ЛАБОРАТОРНА РОБОТА

“Використання ADO.NET, що працюють з базами даних”

Введение

Цель работы: Познакомиться с языком программирования **C#**, средствами разработки среды **Visual Studio .Net 2003** и с использованием сервера баз данных **SQL Server 2000**, на примере создания клиентского Windows-приложения, работающего с БД.

Задача участника: Написать приложение на **Windows Forms** позволяющее соединяться с удалённой базой данных, просматривать её содержимое, и выполнять стандартные SQL-запросы.

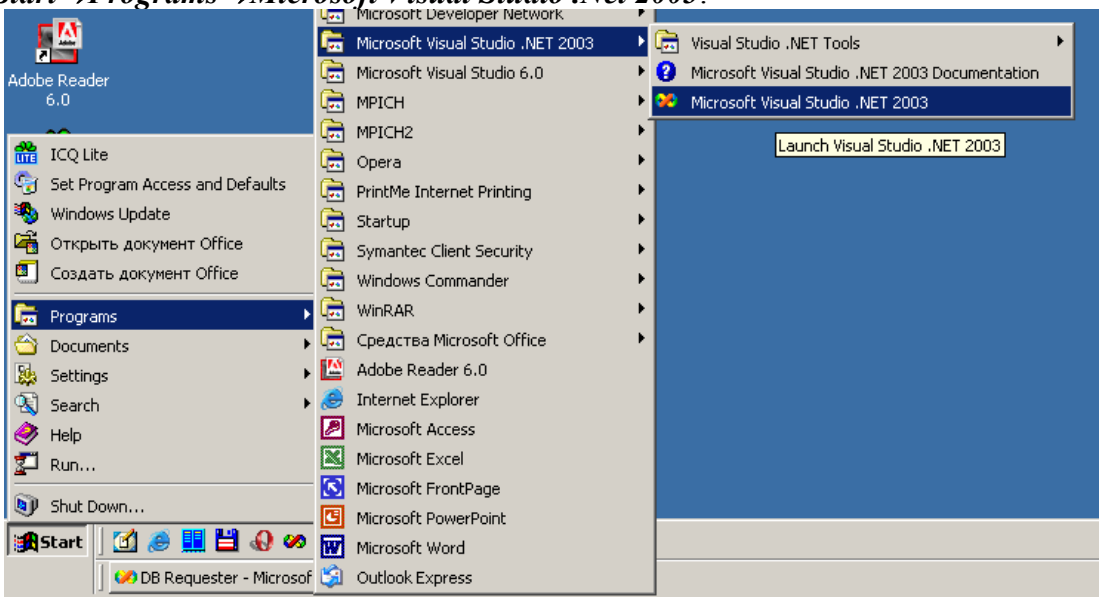
Инструменты: Для выполнения поставленной задачи Вам должен быть предоставлен компьютер с установленной средой разработки **Visual Studio .Net 2003** и доступ к компьютеру с установленным сервером **SQL Server 2000**, на котором предварительно создана база данных.

Если в ходе написания программы у вас возникнут вопросы, ОБЯЗАТЕЛЬНО обращайтесь к помощникам!

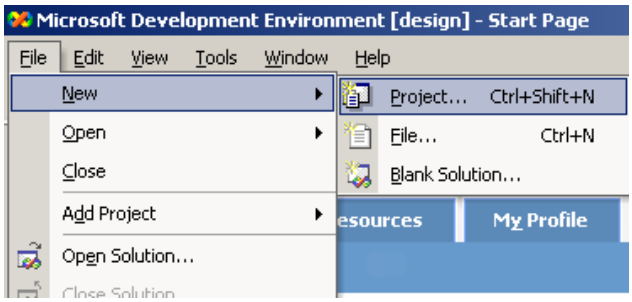
ЧАСТЬ 1: Создание WINDOWS- приложения, работающего с БД

Шаг-1: Создание проекта

Перед тем как начать писать текст программы, Вам необходимо запустить среду разработки Visual Studio .Net 2003. Для этого вы должны выбрать закладку с названием Visual Studio .Net 2003 из меню ***Start→Programs→Microsoft Visual Studio .Net 2003***:



После этого откроется среда разработки Visual Studio .Net 2003.

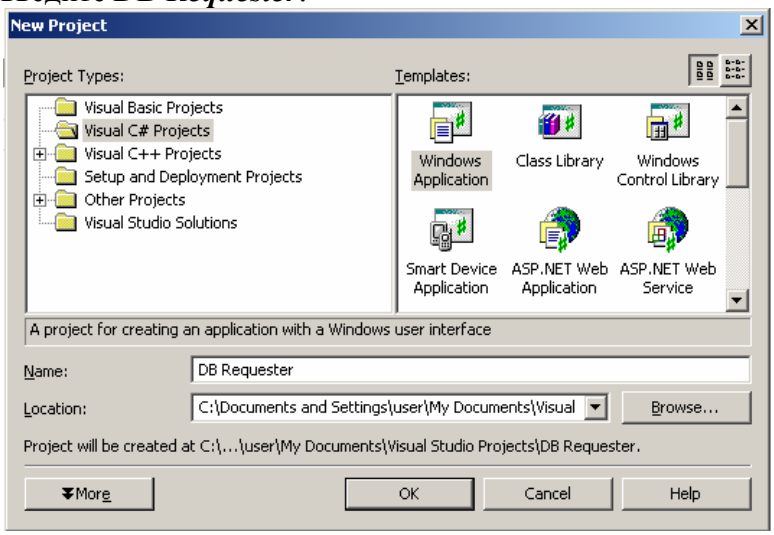


Дальше вам необходимо создать новый проект. Для этого выберете элемент **Project** из меню:

File→New→Project:

С появившимся меню вам нужно проделать следующие действия:

1. В поле Project Types выберите ***Visual C# Projects***;
2. В поле Templates выберите ***Windows Application***;
3. В поле Name введите ***DB Requester***:



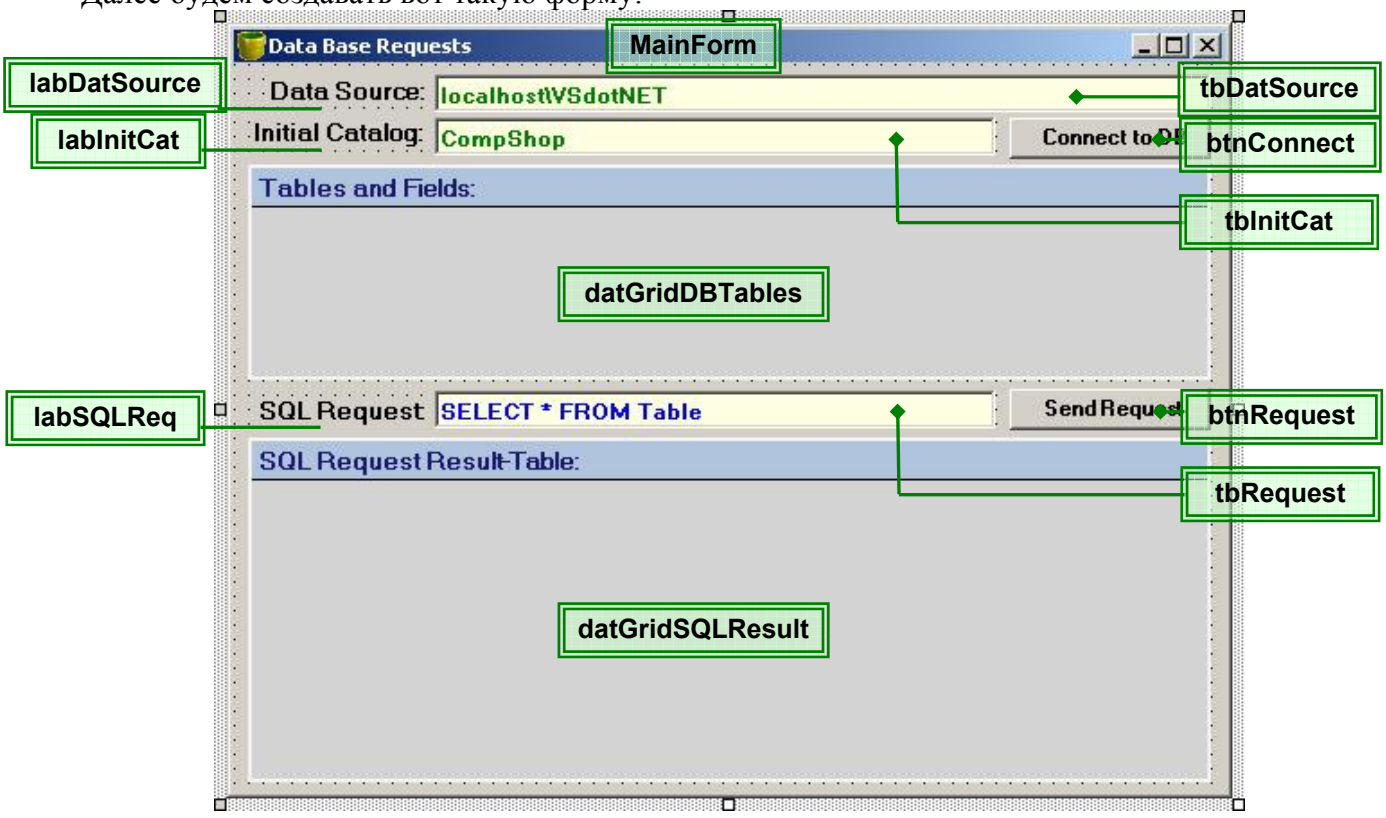
4. Нажмите **Ок**.

Шаг-2: Настройка компонент главной формы

Перед вами появится пустой шаблон формы (окна будущего приложения), создаваемый по умолчанию средой разработки Visual Studio .Net 2003. Теперь вам нужно установить на вашу форму компоненты и настроить их. Для написания более или менее качественной программы вам понадобятся как минимум следующие компоненты:

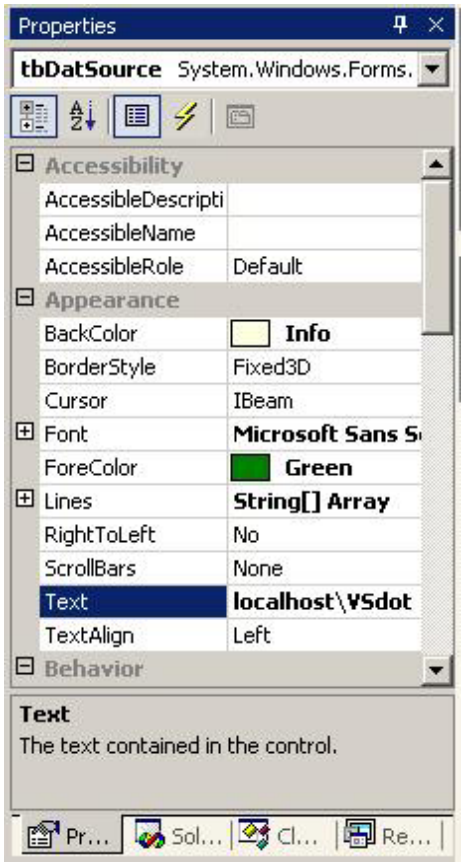
1. Две кнопки (Класс ***Button***)
2. Две компоненты, отображающие табличные данные (Класс ***DataGrid***)
3. Три поля для ввода текстовых данных (Класс ***TextBox***)
4. Три надписи для обозначения полей ввода (Класс ***Label***)

Далее будем создавать вот такую форму:



Для начала растяните форму главного окна Вашего приложения. Поскольку, скорее всего все эти компоненты не поместятся на стандартной начальной форме, а если и поместятся, то будут выглядеть не информативно. Это можно сделать как минимум двумя способами:

- Задать новые значения полю Size в формате (x;y) в списке Properties, где x – ширина(**Width**), а y – высота(**Height**). Предлагаемые значения x=550,y=450;
- Схватив форму за правый нижний край, растянуть её до нужных вам размеров.



Для того, чтобы добавить новую компоненту нужно выбрать её в списке **Toolbox** (в левой части окна), а потом щёлкнуть левой кнопкой по вашей форме. На месте щелчка должна появиться новая компонента.

Далее будет описан порядок создания (через "**ToolBox**") и настройки компонент с помощью окна свойств "**Properties**" (по умолчанию оно находится в правой части окна среды Visual Studio).

Компоненты обозначаются их именами, которые также подписаны на изображении формы; эти названия следует ввести в свойство **Name** каждой из компонент.

Разместите каждую из компонент, описанных ниже, на форме Вашего будущего приложения (с помощью **ToolBox**), как показано на ранее приведённом рисунке. Затем настройте компоненты, задавая значения перечисленных свойств следующих компонент:

1. Свойства класса **Form** – формы главного окна приложения:

Свойство	Значение
Name	MainForm
Text	Data Base Requests
MinimumSize	300; 350
Size	560; 432
StartPosition	CenterScreen

2. Свойства компоненты **Label** – подпись к полю ввода источника данных:

Свойство	Значение
Name	labDatSource
Text	Data Source:

3. Свойства компоненты **TextBox** – текстовое поле ввода источника данных:

Свойство	Значение
Name	tbDatSource
Text	localhost\VSdotNET
Anchor	Top, Left, Right
BackColor	Info
ForeColor	Green

4. Свойства компоненты **Label** – подпись к полю ввода имени базы данных:

Свойство	Значение
Name	labInitCat
Text	Initial Catalog:

5. Свойства компоненты **TextBox** - текстовое поле ввода наименования базы данных:

Свойство	Значение
Name	tbInitCat
Text	CompShop
Anchor	Top, Left, Right
BackColor	Info
ForeColor	Green

6. Свойства компоненты **Button** – кнопка соединения с указанной базой данных:

Свойство	Значение
Name	btnConnect
Text	Connect to DB
Anchor	Top, Right

7. Свойства компоненты **DataGrid** – отображает для таблиц базы данных соответствующие им поля (с типами) в табличной форме:

Свойство	Значение
Name	DatGridDBTables
CaptionText	Tables and Fields:
ReadOnly	True
Enabled	False
Anchor	Top, Left, Right
TableStyles	Щёлкните по кнопке  и выполните настройку:

В появившемся диалоговом окне добавьте новый стиль таблицы, нажав на кнопку **“Add”**. Выполните настройку появившегося объекта стиля таблицы:

Свойство	Значение
Name	TabStyle
MappingName	TabFields
Enabled	False
GridColumnStyles	Щёлкните по кнопке  и выполните настройку:

Во вновь появившемся диалоговом окне добавьте два столбца, дважды нажав кнопку **“Add ”**. Для каждого из столбцов, поочерёдно, введите их свойства:

Свойство	Значение
Свойства первого столбца:	
Name	ColTables
MappingName	Tables
HeaderText	Table Name
Width	80
Свойства второго столбца:	
Name	ColFields
MappingName	Fields
HeaderText	Field Names in Table
Width	280
ReadOnly	True

При желании, Вы можете изменить цветовое оформление таблицы, нажав правую кнопку мыши на установленной компоненте **DataGrid** и выбрав из меню пункт “**AutoFormat**”. В появившемся окне выберите цветовую схему **“Professional 4”** или другую, на Ваш вкус.

8. Свойства компоненты **Label** – подпись к полю ввода строки SQL-запроса:

Свойство	Значение
Name	labSQLReq
Text	SQL Request:

9. Свойства компоненты **TextBox** – поле ввода строки SQL-запроса:

Свойство	Значение
Name	tbRequest
Text	SELECT * FROM Table
Anchor	Top, Left, Right
Enabled	False
BackColor	Info
ForeColor	Blue

10. Свойства компоненты **Button** – кнопка выполнения указанного SQL-запроса к БД;

Свойство	Значение
Name	btnRequest
Text	Send Request
Anchor	Top, Right
Enabled	False

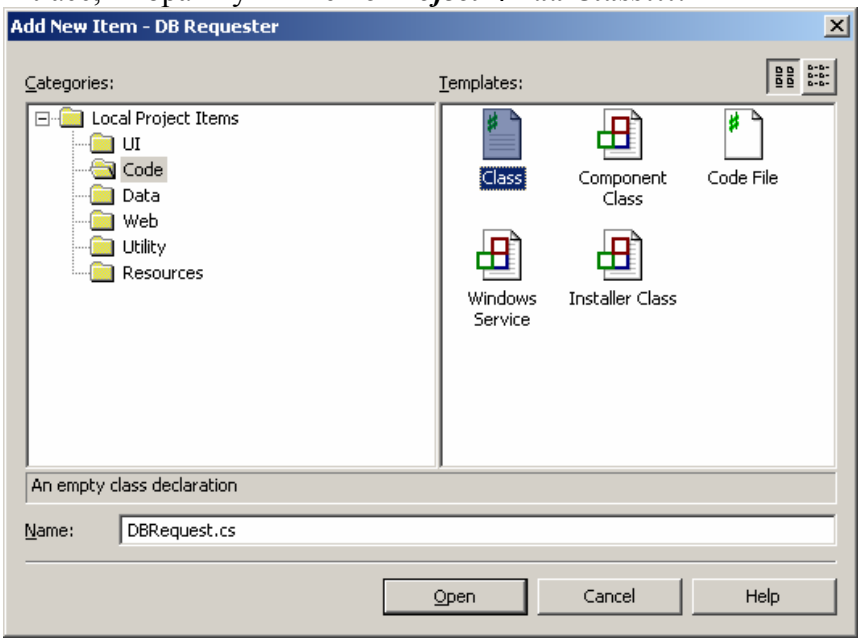
11. Свойства компоненты **DataGrid** – отображает таблицу, являющуюся результатом выполнения SQL-запроса сервером баз данных:

Свойство	Значение
Name	datGridSQLResult
CaptionText	SQL Request Result-Table:
ReadOnly	True
Enabled	False
Anchor	Top, Left, Right

При желании, Вы можете изменить цветовое оформление таблицы, нажав правую кнопку мыши на установленной компоненте **DataGrid** и выбрав из меню пункт "**AutoFormat**". В появившемся окне выберите цветовую схему **"Professional 4"** или другую, на Ваш вкус.

Шаг-3: Пишем класс, осуществляющий соединение и выполнение запросов к базе данных

Создайте новый класс, выбрав пункт меню ***Project→Add Class…***:



В открывшемся диалоговом окне введите имя файла с исходным кодом класса:

"DBRequest.cs".

Откроется окно редактора исходного кода класса **DBRequest.**

В тело класса DBRequest добавьте поле типа **OleDbConnection** и его автоматическую инициализацию (выполняемую при создании экземпляра объекта DBRequest):

```
private OleDbConnection DBCon = new OleDbConnection();
```

Объект класса OleDbConnection – DBCon способен выполнять соединение пользовательского приложения с базой данных на SQL Server 2000.

Напишите метод **ConnectTo**, открывающий соединение с базой данных на сервере:

```
//************** Открывает Соединение с Базой Данных*****************
public void ConnectTo( string DataSource, string InitialCatalog )
{
  DBCon.ConnectionString = "Provider=SQLOLEDB;"+
                           "Integrated Security=SSPI;"+
                           "Data Source="+DataSource+";"+
                           "Initial Catalog="+InitialCatalog+";"+
                           "User ID=sa; Password=";
  try
  {
    DBCon.Open();
  }
  catch(Exception e)
  {
    throw e;
  }
}
```

Метод формирует строку с командой соединения с БД по передаваемым в функцию полям **DataSource** – строке с сетевым путём к компьютеру с установленным SQL Server 2000 или MSDE, а так же **InitialCatalog** – строке с названием каталога базы данных на сервере. Затем делается попытка открыть соединение с базой данных. В случае неудачи, возникающее исключение передаётся вышестоящей по стеку вызовов функции.

Напишите метод **Disconnect**, выполняющий разрыв открытого соединения с БД:

```
//************** Закрывает Соединение с Базой Данных *****************
public void Disconnect()
{
  try
  {
    if (DBCon.State == ConnectionState.Open)
      DBCon.Close();
  }
  catch
  {
  }
}
```

В функции делается проверка – если соединение открыто, то выполняется его закрытие.

Добавьте деструктор класса, выполняющий закрытие, открытых соединений:

```
//************************* Деструктор ******************************
~DBRequest()
{
    Disconnect();
}
```

Теперь напишем пару методов, реализующих основную функциональность нашего приложения. Первая из этих функций – **GetTableFields**, для запрашиваемой таблицы из базы (по её имени **TableName**) возвращает строку с перечислением полей и их типов:

```
//************* Получение Списка Полей Некоторой Таблицы **************
public string GetTableFields( string TableName )
{
  if (DBCon.State == ConnectionState.Open)
```

```
    {
      DataTable CurTable = new DataTable( "ConnectedTab" );
      OleDbDataAdapter DBAdapt;
      try
      {
        DBAdapt = new OleDbDataAdapter( "SELECT * FROM "+TableName, DBCon );
        DBAdapt.Fill( CurTable );
      }
      catch(Exception e)
      {
        throw e;
      }
      string ResStr = "";
      int ColCount = CurTable.Columns.Count;
      for(int i=0; i<ColCount; i++)
      {
        string StrCon = ", ";
        if (i == ColCount-1) StrCon = ";";
        ResStr += CurTable.Columns[i].ColumnName + "[" +
                  CurTable.Columns[i].DataType.Name + "]"+ StrCon;
      }
      return ResStr;
    }
    else
      return null;
  }
```

В теле функции проверяется, открыто ли соединение с базой данных, если это так то выполняются действия, формирующие строку со списком полей таблицы:

1. Создаётся объект класса **DataTable**, представляющего некоторую таблицу;
2. Создаётся объект класса **OleDbDataAdapter**, осуществляющего обработку всевозможных SQL запросов к базе данных. В конструктор передаётся строка с SQL запросом, выполняющим выборку всех строк из таблицы с именем **TableName** и ссылка на объект соединения с базой;
3. Вызов метода **Fill** объекта **DBAdapt** заполняет созданный нами объект **CurTable** данными таблицы БД, загруженными с сервера. Выполнение функций производится с перехватом возможных исключений;
4. Затем, в цикле **for** производится непосредственно формирование строки **ResStr** со списком полей загруженной нами таблицы. Доступ к полям производится через поле **Columns** класса **DataTable**, представляющего объект списка **ArrayList**.

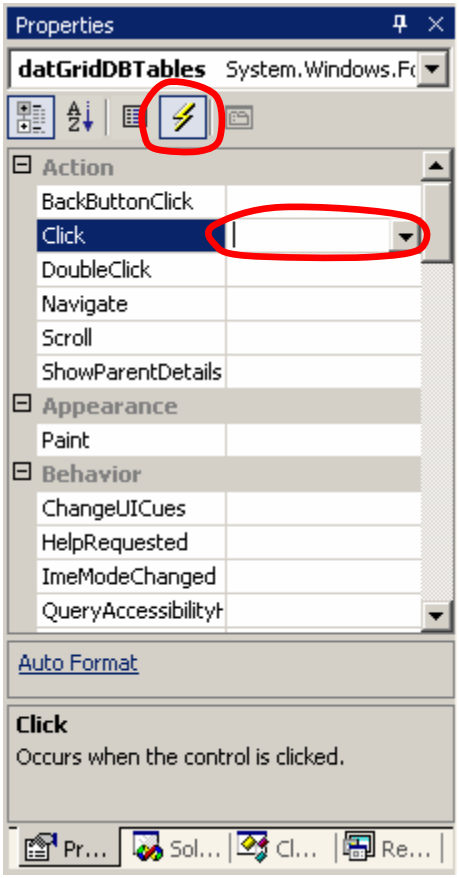
Наконец, напишем метод **SqlRequest**, выполняющий произвольный SQL-запрос к базе данных и возвращающий таблицу с данными по запросу:

```
  //**************** Выполнение некоторого SQL-запроса ***************
  public DataTable SQLRequest( string RequestStr )
  {
    if (DBCon.State == ConnectionState.Open)
    {
      DataTable ResultTab = new DataTable( "SQLresult" );
      OleDbDataAdapter DBAdapt;
      try
      {
        DBAdapt = new OleDbDataAdapter( RequestStr, DBCon );
        DBAdapt.Fill( ResultTab );
      }
      catch(Exception e)
      {
        throw e;
      }
      return ResultTab;
    }
    else
```

```
        return null;
    }
```

Этот метод очень прост и во многом похож на предыдущий. В нём опять же создаётся объект таблицы **DataTable**, затем создаётся дата-адаптер **DBAdapt** по переданной строке с SQL-запросом и ссылке на объект соединения **DBCon** и производится заполнение таблицы данными из базы. Выполняется перехват возникающих исключений при выполнении команд программы. После выполнения функция возвращает ссылку на созданную таблицу с данными по запросу из базы – **ResultTab**;

Теперь наш класс **DBRequest** полностью написан и, пришло время, перейти к подключению его методов к интерфейсу нашей программы.



Шаг-4: Подключение методов класса DBRequest к обработчикам событий компонент интерфейса программы

Создание обработчиков событий компонент интерфейса нашей программы в **Visual Studio** производится через окно **Properties** среды разработки. Чтобы создать обработчик события, зайдите в окно дизайнера Вашей формы, щёлкните левой кнопкой мыши по одной из компонент и в окне **Properties** нажмите на кнопку отображения списка событий компоненты. Для того, чтобы создать обработчик одного из событий, дважды кликните мышью на пустом поле справа от названия события.

Прежде чем начать создавать обработчики событий, добавьте несколько полей в класс формы **MainForm**:

```
    // Класс выполняющий соединение с БД и запрос к ней:
    public DBRequest UserReq;
    // Таблица-результат выполнения запроса:
    private DataTable RequestTab;
    // Таблица структуры таблиц из БД:
    private DataTable StructTab;
    // Название Таблицы, обработанной последний раз в StructTab_OnRowChanged
    private string LastTabName;
```

Итак, теперь создайте обработчик события загрузки главной формы нашего приложения: **MainForm_Load**. Это событие происходит при запуске приложения и загрузке данных формы в память компьютера. В появившемся теле обработчика события выполним инициализацию начальных данных программы:

```
    //********************** Загрузка формы ****************************
    private void MainForm_Load(object sender, System.EventArgs e)
    {
      // Создаём класс взаимодействия с Базой Данных
      UserReq = new DBRequest();
      // Создаём таблицу и добавляем в неё столбцы:
      StructTab = new DataTable( "TabFields" );
      DataColumn NewDatCol = new DataColumn("Tables", Type.GetType("System.String"));
      NewDatCol.AllowDBNull = false;
      NewDatCol.Unique = true;
      StructTab.Columns.Add( NewDatCol );
      NewDatCol = new DataColumn( "Fields", Type.GetType("System.String") );
      NewDatCol.AllowDBNull = false;
      NewDatCol.DefaultValue = "none;";
      StructTab.Columns.Add( NewDatCol );
      datGridDBTables.DataSource = StructTab;
```

```
      datGridDBTables.ReadOnly = false;
      datGridSQLResult.DataSource = RequestTab;
      // Подключаем к таблице обработчик события изменения строки:
      StructTab.RowChanged += new DataRowChangeEventHandler( StructTab_OnRowChanged);
    }
```

В данном методе создаётся объект нашего класса **DBRequest**, и таблица **StructTab,** хранящая названия таблиц базы и соответствующие им списки полей таблицы. В методе **MainForm_Load** создаются столбцы этой таблицы, выполняется их настройка и они добавляются в таблицу **StructTab**. Таблица **StructTab** устанавливается в качестве источника данных для визуальной компоненты **datGridDBTables**, которая будет отображать данные этой таблицы на форме. Аналогично для компоненты datGridSQLResult, в качестве источника данных устанавливается таблица **RequestTab**. Так же к таблице **StructTab** подключается обработчик события изменения строки **StructTab.RowChanged.**

Напишем код обработчика события изменения строки в таблице: **StructTab.RowChanged:**

```
    //*************** Обработчик изменения строки в БД *****************
    private void StructTab_OnRowChanged( object sender, DataRowChangeEventArgs e )
    {
      try
      {
        if (LastTabName != (string) e.Row["Tables"])
        {
          LastTabName = (string) e.Row["Tables"];
          string Fields   = UserReq.GetTableFields( LastTabName );
          e.Row["Fields"] = Fields;
        }
      }
      catch(Exception ex)
      {
        MessageBox.Show( this, ex.Message, "Connection Error", MessageBoxButtons.OK,
MessageBoxIcon.Error );
      }
    }
```

В теле метода извлекается название таблицы в базе из изменённой строки **e.Row["Tables"]** и метод нашего класса **DBRequest.GetTableFields** формирует строку со списком полей из подключенной базы, которая записывается в столбец **“Fields”** изменённой строки: **e.Row["Fields"] = Fields**. В случае возникновения исключения создаётся диалоговое окно с сообщением об ошибке.

Добавьте обработчик события кнопки **btnConnect** на её нажатие: **btnConnect_Click**. В этом обработчике выполняется попытка соединения с базой данных и инициализация компонент. Введите следующее тело метода:

```
//*************** Попытка Соединения с Базой Данных *****************
    private void btnConnect_Click(object sender, System.EventArgs e)
    {
      bool Connected = false;
      Cursor = Cursors.WaitCursor;
      // Пробуем соединиться с БД:
      try
      {
        UserReq.Disconnect();
        UserReq.ConnectTo( tbDatSource.Text, tbInitCat.Text );
        Connected = true;
      }
      catch( Exception e1 )
      {
        MessageBox.Show( this, e1.Message, "Connection Error", MessageBoxButtons.OK,
MessageBoxIcon.Error );
        Connected = false;
      }
```

```
      if (Connected)
      {
        // Открываем доступ к управлению выполнением запросов:
        tbRequest.Enabled        = true;
        btnRequest.Enabled       = true;
        datGridSQLResult.Enabled = true;
        datGridDBTables.Enabled  = true;
      }
      else
      {// Закрываем доступ к управлению выполнением запросов:
        tbRequest.Enabled        = false;
        btnRequest.Enabled       = false;
        datGridSQLResult.Enabled = false;
        datGridDBTables.Enabled  = false;
      }
      Cursor = Cursors.Arrow;
      try
      {
        StructTab.Clear();
        RequestTab.Clear();
      }
      catch
      {
      }
    }
```

Обработчик опять же использует вызов уже написанного метода **DBRequest.ConnectTo**, производящего попытку соединения с базой данных. Производится обработка исключений с выводом сообщения об ошибке. И если соединение устанавливается, то к визуальным компонентам открывается доступ для ввода. Так же очищаются таблицы **StructTab** и **RequestTab.**

Добавьте обработчик события нажатия на кнопку выполнения SQL-запроса к базе: **btnRequest_Click:**

```
    //*************** Выполнение Заданного SQL-запроса ******************
    private void btnRequest_Click(object sender, System.EventArgs e)
    {
      Cursor = Cursors.WaitCursor;
      try
      {
        RequestTab = UserReq.SQLRequest( tbRequest.Text );
        datGridSQLResult.DataSource = RequestTab;
      }
      catch( Exception ex )
      {
        MessageBox.Show( this, ex.Message, "Request Error", MessageBoxButtons.OK,
MessageBoxIcon.Error );
      }
      Cursor = Cursors.Arrow;
    }
```

В теле метода, основной, является строка с вызовом написанного нами метода DBRequest.SQLRequest, который производит выполнение запроса, записанного в строке **tbRequest.Text** и сохранение его результатов в таблице **RequestTab.**

Добавьте обработчик события поднятия кнопки мыши над компонентой, отображающей названия таблиц и полей базы: **datGridDBTables_MouseUp**:

```
    //*************** Выводим Данные Выбранной Таблицы ******************
    private void datGridDBTables_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)
    {
      DataGrid.HitTestInfo HitInfo = datGridDBTables.HitTest(e.X, e.Y);
      if ((HitInfo.Row >= 0) && (HitInfo.Row < StructTab.Rows.Count))
      {
```

```
        tbRequest.Text = "SELECT * FROM " +(string) StructTab.Rows[ HitInfo.Row ][
"Tables" ];
        btnRequest_Click( this, null );
      }
```

}В данном методе производится отображение данных таблицы, соответствующей той строке, над которой произошло поднятие кнопки мыши – то есть той таблицы которую выбрал пользователем нажатием мыши. Действие отдельных операторов разберите самостоятельно.

И, наконец, добавьте обработчик события изменения размера таблицы для корректного изменения размеров столбцов этой таблицы:

```
    //****************** Изменение Размеров Таблицы *********************
    private void datGridDBTables_Resize(object sender, System.EventArgs e)
    {
      int ColTablesWidth = datGridDBTables.TableStyles[0].GridColumnStyles[0].Width;
      datGridDBTables.TableStyles[0].GridColumnStyles[1].Width =
        datGridDBTables.Width - ColTablesWidth - 57;
    }
```

Поздравляем Вас! Ваше приложение полностью написано. Остаётся выполнить компиляцию проекта и запустить исполняемый файл программы! Теперь перейдите к выполнению второй части лабораторной работы.

Часть 2: Выполнение запросов к базе данных с помощью написанного приложения

Шаг-5: Краткая справка по синтаксису языка SQL (Structured Query Language)

Если Вы ещё не знакомы с синтаксисом языка SQL, то в данной работе Вы познакомитесь с основными возможностями этого языка, прочитав эту краткую справку.

SQL – это язык структурированных запросов для реляционных баз данных. SQL включает в себя следующие языковые подмножества:

- ***Data Definition Language*** (**DDL**) - определения данных;
- ***Data Manipulation Language*** (**DML**) - манипулирования данными;
- ***Transaction Control Language*** (**TCL**) - управление транзакциями;
- ***Data Control Language*** (**DCL**) - управление привилегий;
- ***Cursor Control Language*** (**CCL**) - управление курсором;

В нашей лабораторной мы ограничимся командами языка DML. Язык DML содержит операторы, позволяющие выбирать, добавлять, удалять и модифицировать данные:

- **SELECT** – применяется для выборки данных;
- **INSERT** – применяется для добавления строк к таблице;
- **DELETE** – применяется для удаления строк из таблицы;
- **UPDATE** – применяется для исправления данных;

Написанная нами программа, предусматривает только выборку данных из уже существующей базы. Поэтому нам понадобится знание синтаксиса всего одной команды языка DML, а именно, команды SELECT. Полный синтаксис этой команды записывается в структурированной форме следующим образом:

SELECT [ALL | DISTINCT | TOP n] { * | expr_1 [**AS** c_alias_1] [, ... [, expr_k [**AS** c_alias_k]]]}
FROM table_name_1 [t_alias_1] [, ... [, table_name_n [t_alias_n]]]
[**WHERE** condition]
[**ORDER BY** name_of_attr_i [ASC|DESC] [, ... [, name_of_attr_j [ASC|DESC]]]];
[**GROUP BY** name_of_attr_i [,... [, name_of_attr_j]] [**HAVING** condition]]
[{UNION [ALL] **| INTERSECT | EXCEPT} SELECT** ...**]**
ALL | **DISTINCT** – выбирать все или только различные строки таблиц
TOP n [**PERCENT**] – выбирать только первые n [n процентов] строк

Основные ключевые слова команды SELECT выделены жирным шрифтом. Разберём применение этой команды на простом примере.

Рассмотрим базу данных из 3-х таблиц:

- ***Таблица*** **PART** – таблица товаров – сборных частей (болты, винты, гайки), которые характеризуются наименованием (PNAME) и ценой (PRICE), а так же идентификационным номером в таблице (PNO);
- ***Таблица*** **SUPPLIER** – таблица поставщиков, продающих эти товары. Каждый поставщик характеризуются именем (SNAME) и городом (CITY), а так же идентификационным номером в таблице (SNO);
- ***Таблица*** **SELLS** – таблица продаж, устанавливающее соответствие поставщиков и продаваемых ими товаров. Таблица хранит пары идентификационных номеров товаров (PNO) и поставщиков (SNO);

PNO	PNAME	PRICE
1	Screw	10
2	Nut	8
3	Bolt	15
4	Cam	25

↑ Таблица **PART**

SNO	SNAME	CITY
1	Smith	London
2	Jones	Paris
3	Adams	Vienna
4	Blake	Rome

↑ Таблица **SUPPLIER**

Таблица **SELLS** →

SNO	PNO
1	1
1	2
2	4
3	1
3	3
4	2
4	3
4	4

SELECT * FROM PART	Этот запрос возвратит все поля из таблицы PART.	
SELECT PNAME, PRICE * 2 AS D_PRICE FROM PART WHERE PRICE * 2 < 50;	PNAME | D_PRICE Screw | 20 Nut | 16	Какие товары при увеличенной вдвое стоимости будут стоить дешевле 50
SELECT S.SNAME, P.PNAME FROM SUPPLIER S, PART P, SELLS SE WHERE S.SNO = SE.SNO AND P.PNO = SE.PNO; AND S.SNAME <> 'Blake';	SNAME | PNAME Smith | Screw Smith | Nut Jones | Cam Adams | Screw Adams | Bolt	Запрос полей из двух таблиц Составть список "поставщик-товар" без Блейка
SELECT AVG(PRICE) AS AVG_PRICE FROM PART;	AVG_PRICE 14.5	Итоговые операторы: **AVG**(Column) - среднее значениие **COUNT** (Column) - количество **SUM** (Column) - сумма **MIN** (Column) - миним. значение **MAX** (Column) - максим. значение
SELECT COUNT(PNO) FROM PART	COUNT 4	
SELECT SUM(PRICE) FROM PART	SUM 58	

В таблице приведены возможные варианты использования команды **SELECT**, для выборки данных из приведённой для примера базы данных.

Рассмотрим более подробно использование оператора **WHERE**, который задаёт условие выборки данных. После оператора **WHERE** (в команде **SELECT**) идёт выражение, задающее условие выборки строк из указанных таблиц базы.

В выражении могут использоваться:

- операции отношения: = <> < > <= >=
- логические операции: OR, AND, NOT
- конструкции IS [NOT] NULL
- специальные функции:

ALL	Применяется совместно с операторами сравнения при сравнении со списком значений
ANY	Применяется совместно с операторами сравнения при сравнении со списком значений
BETWEEN	Применяется при проверке нахождения значения внутри заданного интервала (включая его границы)
IN	Применяется для проверки наличия значения в списке
LIKE	Применяется при проверке соответствия значения

- Может применяться вложенная выборка (**SELECT**);

Возможные варианты применения оператора **WHERE**, на примере той же базы данных приведены в следующей таблице:

SELECT ProductName, UnitPrice FROM Products WHERE Discontinued IS [NOT] NULL	Запрос полей ProductName, UnitPrice, для которых атрибут Discontinued пуст [не пуст]
SELECT CompanyName, ContactName FROM Customers WHERE CompanyName LIKE 'M%' OR Region='CA'	Сравнение по маске: '%' - любая последовательность символов, '_' - один любой символ
SELECT CompanyName, ContactName FROM Customers WHERE CompanyName BETWEEN 'M' AND 'N'	Лежит в диапазоне, включая границы
SELECT CompanyName, ContactName FROM Customers WHERE CustomerID [NOT] IN ('ALFKI', 'BERGS', 'VINET')	Определение принадлежности [не принадлежности]списку
SELECT * FROM PART WHERE PRICE > (SELECT PRICE FROM PART WHERE PNAME='Screw');	Запрос всех деталей, имеющих цену больше, чем у ‘Screw’
SELECT * FROM SUPPLIER S WHERE NOT EXISTS (SELECT * FROM SELLS SE WHERE SE.SNO = S.SNO);	Запрос поставщиков, которые ничего не продают

Теперь, когда Вы познакомились с синтаксисом основной команды языка SQL – SELECT, можно перейти к практическому применению этих знаний и в этом нам поможет написанная нами программа.

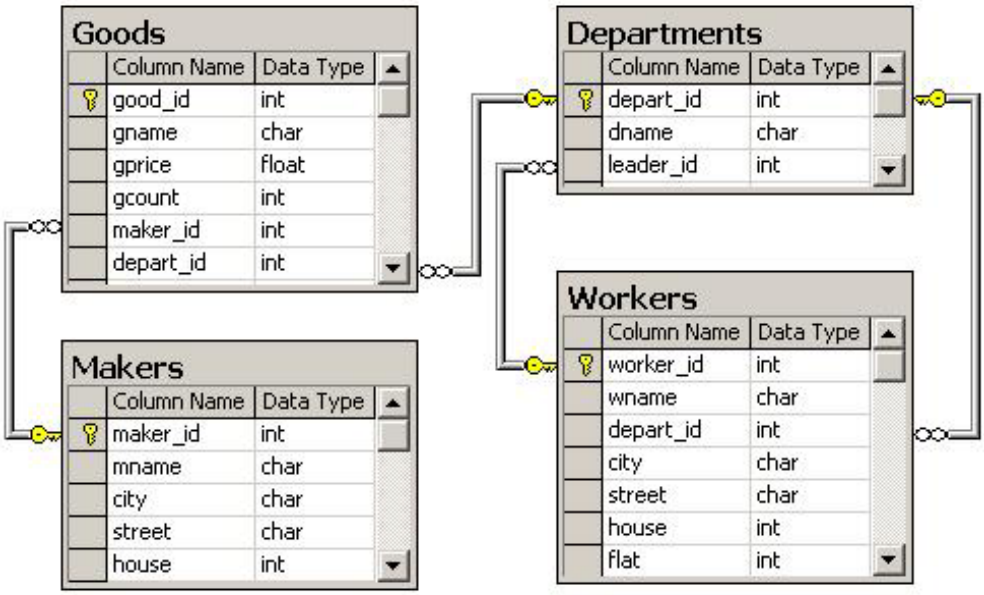
Шаг-6: Выполнение запросов к базе данных “Компьютерный магазин” с помощью написанной программы на ADO.NET

Для приобретения практического навыка использования команды SELECT, мы выполним несколько SQL-запросов, с помощью написанной Вами программы, к уже созданной и размещённой на сервере реальной базе данных “Компьютерный магазин”.

База данных “Компьютерного магазина” состоит из 4-х таблиц:

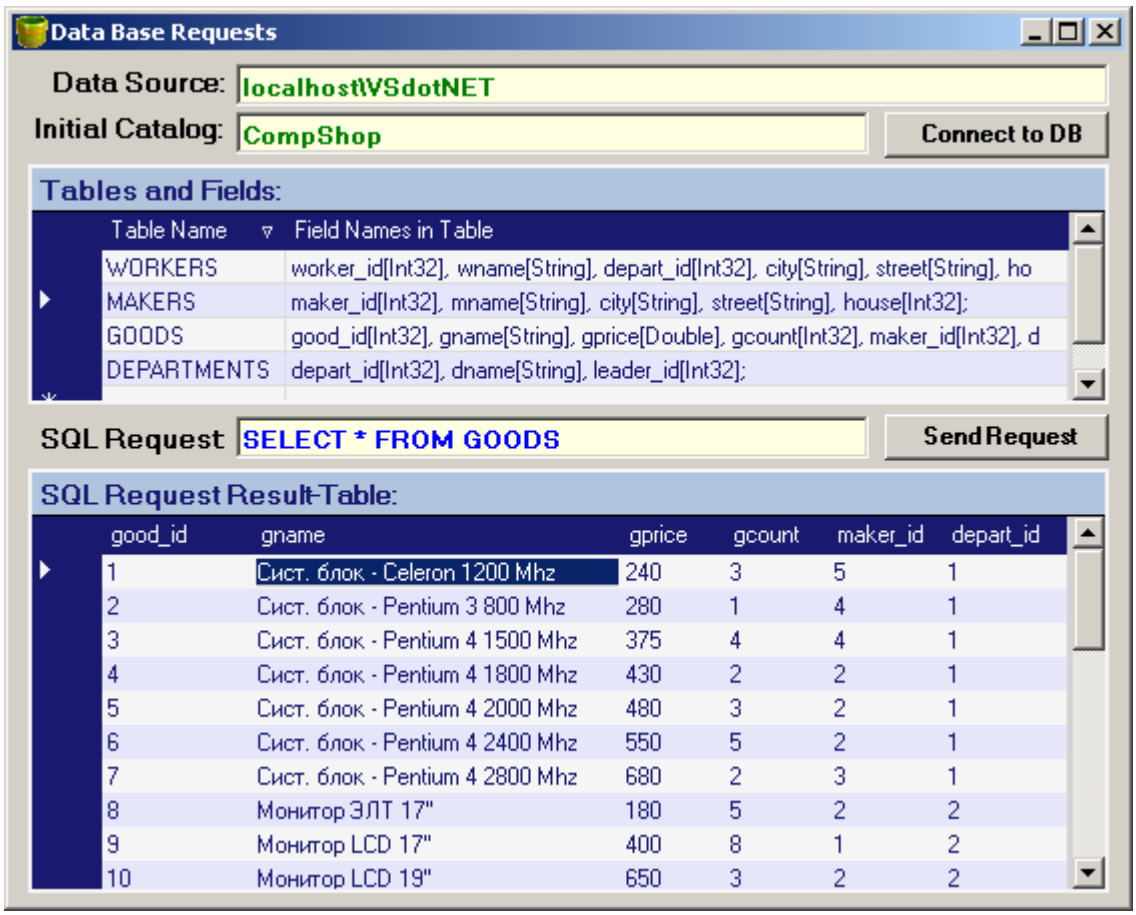
- ***Таблица*** **MAKERS** – производители компьютерной техники, у которых магазин делает закупки товаров для продажи. Производитель характеризуется свои названием (**mname**) и адресом: город (**city**), улица (**street**), дом (**house**), а так же уникальным идентификатором (**maker_id**) в таблице;
- ***Таблица GOODS*** – таблица имеющихся в наличии у магазина товаров. Товар определяется названием названием (**gname**), ценой (**gprice**), количеством единиц на складе (**gcount**), идентификаторами производителя этого товара (**meker_id**) и отдела в котором он продаётся (**depart_id**). Так же товар имеет уникальный идентификатор (**good_id**) в таблице;
- ***Таблица*** **DEPARTMENTS** – отделы магазина, продающие различные товары. Каждый отдел имеет название (**dname**), уникальный идентификатор (**depart_id**), а так же хранит идентификатор главного продавца отдела (**leader_id**);
- ***Таблица*** **WORKERS** – продавцы, работающие в магазине. Каждый продавец характеризуется своим именем (**wname**), номером отдела, в котором он работает (**depart_id**) и адресом проживания: город (**city**), улица (**street**), дом (**house**), квартира (**flat**);

На приведённой ниже диаграмме отражена структура таблиц базы данных и связей между ними. В таблицах указаны поля и их типы, помечены ключевые поля (идентифицирующие строку в таблице). Между таблицами отображены связи “один ко многим”:



После получения инструкций от помощников, проводящих лабораторную работу, подключите, написанную Вами программу к базе данных (**CompShop**) на сервере SQL Server 2000 и введите названия всех таблиц в ячейки DataGrid, отображающего поля таблиц БД.

Щёлкая мышью по строкам DataGrid, с названиями каждой из таблиц просмотрите данные, содержащиеся в таблицах БД. После этого самостоятельно **придумайте 3 демонстрационных SQL-запроса к базе данных** с использованием команды **SELECT** и выполните их с помощью Вашей программы. **Продемонстрируйте работу вашего приложения!**


Data Base Requests
Data Source:
localhost\VSdotNET
Initial Catalog:
CompShop
Connect to DB
Tables and Fields:
Table Name
Field Names in Table
WORKERS
worker_id[Int32], wname[String], depart_id[Int32], city[String], street[String], ho
MAKERS
maker_id[Int32], mname[String], city[String], street[String], house[Int32];
GOODS
good_id[Int32], gname[String], gprice[Double], gcount[Int32], maker_id[Int32], d
DEPARTMENTS
depart_id[Int32], dname[String], leader_id[Int32];
SQL Request
SELECT * FROM GOODS
Send Request
SQL Request Result-Table:
good_id gname gprice gcount maker_id depart_id
1 Сист. блок - Celeron 1200 Mhz 240 3 5 1
2 Сист. блок - Pentium 3 800 Mhz 280 1 4 1
3 Сист. блок - Pentium 4 1500 Mhz 375 4 4 1
4 Сист. блок - Pentium 4 1800 Mhz 430 2 2 1
5 Сист. блок - Pentium 4 2000 Mhz 480 3 2 1
6 Сист. блок - Pentium 4 2400 Mhz 550 5 2 1
7 Сист. блок - Pentium 4 2800 Mhz 680 2 3 1
8 Монитор ЭЛТ 17" 180 5 2 2
9 Монитор LCD 17" 400 8 1 2
10 Монитор LCD 19" 650 3 2 2