed)**  **cmd.Connection.Open();**  **DataSet ds = new DataSet();**  **SQLiteDataAdapter da = new SQLiteDataAdapter(cmd);**  **da.Fill(ds);**  **da.Dispose();**  **cmd.Connection.Close();**  **cmd.Dispose();**  **return ds;**  **}**  **/// <summary>**  **/// Executes the dataset in a SQLite Transaction**  **/// </summary>**  **/// <param name="transaction">SQLiteTransaction. Transaction consists of Connection, Transaction, /// and Command, all of which must be created prior to making this method call. </param>**  **/// <param name="commandText">Command text.</param>**  **/// <param name="commandParameters">Sqlite Command parameters.</param>**  **/// <returns>DataSet</returns>**  **/// <remarks>user must examine Transaction Object and handle transaction.connection .Close, etc.</remarks>**  **public static DataSet ExecuteDataset(SQLiteTransaction transaction, string commandText, params SQLiteParameter[] commandParameters)**  **{**  **if (transaction == null) throw new ArgumentNullException("transaction");**  **if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rolled back or committed, please provide an open transaction.", "transaction");**  **IDbCommand cmd = transaction.Connection.CreateCommand();**  **cmd.CommandText = commandText;**  **foreach (SQLiteParameter parm in commandParameters)**  **{**  **cmd.Parameters.Add(parm);**  **}**  **if (transaction.Connection.State == ConnectionState.Closed)**  **transaction.Connection.Open();**  **DataSet ds = ExecuteDataset((SQLiteCommand)cmd);**  **return ds;**  **}**  **/// <summary>**  **/// Executes the dataset with Transaction and object array of parameter values.**  **/// </summary>**  **/// <param name="transaction">SQLiteTransaction. Transaction consists of Connection, Transaction, /// and Command, all of which must be created prior to making this method call. </param>**  **/// <param name="commandText">Command text.</param>**  **/// <param name="commandParameters">object[] array of parameter values.</param>**  **/// <returns>DataSet</returns>**  **/// <remarks>user must examine Transaction Object and handle transaction.connection .Close, etc.</remarks>**  **public static DataSet ExecuteDataset(SQLiteTransaction transaction, string commandText, object[] commandParameters)**  **{**  **if (transaction == null) throw new ArgumentNullException("transaction");**  **if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rolled back or committed, please provide an open transaction.", "transaction");**  **IDbCommand cmd = transaction.Connection.CreateCommand();**  **cmd.CommandText = commandText;**  **AttachParameters((SQLiteCommand)cmd, cmd.CommandText, commandParameters);**  **if (transaction.Connection.State == ConnectionState.Closed)**  **transaction.Connection.Open();**  **DataSet ds = ExecuteDataset((SQLiteCommand)cmd);**  **return ds;**  **}**  **#region UpdateDataset**  **/// <summary>**  **/// Executes the respective command for each inserted, updated, or deleted row in the DataSet.**  **/// </summary>**  **/// <remarks>**  **/// e.g.:**  **/// UpdateDataset(conn, insertCommand, deleteCommand, updateCommand, dataSet, "Order");**  **/// </remarks>**  **/// <param name="insertCommand">A valid SQL statement to insert new records into the data source</param>**  **/// <param name="deleteCommand">A valid SQL statement to delete records from the data source</param>**  **/// <param name="updateCommand">A valid SQL statement used to update records in the data source</param>**  **/// <param name="dataSet">The DataSet used to update the data source</param>**  **/// <param name="tableName">The DataTable used to update the data source.</param>**  **public static void UpdateDataset(SQLiteCommand insertCommand, SQLiteCommand deleteCommand, SQLiteCommand updateCommand, DataSet dataSet, string tableName)**  **{**  **if (insertCommand == null) throw new ArgumentNullException("insertCommand");**  **if (deleteCommand == null) throw new ArgumentNullException("deleteCommand");**  **if (updateCommand == null) throw new ArgumentNullException("updateCommand");**  **if (tableName == null || tableName.Length == 0) throw new ArgumentNullException("tableName");**  **// Create a SQLiteDataAdapter, and dispose of it after we are done**  **using (SQLiteDataAdapter dataAdapter = new SQLiteDataAdapter())**  **{**  **// Set the data adapter commands**  **dataAdapter.UpdateCommand = updateCommand;**  **dataAdapter.InsertCommand = insertCommand;**  **dataAdapter.DeleteCommand = deleteCommand;**  **// Update the dataset changes in the data source**  **dataAdapter.Update(dataSet, tableName);**  **// Commit all the changes made to the DataSet**  **dataSet.AcceptChanges();**  **}**  **}**  **#endregion**  **/// <summary>**  **/// ShortCut method to return IDataReader**  **/// NOTE: You should explicitly close the Command.connection you passed in as**  **/// well as call Dispose on the Command after reader is closed.**  **/// We do this because IDataReader has no underlying Connection Property.**  **/// </summary>**  **/// <param name="cmd">SQLiteCommand Object</param>**  **/// <param name="commandText">SQL Statement with optional embedded "@param" style parameters</param>**  **/// <param name="paramList">object[] array of parameter values</param>**  **/// <returns>IDataReader</returns>**  **public static IDataReader ExecuteReader(SQLiteCommand cmd, string commandText, object[] paramList)**  **{**  **if (cmd.Connection == null)**  **throw new ArgumentException("Command must have live connection attached.", "cmd");**  **cmd.CommandText = commandText;**  **AttachParameters(cmd, commandText, paramList);**  **if (cmd.Connection.State == ConnectionState.Closed)**  **cmd.Connection.Open();**  **IDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);**  **return rdr;**  **}**  **/// <summary>**  **/// Shortcut to ExecuteNonQuery with SqlStatement and object[] param values**  **/// </summary>**  **/// <param name="connectionString">SQLite Connection String</param>**  **/// <param name="commandText">Sql Statement with embedded "@param" style parameters</param>**  **/// <param name="paramList">object[] array of parameter values</param>**  **/// <returns></returns>**  **public static int ExecuteNonQuery(string connectionString, string commandText, params object[] paramList)**  **{**  **SQLiteConnection cn = new SQLiteConnection(connectionString);**  **SQLiteCommand cmd = cn.CreateCommand();**  **cmd.CommandText = commandText;**  **AttachParameters(cmd, commandText, paramList);**  **if (cn.State == ConnectionState.Closed)**  **cn.Open();**  **int result = cmd.ExecuteNonQuery();**  **cmd.Dispose();**  **cn.Close();**  **return result;**  **}**  **public static int ExecuteNonQuery(SQLiteConnection cn, string commandText, params object[] paramList)**  **{**  **SQLiteCommand cmd = cn.CreateCommand();**  **cmd.CommandText = commandText;**  **AttachParameters(cmd, commandText, paramList);**  **if (cn.State == ConnectionState.Closed)**  **cn.Open();**  **int result = cmd.ExecuteNonQuery();**  **cmd.Dispose();**  **cn.Close();**  **return result;**  **}**  **/// <summary>**  **/// Executes non-query sql Statment with Transaction**  **/// </summary>**  **/// <param name="transaction">SQLiteTransaction. Transaction consists of Connection, Transaction, /// and Command, all of which must be created prior to making this method call. </param>**  **/// <param name="commandText">Command text.</param>**  **/// <param name="paramList">Param list.</param>**  **/// <returns>Integer</returns>**  **/// <remarks>user must examine Transaction Object and handle transaction.connection .Close, etc.</remarks>**  **public static int ExecuteNonQuery(SQLiteTransaction transaction, string commandText, params object[] paramList)**  **{**  **if (transaction == null) throw new ArgumentNullException("transaction");**  **if (transaction != null && transaction.Connection == null) throw new ArgumentException("The transaction was rolled back or committed, please provide an open transaction.", "transaction");**  **IDbCommand cmd = transaction.Connection.CreateCommand();**  **cmd.CommandText = commandText;**  **AttachParameters((SQLiteCommand)cmd, cmd.CommandText, paramList);**  **if (transaction.Connection.State == ConnectionState.Closed)**  **transaction.Connection.Open();**  **int result = cmd.ExecuteNonQuery();**  **cmd.Dispose();**  **return result;**  **}**  **/// <summary>**  **/// Executes the non query.**  **/// </summary>**  **/// <param name="cmd">CMD.</param>**  **/// <returns></returns>**  **public static int ExecuteNonQuery(IDbCommand cmd)**  **{**  **if (cmd.Connection.State == ConnectionState.Closed)**  **cmd.Connection.Open();**  **int result = cmd.ExecuteNonQuery();**  **cmd.Connection.Close();**  **cmd.Dispose();**  **return result;**  **}**  **/// <summary>**  **/// Shortcut to ExecuteScalar with Sql Statement embedded params and object[] param values**  **/// </summary>**  **/// <param name="connectionString">SQLite Connection String</param>**  **/// <param name="commandText">SQL statment with embedded "@param" style parameters</param>**  **/// <param name="paramList">object[] array of param values</param>**  **/// <returns></returns>**  **public static object ExecuteScalar(string connectionString, string commandText, params object[] paramList)**  **{**  **SQLiteConnection cn = new SQLiteConnection(connectionString);**  **SQLiteCommand cmd = cn.CreateCommand();**  **cmd.CommandText = commandText;**  **AttachParameters(cmd, commandText, paramList);**  **if (cn.State == ConnectionState.Closed)**  **cn.Open();**  **object result = cmd.ExecuteScalar();**  **cmd.Dispose();**  **cn.Close();**  **return result;**  **}**  **/// <summary>**  **/// Execute XmlReader with complete Command**  **/// </summary>**  **/// <param name="command">SQLite Command</param>**  **/// <returns>XmlReader</returns>**  **public static XmlReader ExecuteXmlReader(IDbCommand command)**  **{ // open the connection if necessary, but make sure we**  **// know to close it when we�re done.**  **if (command.Connection.State != ConnectionState.Open)**  **{**  **command.Connection.Open();**  **}**  **// get a data adapter**  **SQLiteDataAdapter da = new SQLiteDataAdapter((SQLiteCommand)command);**  **DataSet ds = new DataSet();**  **// fill the data set, and return the schema information**  **da.MissingSchemaAction = MissingSchemaAction.AddWithKey;**  **da.Fill(ds);**  **// convert our dataset to XML**  **StringReader stream = new StringReader(ds.GetXml());**  **command.Connection.Close();**  **// convert our stream of text to an XmlReader**  **return new XmlTextReader(stream);**  **}**  **/// <summary>**  **/// Parses parameter names from SQL Statement, assigns values from object array , /// and returns fully populated ParameterCollection.**  **/// </summary>**  **/// <param name="commandText">Sql Statement with "@param" style embedded parameters</param>**  **/// <param name="paramList">object[] array of parameter values</param>**  **/// <returns>SQLiteParameterCollection</returns>**  **/// <remarks>Status experimental. Regex appears to be handling most issues. Note that parameter object array must be in same ///order as parameter names appear in SQL statement.</remarks>**  **private static SQLiteParameterCollection AttachParameters(SQLiteCommand cmd, string commandText, params object[] paramList)**  **{**  **if (paramList == null || paramList.Length == 0) return null;**  **SQLiteParameterCollection coll = cmd.Parameters;**  **string parmString = commandText.Substring(commandText.IndexOf("@"));**  **// pre-process the string so always at least 1 space after a comma.**  **parmString = parmString.Replace(",", " ,");**  **// get the named parameters into a match collection**  **string pattern = @"(@)\S\*(.\*?)\b";**  **Regex ex = new Regex(pattern, RegexOptions.IgnoreCase);**  **MatchCollection mc = ex.Matches(parmString);**  **string[] paramNames = new string[mc.Count];**  **int i = 0;**  **foreach (Match m in mc)**  **{**  **paramNames[i] = m.Value;**  **i++;**  **}**  **// now let's type the parameters**  **int j = 0;**  **Type t = null;**  **foreach (object o in paramList)**  **{**  **t = o.GetType();**  **SQLiteParameter parm = new SQLiteParameter();**  **switch (t.ToString())**  **{**  **case ("DBNull"):**  **case ("Char"):**  **case ("SByte"):**  **case ("UInt16"):**  **case ("UInt32"):**  **case ("UInt64"):**  **throw new SystemException("Invalid data type");**  **case ("System.String"):**  **parm.DbType = DbType.String;**  **parm.ParameterName = paramNames[j];**  **parm.Value = (string)paramList[j];**  **coll.Add(parm);**  **break;**  **case ("System.Byte[]"):**  **parm.DbType = DbType.Binary;**  **parm.ParameterName = paramNames[j];**  **parm.Value = (byte[])paramList[j];**  **coll.Add(parm);**  **break;**  **case ("System.Int32"):**  **parm.DbType = DbType.Int32;**  **parm.ParameterName = paramNames[j];**  **parm.Value = (int)paramList[j];**  **coll.Add(parm);**  **break;**  **case ("System.Boolean"):**  **parm.DbType = DbType.Boolean;**  **parm.ParameterName = paramNames[j];**  **parm.Value = (bool)paramList[j];**  **coll.Add(parm);**  **break;**  **case ("System.DateTime"):**  **parm.DbType = DbType.DateTime;**  **parm.ParameterName = paramNames[j];**  **parm.Value = Convert.ToDateTime(paramList[j]);**  **coll.Add(parm);**  **break;**  **case ("System.Double"):**  **parm.DbType = DbType.Double;**  **parm.ParameterName = paramNames[j];**  **parm.Value = Convert.ToDouble(paramList[j]);**  **coll.Add(parm);**  **break;**  **case ("System.Decimal"):**  **parm.DbType = DbType.Decimal;**  **parm.ParameterName = paramNames[j];**  **parm.Value = Convert.ToDecimal(paramList[j]);**  **break;**  **case ("System.Guid"):**  **parm.DbType = DbType.Guid;**  **parm.ParameterName = paramNames[j];**  **parm.Value = (System.Guid)(paramList[j]);**  **break;**  **case ("System.Object"):**  **parm.DbType = DbType.Object;**  **parm.ParameterName = paramNames[j];**  **parm.Value = paramList[j];**  **coll.Add(parm);**  **break;**  **default:**  **throw new SystemException("Value is of unknown data type");**  **} // end switch**  **j++;**  **}**  **return coll;**  **}**  **/// <summary>**  **/// Executes non query typed params from a DataRow**  **/// </summary>**  **/// <param name="command">Command.</param>**  **/// <param name="dataRow">Data row.</param>**  **/// <returns>Integer result code</returns>**  **public static int ExecuteNonQueryTypedParams(IDbCommand command, DataRow dataRow)**  **{**  **int retVal = 0;**  **// If the row has values, the store procedure parameters must be initialized**  **if (dataRow != null && dataRow.ItemArray.Length > 0)**  **{**  **// Set the parameters values**  **AssignParameterValues(command.Parameters, dataRow);**  **retVal = ExecuteNonQuery(command);**  **}**  **else**  **{**  **retVal = ExecuteNonQuery(command);**  **}**  **return retVal;**  **}**  **/// <summary>**  **/// This method assigns dataRow column values to an IDataParameterCollection**  **/// </summary>**  **/// <param name="commandParameters">The IDataParameterCollection to be assigned values</param>**  **/// <param name="dataRow">The dataRow used to hold the command's parameter values</param>**  **/// <exception cref="System.InvalidOperationException">Thrown if any of the parameter names are invalid.</exception>**  **protected internal static void AssignParameterValues(IDataParameterCollection commandParameters, DataRow dataRow)**  **{**  **if (commandParameters == null || dataRow == null)**  **{**  **// Do nothing if we get no data**  **return;**  **}**  **DataColumnCollection columns = dataRow.Table.Columns;**  **int i = 0;**  **// Set the parameters values**  **foreach (IDataParameter commandParameter in commandParameters)**  **{**  **// Check the parameter name**  **if (commandParameter.ParameterName == null ||**  **commandParameter.ParameterName.Length <= 1)**  **throw new InvalidOperationException(string.Format(**  **"Please provide a valid parameter name on the parameter #{0}, the ParameterName property has the following value: '{1}'.",**  **i, commandParameter.ParameterName));**  **if (columns.Contains(commandParameter.ParameterName))**  **commandParameter.Value = dataRow[commandParameter.ParameterName];**  **else if (columns.Contains(commandParameter.ParameterName.Substring(1)))**  **commandParameter.Value = dataRow[commandParameter.ParameterName.Substring(1)];**  **i++;**  **}**  **}**  **/// <summary>**  **/// This method assigns dataRow column values to an array of IDataParameters**  **/// </summary>**  **/// <param name="commandParameters">Array of IDataParameters to be assigned values</param>**  **/// <param name="dataRow">The dataRow used to hold the stored procedure's parameter values</param>**  **/// <exception cref="System.InvalidOperationException">Thrown if any of the parameter names are invalid.</exception>**  **protected void AssignParameterValues(IDataParameter[] commandParameters, DataRow dataRow)**  **{**  **if ((commandParameters == null) || (dataRow == null))**  **{**  **// Do nothing if we get no data**  **return;**  **}**  **DataColumnCollection columns = dataRow.Table.Columns;**  **int i = 0;**  **// Set the parameters values**  **foreach (IDataParameter commandParameter in commandParameters)**  **{**  **// Check the parameter name**  **if (commandParameter.ParameterName == null ||**  **commandParameter.ParameterName.Length <= 1)**  **throw new InvalidOperationException(string.Format(**  **"Please provide a valid parameter name on the parameter #{0}, the ParameterName property has the following value: '{1}'.",**  **i, commandParameter.ParameterName));**  **if (columns.Contains(commandParameter.ParameterName))**  **commandParameter.Value = dataRow[commandParameter.ParameterName];**  **else if (columns.Contains(commandParameter.ParameterName.Substring(1)))**  **commandParameter.Value = dataRow[commandParameter.ParameterName.Substring(1)];**  **i++;**  **}**  **}**  **/// <summary>**  **/// This method assigns an array of values to an array of IDataParameters**  **/// </summary>**  **/// <param name="commandParameters">Array of IDataParameters to be assigned values</param>**  **/// <param name="parameterValues">Array of objects holding the values to be assigned</param>**  **/// <exception cref="System.ArgumentException">Thrown if an incorrect number of parameters are passed.</exception>**  **protected void AssignParameterValues(IDataParameter[] commandParameters, params object[] parameterValues)**  **{**  **if ((commandParameters == null) || (parameterValues == null))**  **{**  **// Do nothing if we get no data**  **return;**  **}**  **// We must have the same number of values as we pave parameters to put them in**  **if (commandParameters.Length != parameterValues.Length)**  **{**  **throw new ArgumentException("Parameter count does not match Parameter Value count.");**  **}**  **// Iterate through the IDataParameters, assigning the values from the corresponding position in the**  **// value array**  **for (int i = 0, j = commandParameters.Length, k = 0; i < j; i++)**  **{**  **if (commandParameters[i].Direction != ParameterDirection.ReturnValue)**  **{**  **// If the current array value derives from IDataParameter, then assign its Value property**  **if (parameterValues[k] is IDataParameter)**  **{**  **IDataParameter paramInstance;**  **paramInstance = (IDataParameter)parameterValues[k];**  **if (paramInstance.Direction == ParameterDirection.ReturnValue)**  **{**  **paramInstance = (IDataParameter)parameterValues[++k];**  **}**  **if (paramInstance.Value == null)**  **{**  **commandParameters[i].Value = DBNull.Value;**  **}**  **else**  **{**  **commandParameters[i].Value = paramInstance.Value;**  **}**  **}**  **else if (parameterValues[k] == null)**  **{**  **commandParameters[i].Value = DBNull.Value;**  **}**  **else**  **{**  **commandParameters[i].Value = parameterValues[k];**  **}**  **k++;**  **}**  **}**  **}**  **}** |