

**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ, МОЛОДІ І СПОРТУ УКРАЇНИ  
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ**

**Г.Г. ШВАЧИЧ, О.В. ОВСЯННИКОВ, Л.М. ПЕТРЕЧУК**

**КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ В ДІЛОВОДСТВІ**

**Розділ 1. ADO та MIDAS технології**

Затверджено на засіданні Вченої ради академії  
як навчальний посібник

**Дніпропетровськ НМетАУ 2012**

УДК 004 (075.8)

Швачич Г.Г., Овсянніков О.В., Петречук Л.М. Комп'ютерні технології в діловодстві. Розділ 1. ADO та MIDAS технології: Навч. посібник (російською мовою). – Дніпропетровськ: НМетАУ, 2012. – 62 с.

Викладена реалізація концепції ADO та MIDAS технологій та її застосування при створенні додатків у середовищі Delphi.

Призначений для студентів напрямку 6020105 – документознавство та інформаційна діяльність.

Іл. 44. Табл. 1. Бібліогр.: 10 найм.

Друкується за авторською редакцією.

Відповідальний за випуск                      Г.Г. Швачич, канд. техн. наук, проф.

Рецензенти:                      О.Г. Холод, канд. техн. наук, проф. (Дніпропетровський університет А. Нобеля)  
   І.Г. Бутенко, канд. техн. наук, доц. (УДХТУ)

© Національна металургійна академія  
України, 2012  
© Швачич Г.Г., Овсянніков О.В.,  
Петречук Л.М., 2012

## 1. ОСНОВЫ ТЕХНОЛОГИИ ADO

Технология **ADO** (*Microsoft ActiveX Data Objects*) обеспечивает универсальный доступ к источникам данных из приложений баз данных (БД). Такую возможность предоставляют функции набора интерфейсов, созданные на основе общей модели объектов COM (*Component Object Model*) и описанные в спецификации OLE DB (*Object Linking and Embedding, Database*).

Технология ADO и интерфейсы OLE DB обеспечивают для приложений единый способ доступа к источникам данных различных типов (рис.1.1). Например, приложение, использующее ADO, может применять одинаково сложные операции и к данным, хранящимся на корпоративном сервере SQL, и к электронным таблицам, и локальным СУБД. Запрос SQL, направленный любому источнику данных через ADO, будет выполнен.

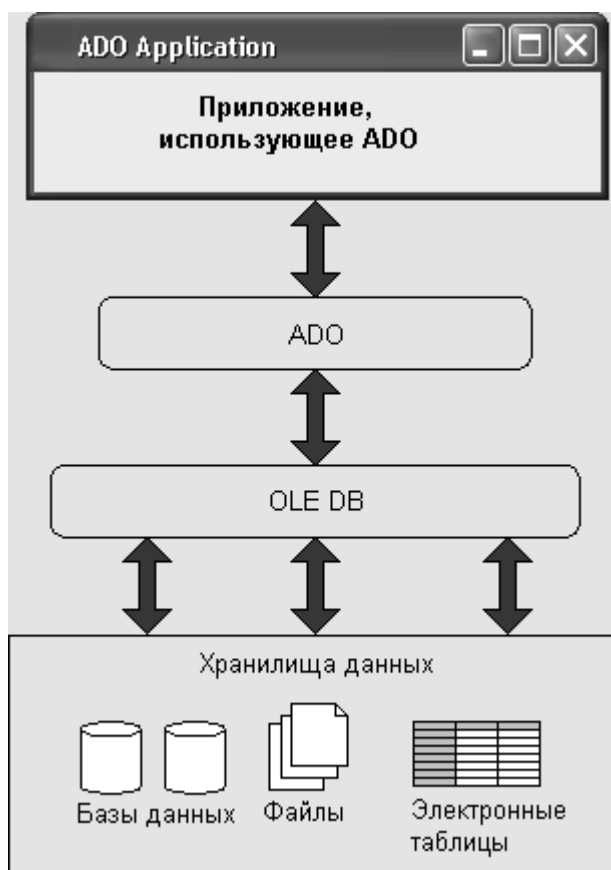


Рис.1.1. Схема доступа к данным через ADO

OLE DB представляет собой набор специализированных объектов COM, инкапсулирующих стандартные функции обработки данных, и специализиро-

ванные функции конкретных источников данных и интерфейсов, обеспечивающих передачу данных между объектами.

Инкапсуляция - это механизм, объединяющий данные и обрабатывающий их код как единое целое. Инкапсуляцией называется включение различных мелких элементов в более крупный объект, в результате чего программист работает непосредственно с этим объектом. Это приводит к упрощению программы, поскольку из нее исключаются второстепенные детали.

Согласно терминологии ADO, любой источник данных (база данных, электронная таблица, файл) называется хранилищем данных, с которым при помощи провайдера данных взаимодействует приложение. Минимальный набор компонентов приложения может включать объект соединения, объект набора данных, объект процессора запросов.

Объекты OLE DB создаются и функционируют так же, как и другие объекты COM. Каждому объекту соответствует идентификатор класса CLSID, хранящийся в системном реестре. Для создания объекта используется метод CoCreateInstance и соответствующая фабрика класса. Объекту соответствует набор интерфейсов, к методам которых можно обращаться после создания объекта.

В результате приложение обращается не прямо к источнику данных, а к объекту OLE DB, который «умеет» представить данные (например, из файла электронной почты) в виде таблицы БД или результата выполнения запроса SQL.

Технология ADO в целом включает в себя не только сами объекты OLE DB, но и механизмы, обеспечивающие взаимодействие объектов с данными и приложениями. На этом уровне важнейшую роль играют провайдеры ADO, координирующие работу приложений с хранилищами данных различных типов.

Так как технология ADO основана на стандартных интерфейсах COM, которые являются системным механизмом Windows, это сокращает общий объем работающего программного кода и позволяет распространять приложения БД без вспомогательных программ и библиотек.

## **1.1. Провайдеры ADO**

Провайдеры ADO обеспечивают соединение приложения, использующего данные через ADO, с источником данных (сервером SQL, локальной СУБД,

файловой системой и т. д.). Для каждого типа хранилища данных должен существовать провайдер ADO.

Провайдер «знает» о местоположении хранилища данных и его содержании, умеет обращаться к данным с запросами и интерпретировать возвращаемую служебную информацию и результаты запросов с целью их передачи приложению.

Список установленных в данной операционной системе провайдеров доступен для выбора при установке соединения через компонент TADOConnection.

## 1.2. Компоненты ADO

Механизмы доступа к данным через ADO, многочисленные объекты и интерфейсы реализованы в библиотеке VCL Delphi в виде набора компонентов, расположенных на странице **ADO**. Все необходимые интерфейсы, обеспечивающие работу компонентов, объявлены и описаны в файлах OleDB.pas и ADODB.pas в папке \Delphi5.6.7\Source\Vcl.

Компонент TADOConnection вобрал возможности перечислителя, источника данных и сессии с возможностями обслуживания транзакций.

Текстовые команды ADO реализованы в компоненте TADOCommand.

Наборы рядов (нотация Microsoft) можно получить при помощи компонентов **TADOTable**, **TADOQuery**, **TAOostoredProc**. Каждый из них реализует способ доступа к конкретному типу представления данных в хранилище. Далее по тексту, применительно к компонентам Delphi, совокупность возвращаемых из хранилища данных строк будем называть набором записей, что соответствует документации Inprise (см. [www.borland.com](http://www.borland.com) или [www.borland.ru](http://www.borland.ru)).

Набор свойств и методов компонентов ADO обеспечивает реализацию всех необходимых приложению БД функций. Способы использования компонентов ADO немногим отличаются от стандартных компонентов VCL доступа к данным.

Однако при необходимости разработчик может использовать все возможности интерфейсов ADO, обращаясь к ним через соответствующие объекты ADO. Ссылки на объекты имеются в компонентах.

### 1.3. Механизм соединения с хранилищем данных ADO

Компоненты доступа к данным ADO могут использовать два варианта подключения к хранилищу данных. Это стандартный метод ADO и стандартный метод Delphi.

В первом случае компоненты используют свойство **ConnectionString** для прямого обращения к хранилищу данных. Во втором случае используется специальный компонент **TADOConnection**, который обеспечивает расширенное управление соединением и позволяет обращаться к данным нескольким компонентам одновременно.

Свойство **ConnectionString** предназначено для хранения информации о соединении с объектом ADO. В нем через точку с запятой перечисляются все необходимые параметры. Как минимум, это должны быть имена провайдера соединения или удаленного сервера: `ConnectionString := 'Remote Server = ServerName; Provider=ProviderName'`.

При необходимости указываются путь к удаленному провайдеру: `ConnectionString:='Remote Provider=ProviderName'` и параметры, необходимые провайдеру: `'User Name=User_Name;Password=Password'`.

Каждый компонент, обращающийся к хранилищу данных ADO самостоятельно, задавая параметры соединения в свойстве **ConnectionString**, открывает собственное соединение. Чем больше приложение содержит компонентов ADO, тем больше соединений может быть открыто одновременно. Поэтому целесообразно реализовать механизм соединения ADO через специальный компонент - **TADOConnection**. Этот компонент открывает соединение, также заданное свойством **ConnectionString**, и предоставляет разработчику дополнительные средства управления соединением. Компоненты, работающие с хранилищем данных ADO через данное соединение, подключаются к компоненту **TADOConnection** при помощи свойства *property* **Connection: TADOConnection**, которое имеет каждый компонент, инкапсулирующий набор данных ADO.

### 1.4. Компонент TADOConnection

Компонент **TADOConnection** предназначен для управления соединением с объектами хранилища данных ADO. Он обеспечивает доступ к хранилищу данных компонентам ADO, инкапсулирующим набор данных.

Применение этого компонента дает разработчику ряд преимуществ:

- все компоненты доступа к данным ADO обращаются к хранилищу данных через одно соединение;
- возможность напрямую задать объект провайдера соединения;
- доступ к объекту соединения ADO;
- возможность выполнять команды ADO;
- выполнение транзакций;
- расширенное управление соединением при помощи методов-обработчиков событий.

### 1.5. Наборы данных ADO

На странице *ADO палитры компонентов Delphi*, кроме компонентов соединения есть стандартные компоненты, инкапсулирующие набор данных и адаптированные для работы с хранилищем данных ADO (рис.1.2). Это компоненты:

- TADODataSet — универсальный набор данных;
- TADOTable — таблица БД;
- TADOQuery — запрос SQL;
- TADOStoredProc — хранимая процедура.

Как и положено, для компонентов, инкапсулирующих набор данных, их общим предком является класс TDataSet, предоставляющий базовые функции управления набором данных.

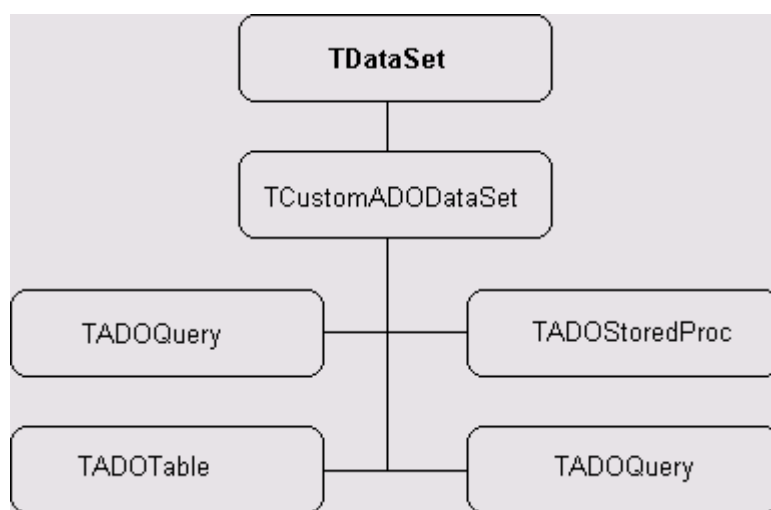


Рис.1.2. Иерархия классов наборов данных ADO

Компоненты ADO обладают обычным набором свойств и методов, а необходимый для доступа к данным через ADO механизм наследуют от своего общего предка - класса TCustomADODataset.

### 1.6. Компонент TADODataset

Компонент TADODataset предназначен для представления набора данных из хранилища данных ADO. Он прост в использовании, имея только несколько собственных свойств и методов, и применяет функции своего предка - класса TCustomADODataset.

Это единственный компонент ADO, инкапсулирующий набор данных, для которого опубликованы свойства, позволяющие управлять командой ADO. Это свойства: **CommandText**: *WideString* и **CommandType**: *TCommandType*.

В результате компонент представляет собой гибкий инструмент, который позволяет (в зависимости от типа команды и ее текста) получать данные из таблиц, запросов SQL, хранимых процедур, файлов и т. д. Например, вы выбираете нужное значение свойства CommandType = cmdText и заносите в свойство **CommandText** текст запроса SQL из редактора: ADODataSet.CommandType = cmdText; ADODataSet.CommandText := Memo1.Lines.Text. Запрос SQL готов к выполнению.

*Примечание.* Для запросов SQL можно применять только язык *Data Manipulation Language* (использовать только *SELECT*).

Соединение с базой данных задается свойством **ConnectionString** или **Connection**. Набор данных открывается и закрывается свойством **Active** или методами **Open** и **Close**.

В приложениях компонент можно применять как все обычные компоненты доступа к данным, связывая инкапсулированный в нем набор данных с визуальными компонентами отображения данных через компонент TDataSource.

### 1.7. Компонент TADOTable

Компонент TADOTable обеспечивает использование в приложениях Delphi таблиц БД, подключенных через провайдеры **OLE DB**. По своим функциональным возможностям и применению он подобен стандартному табличному компоненту.



Как вы уже знаете, в основе компонента лежит использование команды ADO, но ее свойства настроены заранее и изменению не подлежат.

Имя таблицы БД задается свойством **TableName**: *WideString*.

Другие свойства и методы компонента обеспечивают применение индексов (этой возможности лишен любой компонент запроса).

Так как не все провайдеры ADO обеспечивают прямое использование таблиц БД, то для доступа к ним может понадобиться запрос SQL. Если свойство **TableDirect**: *Boolean* имеет значение True, осуществляется прямой доступ к таблице. В противном случае компонент генерирует соответствующий запрос.

Свойство **ReadOnly**: *Boolean* позволяет включить или отключить для таблицы режим «только для чтения».

### 1.8. Компонент TADOQuery

Компонент TADOQuery обеспечивает применение запросов SQL при работе с данными через ADO. По своей функциональности он подобен стандартному компоненту запроса.

Текст запроса задается свойством *property* SQL: TStrings. Параметры запроса определяются свойством *property* Parameters: TParameters.

Если запрос должен возвращать набор данных, для его открытия используется свойство *property* Active: Boolean или метод *procedure* Open. В противном случае достаточно использовать метод *function* ExecSQL: Integer; ExecSQL .

Число обработанных запросом записей возвращает свойство *property* RowsAffected: Integer.

### 1.9. Компонент TADOStoredProc

Компонент TADOStoredProc позволяет использовать в приложениях Delphi, обращающихся к данным через ADO, хранимые процедуры. Он подобен стандартному компоненту хранимой процедуры. Имя хранимой процедуры определяется свойством *property* ProcedureName: WideString.

Для определения входных и выходных параметров используется свойство *property* Parameters: TParameters.

Если процедура будет применяться без изменений многократно, имеет смысл заранее подготовить ее выполнение на сервере. Для этого свойству *property* Prepared: Boolean присваивается значение True.

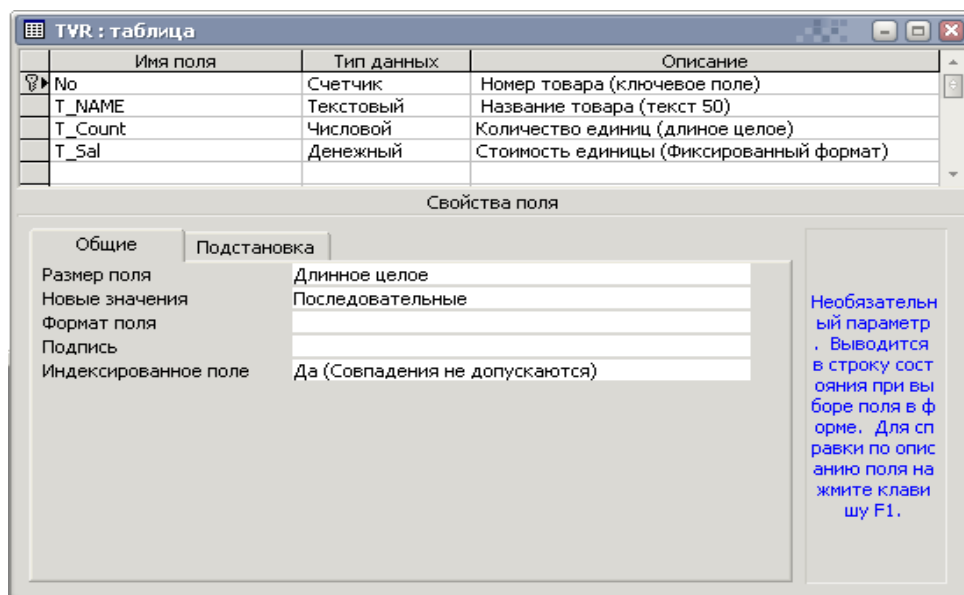
## 1.10. Пример приложения ADO клиент-сервер СУБД на основе ADOTable

Теперь попробуем применить на практике представленную в этой главе информацию о реализации ADO в Delphi. В качестве примера создадим приложение, которое «умеет» отображать пару таблиц БД, сохранять изменения при помощи групповых операций, сортировать записи и устанавливать фильтры на выбранные записи.

### Создание базы данных

Создайте в **ACCESS** базу данных с именем «**ТОВАР**» и сохраните ее в личной папке, например в папке с именем «**ADO\_TEST**».

Создайте таблицу базы данных и назовите ее «**TVR**», структура этой таблицы приведена на рисунке 1.3.



Имя поля	Тип данных	Описание
No	Счетчик	Номер товара (ключевое поле)
T_NAME	Текстовый	Название товара (текст 50)
T_Count	Числовой	Количество единиц (длинное целое)
T_Sal	Денежный	Стоимость единицы (Фиксированный формат)

Свойства поля

Общие | Подстановка

Размер поля: Длинное целое  
Новые значения: Последовательные  
Формат поля:  
Подпись:  
Индексированное поле: Да (Совпадения не допускаются)

Необязательный параметр. Выводится в строку описания при выборе поля в форме. Для справки по описанию поля нажмите клавишу F1.

Рис.1.3. Структура таблицы TVR

### Дизайн приложения

Создайте новый проект в **Delphi**. Установите в форму компоненты в соответствии с рисунком 1.4, где компонентом доступа к данным является **ADOTable**. Сохраните проект в личной папке.

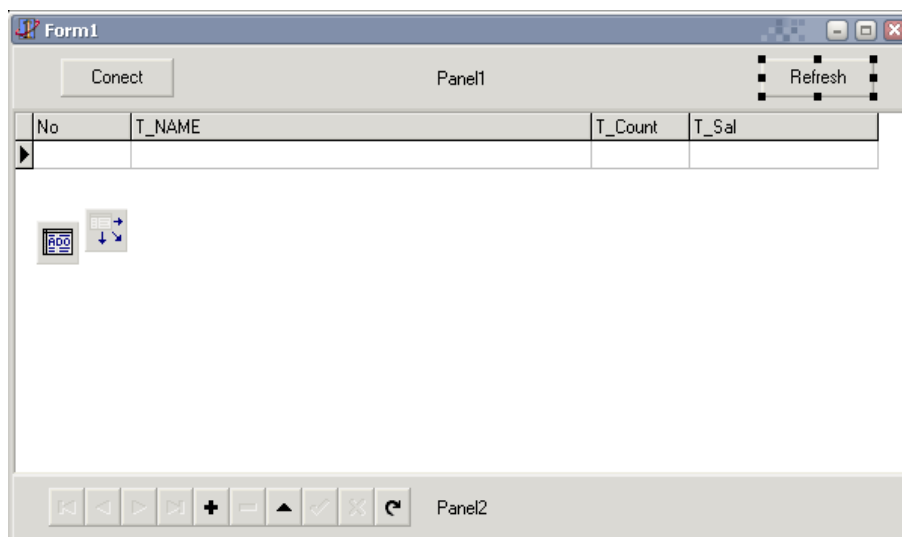


Рис.1.4. Вид приложения в период дизайна

Для установки связи с данными выберите компонент **ADOTabe** и в *инспекторе объектов* щелкните по кнопке [...] свойства **ConnectionString**. В открывшемся диалоговом окне (рис.1.5) нажмите кнопку **Build** (опция **User Connection Sring**), которая вызовет окно «Свойство связи с данными» (рис.1.6). В указанном окне выберите **Microsoft.Jet.OLEDB.4.0 Provider** и выполните команду «Далее».

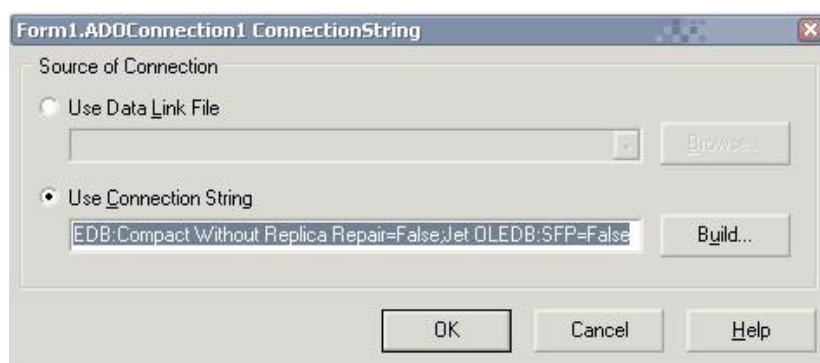


Рис. 1.5. Выбор соединения

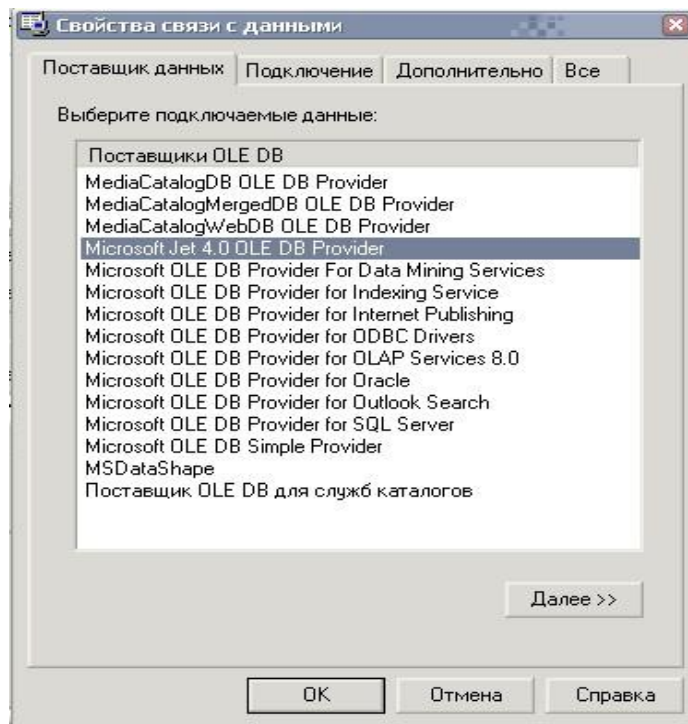


Рис.1.6. Свойства связи с данными

На закладке «Подключение» (рис.1.7) выберите созданную базу данных, посредством стандартного диалога, который вызывается щелчком по кнопке [...].

После установления связи выполните команду «Проверить подключение» и в случае положительного сообщения закройте диалоговые окна.

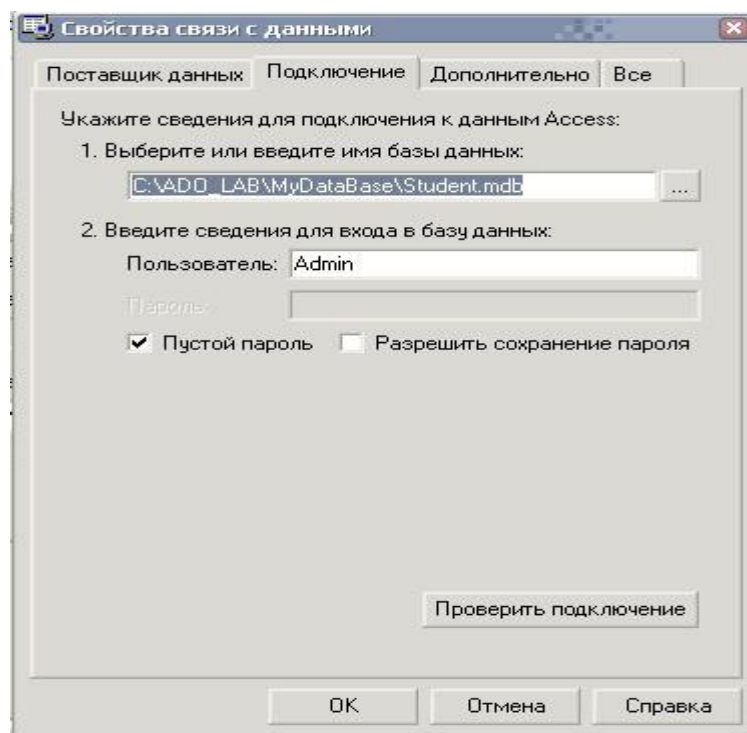


Рис.1.7. Соединение с базой данных

Для свойства **TableName** компонента **ADOTable** выберите значение (имя таблицы) **TVR**.

Выполните все необходимые соединения для компонентов управления данными и откройте таблицу, установив свойство **Active := True**.

Внешний вид приложения в процессе его завершения дизайна должен соответствовать виду, приведенному на рисунке 1.8.

### Запуск приложения СУБД

Запустите приложение и введите несколько записей в таблицу (рис.1.8). В процессе ввода данных поле **No** заполняется автоматически после выполнения команды навигатора **POST**.

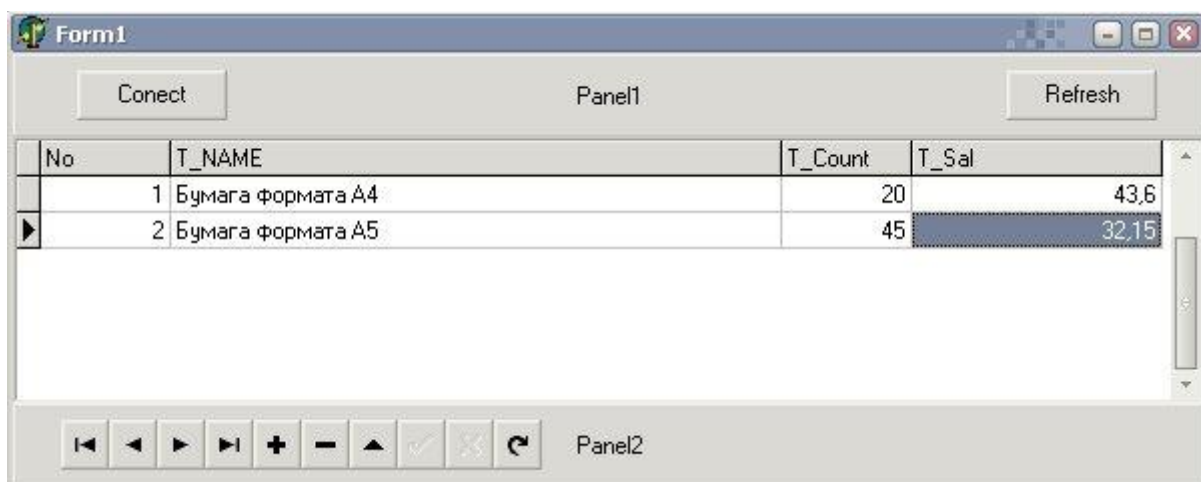


Рис.1.8. Вид приложения

### Команда обновления (отражения модификации) данных

Для обновления данных, измененных другими пользователями, в обработчике события команды (кнопки) **Refresh** запишите повторное открытие таблицы.

```
procedure TForm1.Button2Click(Sender: TObject);  
begin  
    ADOTable1.Close; ADOTable1.Open;  
end;
```

Для автоматического обновления данных, после ввода новых записей, аналогичный метод определите в обработчике события **AfterPost** компонента **ADOTable**.

```
procedure TForm1.ADOTable1AfterPost(DataSet: TDataSet);  
begin  
    ADOTable1.Close;  ADOTable1.Open;  
end;
```

Скомпилируйте приложение.

### Совместная работа приложений

Закройте среду **Delphi** и запустите из личной папки два экземпляра приложения (рис.1.9).

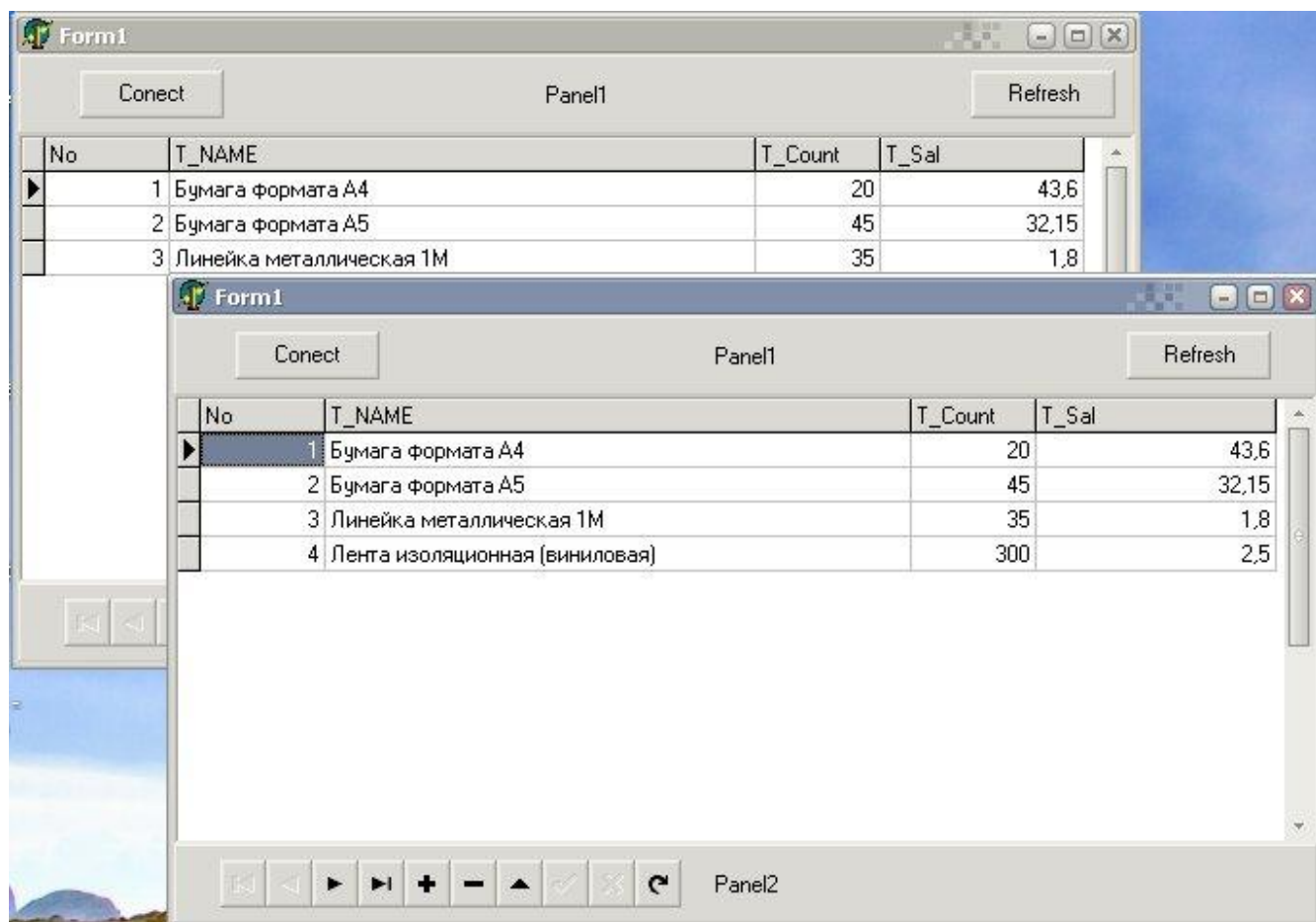


Рис.1.9. Совместная работа 2-х приложений

Поочередно введите новые записи в каждый экземпляр приложения.

Вы увидите, что модифицированные данные одного приложения будут отражаться в другом приложении после выполнении команды **Post** или **Refresh**.

### Подключение Link файла

Для обеспечения диалогового доступа к базе данных, используя **udl**-файл, выполните следующие действия.

Скопируйте в папку проекта файл **DBDEMOS.UDL**, находящийся по следующему пути: **Program Files /Common Files / System / Ole DB /Data Link /DBDEMOS.udl**.

Измените имя указанного файла на **LocalLink.udl**. Запустите файл **LocalLink.udl**, выполнив двойной щелчок мышью. Установите соединение с Вашей базой данных.

### Диалоговое соединение с базой данных

Откройте разрабатываемый проект в среде **Delphi**.

Закройте доступ к данным компонента **ADOTable** посредством свойства **Active := False**. Повторно установите связь с данными компонента **ADOTabe**, щелкнув по кнопке [...] свойства **ConectionString** и в открывшемся диалоговом окне (рис.1.10) выполните команду **Browse** (опция **User Data Link File**). В окне проводника выберите файл **LocalLink.udl**, находящийся в папке проекта.

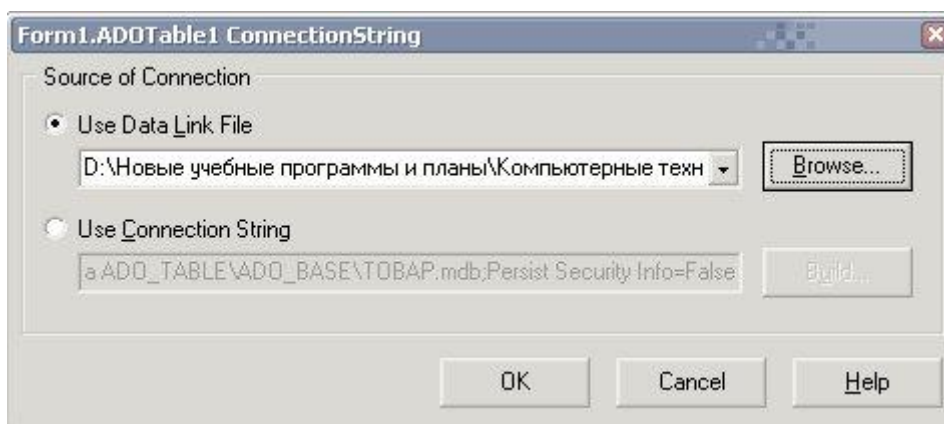


Рис.1.10. Выбор соединения

В раздел **USES** интерфейсной части **Unit'a** допишите **shellapi**.

**uses**

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, DBCtrls, ExtCtrls, Grids, DBGrids, Db, ADODB, **shellapi**;

В разделе **Private** объявите переменную **LocalPath : String**.

**private**

*{ Private declarations }*

LocalPath: **string**;

Значение переменной определите в обработчике события формы **OnCreate**.

```
procedure TForm1.FormCreate(Sender: TObject);
```

**begin**

```
LocalPath := ExtractFilePath(application.ExeName);
```

```
ADOTable1.ConnectionString := 'FILE NAME =' + LocalPath + 'LocalLink.udl';
```

```
end;
```

В обработчике события **OnClick** кнопки **Conect** вызовите ShellApi - функцию **ShellExecute**:

```
procedure TForm1.Button1Click(Sender: TObject);
```

**begin**

```
ADOTable1.Close;
```

```
ShellExecute(Form1.Handle, 'open', PCHAR(LocalPath + 'LocalLink.udl'), nil, ' ', SW_SHOW);
```

```
end;
```

Установите в форму дополнительную кнопку «Open», разместив ее справа от кнопки Conect. В обработчике события OnClick, установленной кнопки, напишите соответствующий метод:



```
procedure TForm1.Button3Click(Sender: TObject);  
begin  
    ADOTable1.Open;  
end;
```

Скомпилируйте приложение и проверьте его работу.

## Функционирование КЛИЕНТ-СЕРВЕР

Закройте среду **Delphi**. Запустите приложение из рабочей папки.

Выполните соединение (несколько пользователей, компьютеров) с одной выбранной базой данных, расположенной на выбранном Вами компьютере.

Проверьте функциональность системы в многопользовательском режиме, используя:

1. Отдельные экземпляры приложений (**exe** и **udl** файлы);
2. Вынесенные ярлыки exe-файла на компьютеры-клиенты;

## Применение закладок

В процессе совместной работы нескольких клиентов происходит модификация базы данных различными пользователями, вследствие чего у пользователей может потеряться фокус курсора (указатель на запись) после выполнении команды **Refresh**. Для исключения указанной ситуации используются *закладки*, позволяющие сохранять указатель курсора на модифицированной записи.

Для включения закладок модифицируем методы **Refresh**, которые описаны в обработчиках событий **AfterPost** и **OnClick** кнопки **Refresh**.

Учитывая тот факт, что программный код для обоих обработчиков событий будет одинаков, целесообразно описать собственный метод **MyRefresh**, к которому будем обращаться в соответствующих обработчиках событий.

Декларирование метода выполните в разделе **Private** unit-а.

```
private  
    { Private declarations }  
    LocalPath: string;  
procedure MyRefresh; // декларирование метода MyRefresh
```

В любом месте unit'a создайте конструкцию нового метода.

**Примечание.** Конструкция вновь создаваемых методов пишется вручную.

Опишите собственный метод *MyRefresh*, позволяющий сохранять указатель курсора на модифицированной записи.

```
{=====Собственный метод MyRefresh=====}  
procedure TForm1.MyRefresh;  
var  
BM : TBookmark; // объявление локальной переменной класса TBookmark  
begin  
    BM := ADOTable1.GetBookmark; // получить позицию закладки  
    ADOTable1.Close;  
    ADOTable1.Open;  
    ADOTable1.GotoBookmark(BM); // перейти в позицию закладки  
    ADOTable1.FreeBookmark(BM); // очистить закладку  
end;
```

Обращение к методу выполните в обработчиках событий **AfterPost** и **OnClick** кнопки **Refresh**, предварительно удалив предыдущий программный код.

```
procedure TForm1.ADOTable1AfterPost(DataSet: TDataSet);  
begin  
    MyRefresh; // вызов метода MyRefresh  
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);  
begin  
    MyRefresh; // вызов метода MyRefresh  
end;
```

Скомпилируйте приложение.

Запустите на одном компьютере два экземпляра приложения и проверьте функционирование метода, путем модификации различных записей и добавления новых записей в разных экземплярах приложений.

## Вычисляемые поля

В процессе разработки реальных СУБД возникает необходимость выполнения каких либо вычислений в строке. Наиболее удобно выполнять вычисления в виртуальных полях.

Предположим, что нам необходимо вычислить суммарную стоимость каждой позиции товара. С целью упрощения доступа к значениям полей Вашей таблицы, т.е. повышения читаемости программного кода, создайте доменную структуру таблицы, (рис.1.11) выполнив команду Add All Fields контекстного меню редактора полей компонента ADOTable.

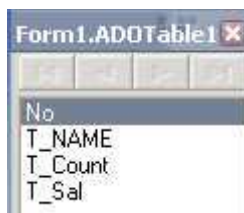


Рис.1.11. Вид доменной структуры таблицы

Добавьте новое вычисляемое (**Calculated**) поле **T\_SUM** типа **Currency** (рис.1.12), выполнив команду **New Field** контекстного меню редактора полей.

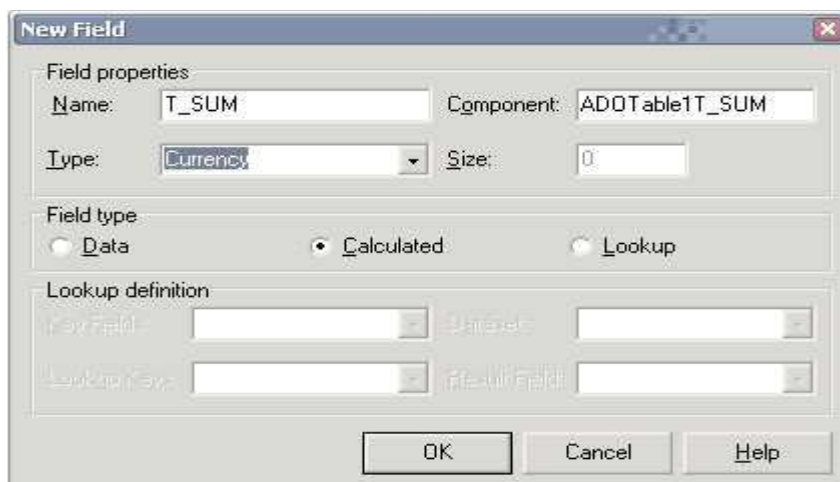


Рис.1.12. Вид окна добавления полей

Вычисление суммарной стоимости позиции товара выполните в обработчике события **OnCalcFields** компонента **ADOTable**.

```

procedure TForm1.ADOTable1CalcFields(DataSet: TDataSet);
begin
  ADOTable1T_SUM.Value:=ADOTable1T_Count.Value* ADOTable1T_Sal.Value;
end;

```

Скомпилируйте приложение и сравните представление информации в полях (рис.1.13).

No	T_NAME	T_Count	T_Sal	T_SUM
1	Бумага формата А4	20	43,6	872,00
2	Бумага формата А6	45	32,15	1 446,75
3	Линейка металлическая	35	1,8	63,00
4	Лента изоляционная (виниловая)	300	2,5	750,00

Рис.1.13. Представление информации в вычисляемом поле

### Пользовательские агрегатные методы

Довольно часто пользователю необходимо знать определенные данные, выбранные из значений столбцов. Такие данные достаточно просто получить, выполнив в цикле перебор значений строк с проверкой необходимых условий.

Предположим, необходимо вычислить общую сумму товара по вычисляемому полю T\_SUM, а также максимальную и минимальную суммы с указанием номеров записей.

Для решения поставленной задачи добавьте в форму кнопку Info и три компонента Label (рис.1.14).

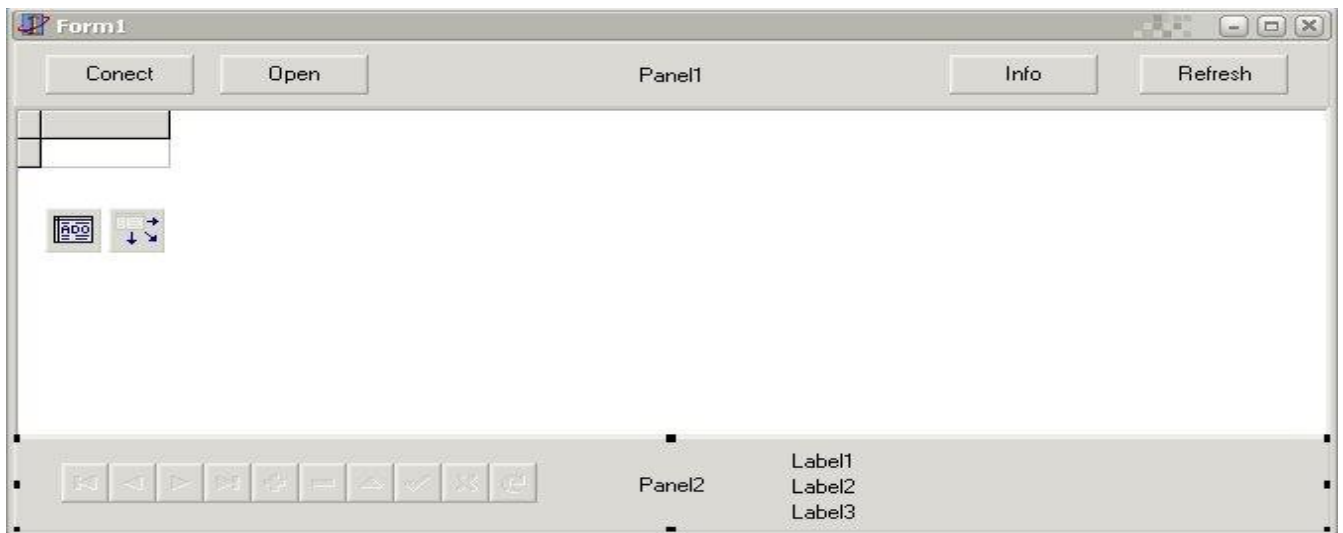


Рис.1.14. Добавление компонентов в форму

В обработчике события щелчка (Click) кнопки **Info** создайте конструкцию перебора всех строк таблицы, начиная с первой и завершая последней записью, с последующим возвратом к первой строке.

```

procedure TForm1.Button4Click(Sender: TObject);
var
i: integer;
begin
    ADOTable1.First;           // переход к первой записи
    for i := 1 to ADOTable1.RecordCount do // с первой записи до конца таблицы
    begin                       // выполняем в цикле
        {место написания будущих операций}

    ADOTable1.Next;           // переход к следующей записи
    end;                       // завершаем цикл
    ADOTable1.First;         // возврат к первой записи
end;

```

Скомпилируйте приложение с целью проверки правильности конструкции метода.

Далее объявите и определите начальные значения переменных:

**ALL\_SUM** –общая сумма значения по полю T\_SUM;

**MAX\_SUM, MIN\_SUM** – максимальное и минимальное значение в поле T\_SUM;

**RNMAX, RNMIN** – номера записей, соответствующие максимальному и минимальному значениям в поле T\_SUM.

```
procedure TForm1.Button4Click(Sender: TObject);  
var  
i, RNMAX, RNMIN: integer;  
ALL_SUM, MAX_SUM, MIN_SUM: Currency;  
begin  
  ADOTable1.First;  
  ALL_SUM := 0; // начальное значение переменной  
  MAX_SUM := ADOTable1T_SUM.Value; // начальное значение переменной  
  MIN_SUM := MAX_SUM; // начальное значение переменной  
  RNMAX := 1; RNMIN := 1; // начальные значения переменных  
  for i := 1 to ADOTable1.RecordCount do  
    begin  
      {место написания будущих операций}  
  
      ADOTable1.Next;  
    end;  
  ADOTable1.First;  
end;
```

Далее, в теле цикла, вычислите значение ALL\_SUM (общей суммы по полю T\_SUM), результат которой выведете после завершения цикла.

```
procedure TForm1.Button4Click(Sender: TObject);  
var  
i, RNMAX, RNMIN: integer;  
ALL_SUM, MAX_SUM, MIN_SUM: Currency;  
begin  
  ADOTable1.First;  
  ALL_SUM := 0;
```

```

MAX_SUM := ADOTable1T_SUM.Value;
MIN_SUM := MAX_SUM;
RNMAX := 1; RNMIN := 1;
for i := 1 to ADOTable1.RecordCount do
  begin
ALL_SUM := All_SUM + ADOTable1T_SUM.Value; // вычисление общей суммы
ADOTable1.Next;
  end;
  ADOTable1.First;
Label3.Caption := 'Общая сумма = ' + FloatToStr(ALL_SUM); // вывод результата
end;

```

Скомпилируйте приложение и проверьте правильность выполнения вычислений.

Вычислите максимальное значение поля и номер строки, соответствующий найденному значению и выведете соответствующую информацию.

```

procedure TForm1.Button4Click(Sender: TObject);
var
i, RNMAX, RNMIN: integer;
ALL_SUM, MAX_SUM, MIN_SUM: Currency;
begin
  ADOTable1.First;
  ALL_SUM := 0;
  MAX_SUM := ADOTable1T_SUM.Value;
  MIN_SUM := MAX_SUM;
  RNMAX := 1; RNMIN := 1;
  for i := 1 to ADOTable1.RecordCount do
    begin
      ALL_SUM := All_SUM + ADOTable1T_SUM.Value;

      if ADOTable1T_SUM.Value > MAX_SUM then // если, текущее знач. >
MAX_SUM, то:

    begin

```

```

MAX_SUM := ADOTable1T_SUM.Value; // MAX_SUM = текущему значению
RNMAX := ADOTable1.RecNo;      // и номер записи = текущему номеру
end;
ADOTable1.Next;
end;
ADOTable1.First;
Label1.Caption := 'Максимальная сумма = ' + FloatToStr(MAX_SUM) +
                 ' № записи ' + IntToStr(RNMAX);      // вывод результата
Label3.Caption := 'Общая сумма = ' + FloatToStr(ALL_SUM);
end;

```

Скомпилируйте приложение и проверьте правильность выполнения вычислений.

Аналогично вычислите минимальное значение поля и номер строки, соответствующий найденному значению и выведете соответствующую информацию.

```

procedure TForm1.Button4Click(Sender: TObject);
var
i, RNMAX, RNMIN: integer;
ALL_SUM, MAX_SUM, MIN_SUM: Currency;
begin
  ADOTable1.First;
  ALL_SUM := 0;
  MAX_SUM := ADOTable1T_SUM.Value;
  MIN_SUM := MAX_SUM;
  RNMAX := 1; RNMIN := 1;
  for i := 1 to ADOTable1.RecordCount do
  begin
    ALL_SUM := All_SUM + ADOTable1T_SUM.Value;

    if ADOTable1T_SUM.Value > MAX_SUM then
    begin

```



```

MAX_SUM := ADOTable1T_SUM.Value;
RNMAX := ADOTable1.RecNo;
end;
if ADOTable1T_SUM.Value < MIN_SUM then // если, текущее знач. <
MIN_SUM, то:
begin
MIN_SUM := ADOTable1T_SUM.Value; // MIN_SUM = текущему значению
RNMIN := ADOTable1.RecNo; // и номер записи = текущему номеру
end;
ADOTable1.Next;
end;
ADOTable1.First;
Label1.Caption := 'Максимальная сумма = ' + FloatToStr(MAX_SUM) +
' № записи ' + IntToStr(RNMAX);
Label2.Caption := 'Минимальная сумма = ' + FloatToStr(MIN_SUM) +
' № записи ' + IntToStr(RNMIN); // вывод результата
Label3.Caption := 'Общая сумма = ' + FloatToStr(ALL_SUM);
end;

```

Скомпилируйте приложение и проверьте правильность выполнения вычислений.

## 1.11. Пример приложения ADO с использованием компонента ADOConnection

### Создание базы данных

Создайте в **ACCESS** базу данных с именем «**MODULE**» и сохраните ее в личной папке с именем «**DATA**». Название создаваемых таблиц и их структура приведены на рисунках 1.15 - 1.19.

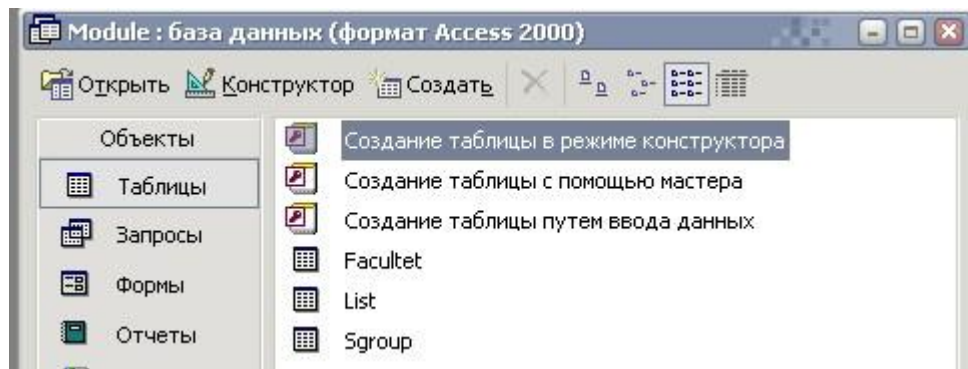


Рис.1.15. Перечень и названия таблиц

Имя поля	Тип данных	Описание
No	Счетчик	Ключевое поле
FName	Текстовый	Название факультета (25)

Свойства поля

Рис.1.16. Структура таблицы Facultet

Имя поля	Тип данных	Описание
No	Счетчик	Ключевое поле
Link	Числовой	Поле связи
StudGroup	Текстовый	Студенческая группа (12)

Свойства поля

Рис.1.17. Структура таблицы Sgroup

Имя поля	Тип данных	Описание
No	Счетчик	Ключевое поле
Link	Числовой	Поле связи
RegNo	Текстовый	Номер зачетной книжки (12)
FIO	Текстовый	Фамилия и инициалы (25)

Свойства поля

Рис.1.18. Структура таблицы List

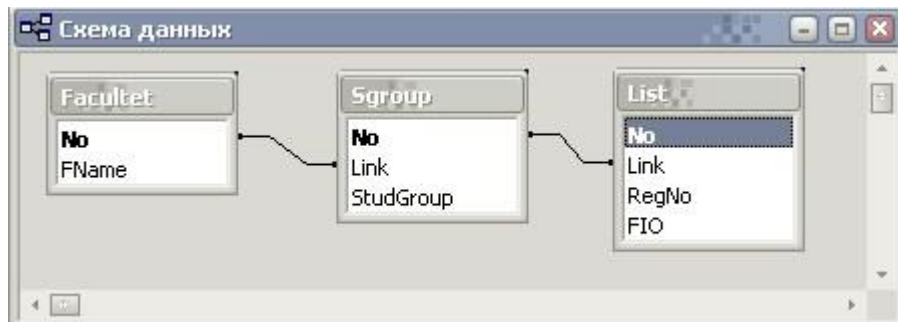


Рис.1.19. Схема данных

## Дизайн приложения

Создайте в Delphi новый проект и сохраните его личной папке.

Установите в форму, согласно рисунку 1.20, следующие компоненты:

- Panel1, в которой разместите кнопки Button1 - Button3 и поле ввода Memo1;
- Panel2, в которой разместите группирующие элементы GroupBox1 - GroupBox3;
- В каждый GroupBox соответственно поместите DBGrid1- DBGrid3 и DBNavigator1- DBNavigator3.

Установите компоненты доступа к данным ADOConnection1, ADOTable1- ADOTable3 и DataSource1- DataSource3.

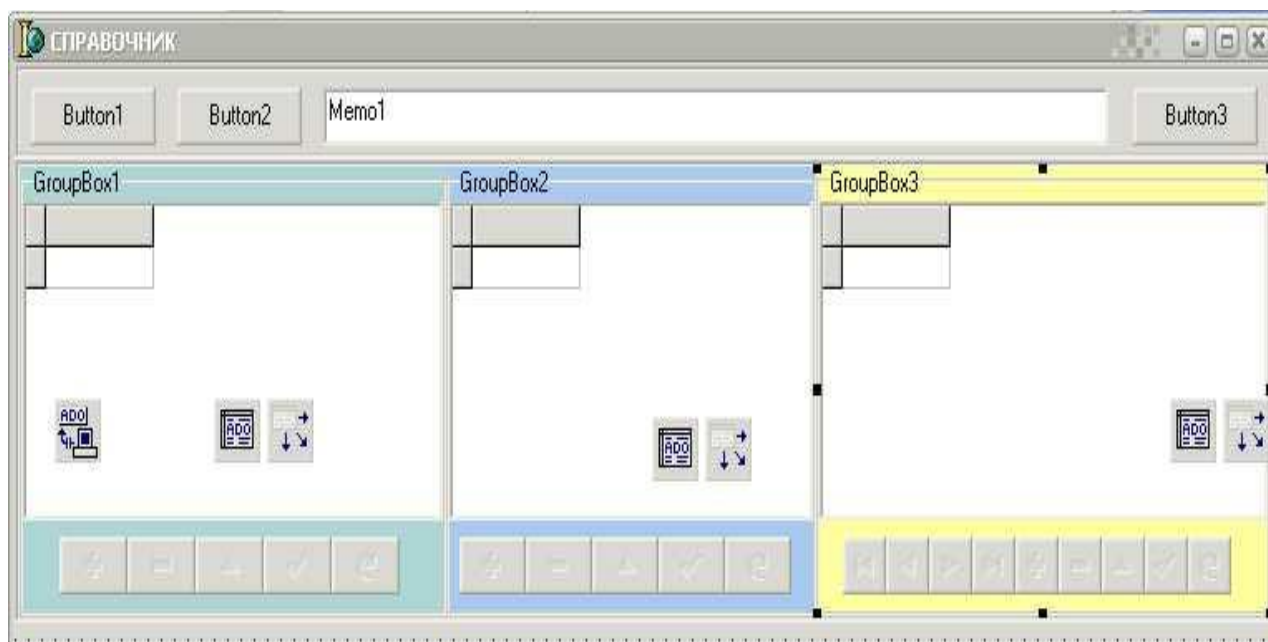


Рис.1.20. Дизайн приложения

Для отключения функции проверки пароля и пользователя установите свойство LoginPromt компонента ADOConnection1 в состояние False.

Посредством свойства ConectionString установите соединение с базой данных «MODULE», применив условие Use Connection String (рис.1.21) и провайдер Microsoft Jet 4.0 OLE DB Provider (рис.1.22).

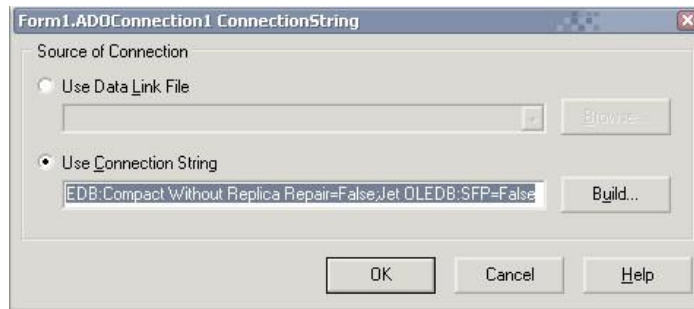


Рис.1.21. Условие соединения

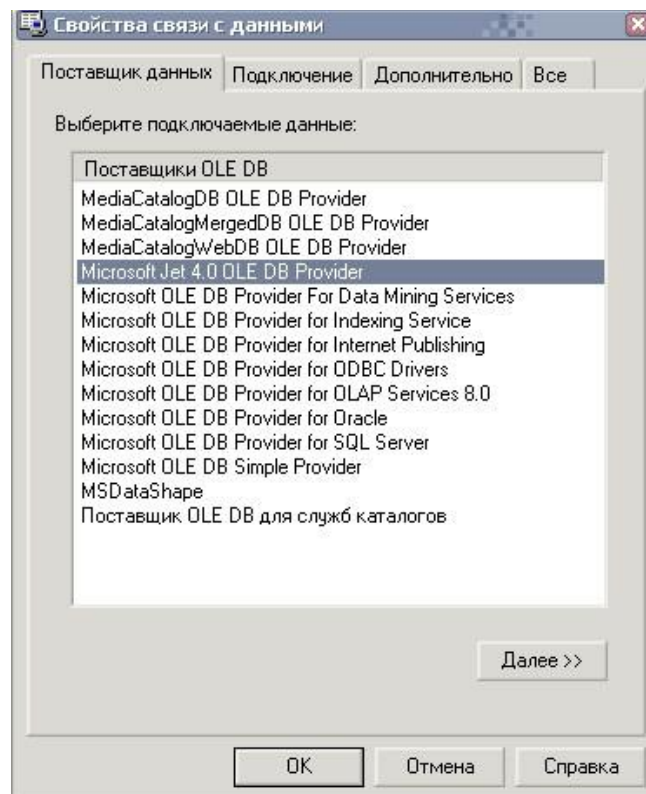


Рис.1.22. Выбор провайдера

Свяжите компоненты **ADOTable1**, **ADOTable2**, **ADOTable3** с компонентом **ADOConnection1** посредством свойства **Connection** каждой таблицы.

Подключите таблицы **ADOTable1**, **ADOTable2** и **ADOTable3** к соответствующим таблицам базы данных **Facultet**, **Sgroup** и **List** выбрав их значения в свойстве **TableName**.

Соедините элементы управления **DBGrid1-3**, **DBNavigator1-3** с соответствующими таблицами посредством компонентов связи **DataSource1-3**.

Реализуйте схему соединения данных (рис.1.19) используя свойства MasterSource и MasterFields соответствующих наборов данных ADOTable2 и ADOTable3.

Переведите наборы данных ADOTable1, ADOTable2, ADOTable3 в активное состояние и скомпилируйте приложение.

Заполните по несколько записей в базу данных, как показано на рисунке 1.23.

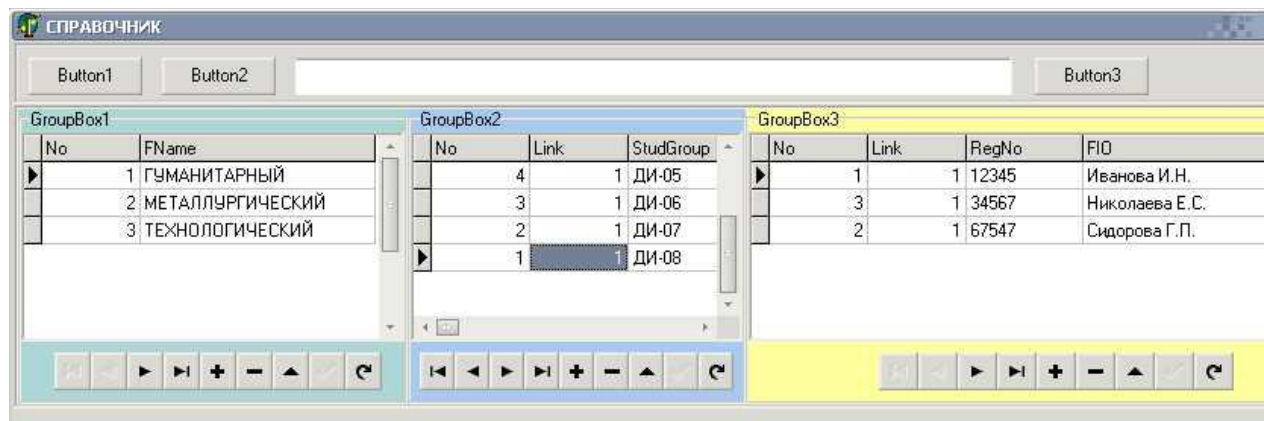


Рис.1.23. Записи в базе данных

### Программное управление приложением

В раздел **USES** интерфейсной части **Unit'a** добавьте модуль **AdoConEd**.  
Объявите переменную пути к папке приложения.

```

var
  Form1: TForm1;
  LinkPath : string;
implementation

```

В обработчике события формы OnCreate определите путь к папке приложения.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
  LinkPath := ExtractFilePath(application.ExeName);
end;

```

В обработчике события щелчка (Click) кнопки Button1 напишите программный код, реализующий вызов соединения с базой данных и сохранение описания соединения в собственном файле Link.txt.

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    AdoConnection1.Close;  
    AdoConnection1.ConnectionString := Memo1.Text;  
    if EditConnectionString(AdoConnection1) then  
        begin  
            Memo1.Text := AdoConnection1.ConnectionString;  
            Memo1.Lines.SaveToFile(LinkPath + 'Link.txt');  
        end;  
end;
```

Проверьте работу приложения. Просмотрите содержание файла Link.txt.

В обработчике события OnCreate формы реализуйте загрузку файла соединения Link.txt и сортировку записей в полях таблиц по алфавиту.

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    LinkPath := ExtractFilePath(application.ExeName);  
    AdoConnection1.Close;  
    Memo1.Lines.LoadFromFile(LinkPath + 'Link.txt');  
    ADOTable1.IndexFieldNames := 'FName';  
    ADOTable2.IndexFieldNames := 'Link; StudGroup';  
    ADOTable3.IndexFieldNames := 'Link; FIO';  
end;
```

В обработчике события щелчка кнопки Button2 напишите программный код, реализующий открытие таблиц базы данных с учетом загруженного или выбранного соединения.

```

procedure TForm1.Button2Click(Sender: TObject);
begin
  if AdoConnection1.Connected = false then
    begin
      AdoConnection1.ConnectionString := Memo1.Text;
      ADOTable1.Open; ADOTable2.Open; ADOTable3.Open;
    end;
end;

```

Проверьте работу приложения. Проверьте работу приложения в многопользовательском режиме, используя общую базу данных, размещенную на выбранном Вами компьютере.

Далее предполагается самостоятельное завершение дизайна приложения. Внешний вид доработанного приложения должен соответствовать виду, показанному на рисунке 1.24.

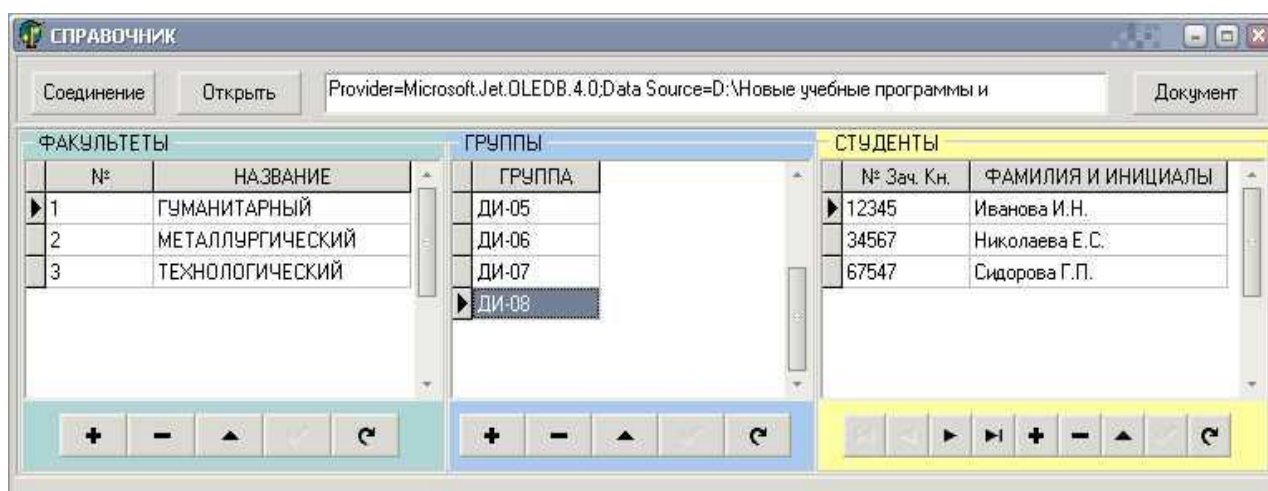


Рис.1.24. Вид приложения

Также рекомендуется разработать шаблон Master – Detail – Detail документа (отчета) позволяющего распечатать **подчиненно** (в древовидном представлении): **факультеты, группы и списки студентов.**

## 2. ТЕХНОЛОГИЯ MIDAS

Ранее мы рассматривали вопросы создания обычных приложений БД, работающих с базами данных на локальных компьютерах или в пределах локальной сети. Однако, как быть, если необходимо создать приложение, которое может с одинаковым успехом работать как в локальной сети, так и на удаленном компьютере.

Очевидно, что в этом случае модель доступа к данным должна быть расширена, т. к. наличие большого числа удаленных клиентов делает традиционные схемы создания приложений БД малоэффективными.

Рассмотрим модель распределенного приложения БД, которая называется многозвенной (multitiered), и, в частности, ее наиболее простой вариант - *трехзвенное распределенное приложение*. Три части такого приложения являются:

- собственно сервер базы данных;
- сервер приложений (серверная часть приложения);
- клиентская часть приложения.

Все они объединены в единое целое единым механизмом взаимодействия (транспортный уровень) и обработки данных (уровень бизнес-логики).

Компоненты и объекты Delphi, обеспечивающие разработку многозвенных приложений, объединены общим названием **DataSnap**.

**Примечание.** В предыдущих версиях Delphi (4 и 5) эти компоненты объединялись под названием *MIDAS (Multi-tier Distributed Applications Services— сервисы многозвенных распределенных приложений)*.

Палитра компонентов Delphi содержит специальную страницу **DataSnap**, на которой доступно большинство рассматриваемых компонентов. Однако при разработке многозвенных приложений нам понадобятся и многие другие компоненты.

Рассмотрим следующие вопросы:

- структура многозвенных приложений;
- механизм удаленного доступа к данным DataSnap;
- удаленные модули удаленных данных;
- компоненты-провайдеры;
- транспортные компоненты удаленных соединений DataSnap;
- вспомогательные компоненты - брокеры соединений.



## 2.1. Структура многозвенного приложения в Delphi

Многозвенная архитектура приложений баз данных определена необходимостью обрабатывать на стороне сервера запросы от большого числа удаленных клиентов. Казалось бы, с этой задачей вполне могут справиться и приложения клиент/сервер. Однако в этом случае при большом числе клиентов вся вычислительная нагрузка ложится на сервер БД, который обладает довольно скудным набором средств для реализации сложной бизнес-логики (хранимые процедуры, триггеры, просмотры и т. д.). И разработчики вынуждены существенно усложнять программный код клиентского программного обеспечения (ПО), а это крайне нежелательно при наличии большого числа удаленных клиентских компьютеров. Ведь с усложнением клиентского ПО возрастает вероятность ошибок и усложняется его обслуживание.

Многозвенная архитектура приложений БД призвана исправить перечисленные недостатки.

Итак, в рамках рассматриваемой архитектуры «тонкие» клиенты представляют собой простейшие приложения, обеспечивающие лишь передачу данных, их локальное кэширование, представление средствами пользовательского интерфейса, редактирование и простейшую обработку.

Клиентские приложения обращаются не к серверу БД напрямую, а к специализированному ПО промежуточного слоя. Это может быть и одно звено (простейшая трехзвенная модель) и более сложная структура.

Программное обеспечение промежуточного слоя называется сервером приложений, принимает запросы клиентов, обрабатывает их в соответствии с запрограммированными правилами бизнес-логики, при необходимости преобразует в форму, удобную для сервера БД и отправляет серверу.

Сервер БД выполняет полученные запросы и отправляет результаты серверу приложений, который адресует данные клиентам.

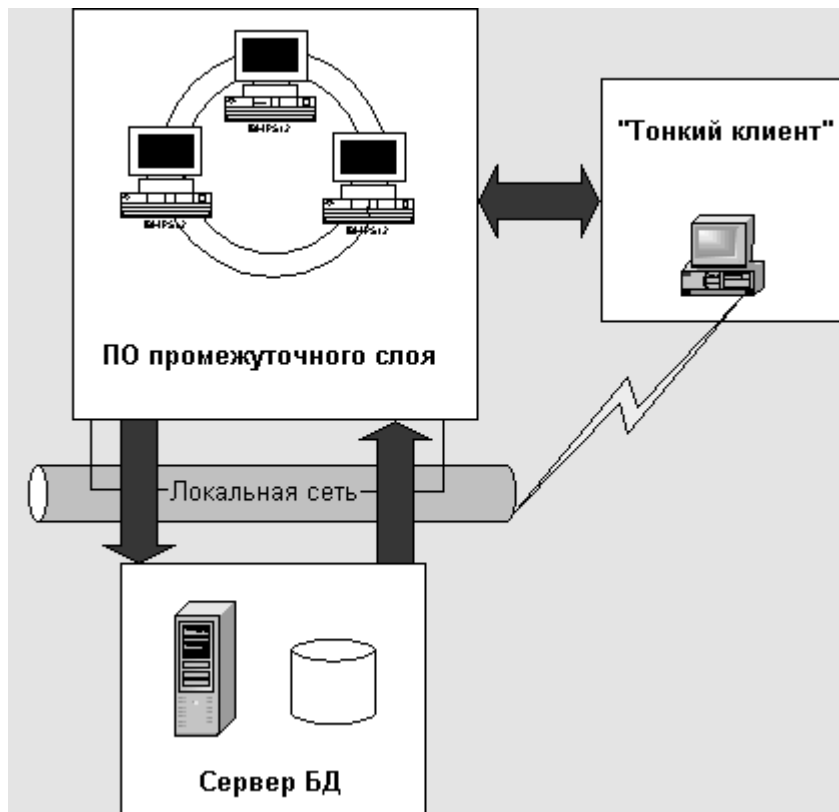


Рис.2.1. Многозвенная архитектура приложений БД

Таким образом, многозвенное приложение БД (рис.2.1) состоит из:

- «тонких» клиентских приложений, обеспечивающих лишь передачу, представление, редактирование и простейшую обработку данных;
- одного или нескольких звеньев ПО промежуточного слоя (сервер приложений), которые могут функционировать как на одном компьютере, так и распределенно - в локальной сети;
- сервера БД (Oracle, Sybase, MS SQL, InterBase и т.д.), поддерживающего функционирование базы данных и обрабатывающего запросы.

Более простая трехзвенная модель содержит следующие элементы:

- «тонкие» клиенты;
- сервер приложений;
- сервер БД.

Сервер приложений взаимодействует с сервером БД, используя одну из технологий доступа к данным, реализованным в Delphi. Это технологии ADO, BDE, InterBase Express и dbExpress. Разработчик может выбрать наиболее подходящую, исходя из поставленной задачи и параметров сервера БД.

Удаленные клиентские приложения создаются с использованием специального набора компонентов, объединенных общим названием DataSnap. Эти

компоненты инкапсулируют стандартные транспорты (DCOM, HTTP, сокет) и обеспечивают соединение удаленного клиентского приложения с сервером приложения. Также компоненты DataSnap обеспечивают доступ клиента к функциям сервера приложений за счет использования интерфейса AppServer.

Важную роль при разработке клиентских приложений играет компонент, инкапсулирующий клиентский набор данных. Его реализации также зависят от технологий доступа к данным и рассматриваются далее.

Наряду с перечисленными выше преимуществами, наличие дополнительного звена - сервера приложений - дает некоторые преимущества, которые могут быть весьма существенным подспорьем с точки зрения повышения надежности и эффективности системы.

Так как зачастую клиентские компьютеры - это достаточно слабые машины, реализация сложной бизнес-логики на стороне сервера позволяет существенно повысить быстродействие системы в целом, не только за счет более мощной техники, но и за счет оптимизации выполнения однородных запросов пользователей.

Например, при чрезмерной загрузке сервера, сервер приложений может самостоятельно обрабатывать запросы пользователей (ставить их в очередь или отменять) без дополнительной загрузки сервера БД.

Наличие сервера приложений повышает безопасность системы, т. к. на этом уровне можно организовать авторизацию пользователей, и любые другие функции безопасности без прямого доступа к данным.

Кроме того, вы легко сможете использовать защищенные каналы передачи данных, например HTTPS.

## **2.2. Трехзвенное приложение в Delphi**

Рассмотрим составные части трехзвенного распределенного приложения в Delphi (рис.2.2). Как рассматривалось ранее, в Delphi целесообразно разрабатывать клиентскую часть трехзвенного приложения и программное обеспечение промежуточного слоя - сервер приложений.

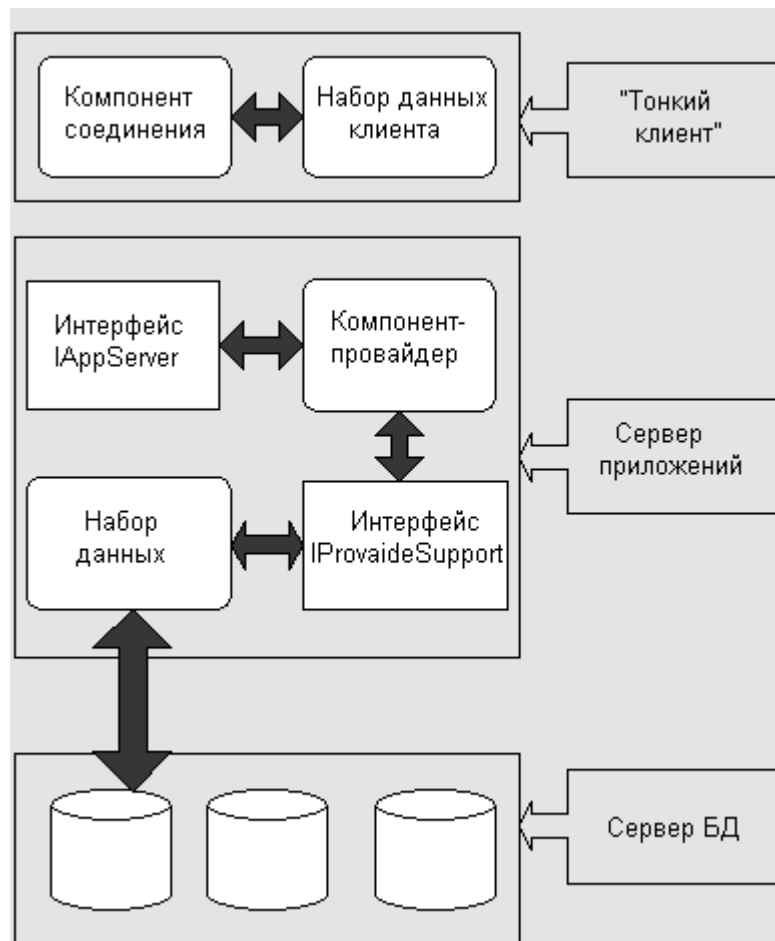


Рис.2.2. Схема трехзвенного распределенного приложения

Части трехзвенных приложений разрабатываются с использованием компонентов DataSnap, а также некоторых других специализированных компонентов, в основном обеспечивающих функционирование клиента. Для доступа к данным применяется одна из четырех технологий, реализованных в Delphi.

***Примечание.** Разработку трехзвенных приложений целесообразно вести, используя в среде разработки группу проектов вместо одиночных проектов. Для этого используется утилита Project Manager (меню View | Project Manager).*

Для передачи данных между сервером приложений и клиентами используется интерфейс **AppServer**, предоставляемый удаленным модулем данных сервера приложений. Этот интерфейс используют компоненты-провайдеры **TDataSetProvider** на стороне сервера и компоненты **TClientDataSet** на стороне клиента.

## 2.3. Сервер приложений

Сервер приложений инкапсулирует большую часть бизнес-логики распределенного приложения и обеспечивает доступ клиентов к базе данных. Основной частью сервера приложений является удаленный модуль данных.

Во-первых, подобно обычному модулю данных он является платформой для размещения невизуальных компонентов доступа к данным и компонентов-провайдеров. Размещенные на нем компоненты соединений, транзакций и компоненты, инкапсулирующие наборы данных, обеспечивают трехзвенное приложение связью с сервером БД. Это могут быть наборы компонентов для технологий ADO, BDE, InterBase Express, dbExpress.

Во-вторых, удаленный модуль данных реализует основные функции сервера приложений на основе предоставления клиентам интерфейса IAppServer или его потомка. Для этого удаленный модуль данных должен содержать необходимое число компонентов-провайдеров TDataSetProvider. Эти компоненты передают пакеты данных клиентскому приложению, а точнее компонентам TClientDataSet, а также обеспечивают доступ к методам интерфейса.

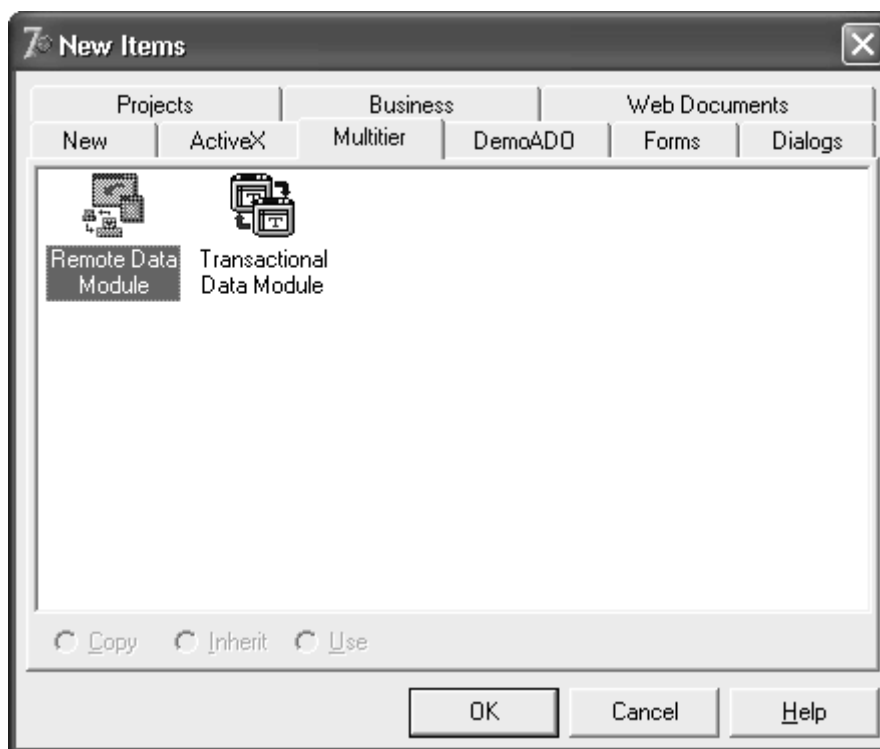


Рис.2.3. Выбор удаленных модулей данных в репозитории Delphi

В состав Delphi входят удаленные модули данных. Для их создания используйте страницы **Multitier**, **WebSnap** и **WebServices репозитория Delphi** (рис.2.3), это:

- Remote Data Module — удаленный модуль данных, инкапсулирующий сервер Автоматизации. Используется для организации соединений через DCOM, HTTP, сокет;
- Transactional Data Module — удаленный модуль данных, инкапсулирующий сервер MTS (Microsoft Transaction Server);
- SOAP Server Data Module — удаленный модуль данных, инкапсулирующий сервер SOAP (Simple Object Access Protocol);
- WebSnap Data Module — удаленный модуль данных, использующий Web-службы и Web-браузер в качестве сервера.

Помимо удаленного модуля данных неотъемлемой частью сервера приложений являются компоненты-провайдеры TDataSetProvider. С каждым компонентом, инкапсулирующим набор данных, предназначенным для передачи клиенту, в модуле данных должен быть связан компонент-провайдер.

Многозвенные распределенные приложения обеспечивают эффективный доступ удаленных клиентов к базе данных, так как в них для управления доступом к данным применяется специализированное ПО промежуточного слоя. В наиболее распространенной схеме — трехзвенном приложении — это сервер приложения, который выполняет следующие функции:

- обеспечивает авторизацию пользователей;
- принимает и передает запросы пользователей и пакеты данных;
- регулирует доступ клиентских запросов к серверу БД, балансируя нагрузку сервера БД;
- может содержать часть бизнес-логики распределенного приложения, обеспечивая существование «тонких» клиентов.

Delphi обеспечивает разработку серверов приложений на основе использования ряда технологий:

- Web;
- Автоматизация;
- MTS;
- SOAP.

Рассмотрим следующие вопросы:

- программные элементы сервера приложения Delphi;

- структура сервера приложения;
- типы удаленных модулей данных;
- создание и настройка удаленных модулей данных;
- роль компонентов-провайдеров в передаче данных клиентам;
- методы интерфейса IAppServer;
- регистрация сервера приложения.

## 2.4. Структура сервера приложения

Итак, сервер приложения — это программное обеспечение промежуточного слоя трехзвенного распределенного приложения (см. рис.2.2). Его основой является удаленный модуль данных. В Delphi предусмотрено использование удаленных модулей данных пяти типов.

Каждый удаленный модуль данных инкапсулирует интерфейс IAppServer, методы которого используются в механизме удаленного доступа клиентов к серверу БД.

Для обмена данными с сервером БД модуль данных может содержать некоторое количество компонентов доступа к данным (компонентов соединений и компонентов, инкапсулирующих набор данных).

Для обеспечения передачи данных клиентам удаленный модуль данных обязательно должен содержать необходимое количество компонентов TDataSetProvider, каждый из которых должен быть связан с соответствующим набором данных.

Обмен данными сервера приложения с клиентами обеспечивает динамическая библиотека MIDAS.DLL, которая должна быть зарегистрирована на компьютере сервера приложения.

## 2.5. Клиентское приложение

Клиентское приложение в трехзвенной модели должно обладать лишь минимально необходимым набором функций, делегируя большинство операций по обработке данных серверу приложений.

В первую очередь удаленное клиентское приложение должно обеспечить соединение с сервером приложений. Для этого используются компоненты соединений DataSnap:

- TDCOMConnection — использует DCOM;
- TSocketconnection — использует сокет Windows;
- TWebConnection — использует HTTP.

Компоненты соединения DataSnap предоставляют интерфейс IAppServer, используемый компонентами-провайдерами на стороне сервера и компонентами TClientDataSet на стороне клиента для передачи пакетов данных.

Для работы с наборами данных используются компоненты TClientDataSet, работающие в режиме кэширования данных.

Для представления данных и создания пользовательского интерфейса в клиентском ПО применяются стандартные компоненты со страницы **Data Controls** палитры компонентов.

## 2.6. Механизм удаленного доступа к данным DataSnap

Для передачи пакетов данных между компонентом-провайдером и клиентским набором данных, т.е. между клиентом и сервером (рис.2.2), должен существовать некий транспортный канал, обеспечивающий физическую передачу данных. Для этого могут использоваться разнообразные транспортные протоколы, поддерживаемые операционной системой.

Различные типы соединений, позволяющие настроить транспорт и начать передачу и прием данных, инкапсулированы в нескольких компонентах DataSnap. Для создания соединения с тем или иным транспортным протоколом разработчику достаточно перенести соответствующий компонент на форму и правильно настроить несколько свойств. Ниже рассматриваются варианты настройки транспортных протоколов для компонентов, использующих DCOM, сокет TCP/IP, http.

## 2.7. Компонент TDCOMConnection

Компонент TDCOMConnection предоставляет транспорт на основе технологии Distributed COM и применяется в основном для организации транспорта в рамках локальной сети.

Для настройки соединения DCOM в первую очередь необходимо задать имя компьютера, на котором функционирует сервер приложений. Для компонента TDCOMConnection это должен быть зарегистрированный сервер *автоматизации*. Имя компьютера задается свойством *property* ComputerName: string.



Если оно задано правильно, в списке свойства *property* `ServerName: string`, в *инспекторе объектов* можно выбрать один из доступных серверов. При выборе сервера также автоматически заполняется свойство *property* `ServerGUID: string`. Причем для успешного соединения клиента с сервером приложений оба свойства должны быть заданы в обязательном порядке. Только имя сервера или только его GUID не обеспечат правильный доступ к удаленному объекту COM.

Открытие и закрытие соединения осуществляется свойством *property* `Connected: Boolean` или методами *procedure* `Open`, *procedure* `Close` соответственно.

Для организации передачи данных между клиентом и сервером компонент `TDCOMConnection` предоставляет интерфейс `IAppServer` *property* `AppServer: Variant`, который также может быть получен методом *function* `GetServer: IAppServer - override`.

Свойство *property* `ObjectBroker: TCustomObjectBroker` позволяет использовать экземпляр компонента `TsimpleObjectBroker` для получения списка доступных серверов во время выполнения приложения.

Методы-обработчики компонента `TDCOMConnection` представлены в таблице 2.1.

Таблица 2.1

Методы-обработчики событий компонента `TDCOMConnection`

Объявление	Описание
<code>property AfterConnect: TNotifyEvent;</code>	Вызывается после установления соединения.
<code>property AfterDisconnect: TNotifyEvent;</code>	Вызывается после разрыва соединения.
<code>property BeforeConnect: TNotifyEvent;</code>	Вызывается перед установлением соединения.
<code>property BeforeDisconnect: TNotifyEvent;</code>	Вызывается перед разрывом соединения.

<pre> type TGetUsernameEvent = procedure ( Sender : TObject ; var Username: string) of ob- ject; property OnGetUsername : TGetUsernameEvent ; </pre>	<p>Вызывается непосредственно перед появлением диалога удаленной авторизации пользователя. Для этого свойство LoginPrompt должно иметь значение True. Параметр Username может содержать имя пользователя по умолчанию, которое появится в диалоге.</p>
<pre> type TLoginEvent = procedure ( Sender: TObject; Username, Password: string) of object; property OnLogin: TLoginEvent; </pre>	<p>Вызывается после открытия соединения, если свойство LoginPrompt имеет значение True. Параметры Username и Password содержат имя пользователя и пароль, введенные при авторизации.</p>

## 2.8. Компонент TSocketConnection

Компонент TSocketConnection обеспечивает соединение клиента с сервером приложений за счет использования сокетов TCP/IP. Для успешного открытия соединения на стороне сервера должен работать сокет-сервер (приложение ScktSrvr.exe, рис.2.4).

Для успешного соединения свойство *property* Host: String должно содержать имя компьютера сервера.

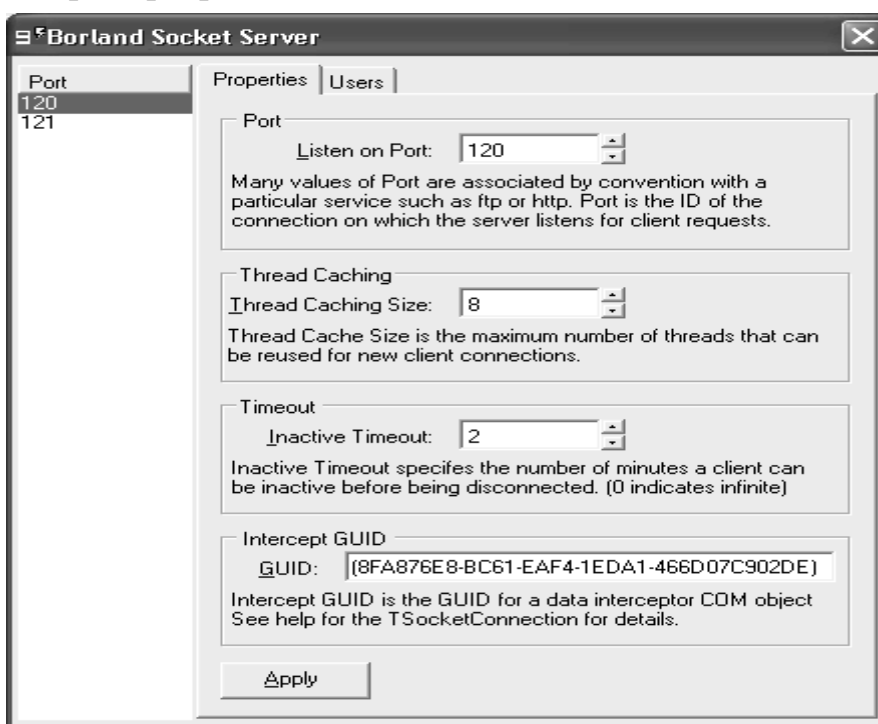


Рис.2.4. Сокет-сервер ScktSrvr.exe.

Дополнительно, свойство *property* Address: String должно содержать IP-адрес сервера.

Для открытия соединения должны быть заданы оба этих свойства.

Свойство *property* Port: Integer устанавливает номер используемого порта. По умолчанию это порт **211**, но разработчик волен изменить порт, например, для использования различными категориями пользователей или для создания защищенного канала.

После правильного выбора компьютера в списке свойства *property* ServerName: string в *инспекторе объектов* появляется перечень доступных серверов *автоматизации*. После выбора сервера свойство *property* ServerGUID: string, которое содержит имя компьютера GUID зарегистрированного сервера, задается автоматически, хотя его можно задать и вручную.

Метод *function* GetServerList: OleVariant virtual, возвращает список зарегистрированных серверов *автоматизации*. Открытие и закрытие соединения осуществляется свойством *property* Connected: Boolean или методами *procedure* Open, *procedure* Close соответственно.

Канал сокета TCP/IP может быть зашифрован. Для этого используется свойство *property* InterceptName: string, содержащее программный идентификатор объекта COM, обеспечивающего шифрование/дешифрование данных в канале, и свойство *property* InterceptGUID: string, содержащее имя компьютера GUID этого объекта.

Этот COM объект перехватывает данные в канале и осуществляет их обработку, предусмотренную собственным программным кодом. Это может быть шифрование, сжатие, обработка шумов и т. д.

**Примечание.** Создание объекта COM, обеспечивающего дополнительную обработку данных в канале, ложится на плечи разработчика. Объект-перехватчик должен поддерживать стандартный интерфейс *IDataintercept*.

Естественно, на стороне сервера должен быть зарегистрирован объект COM, выполняющий обратную операцию. Для этого также используется сокет-сервер (рис.2.5). Строка **Interceptor** на странице должна содержать имя компьютера GUID объекта-перехватчика COM.

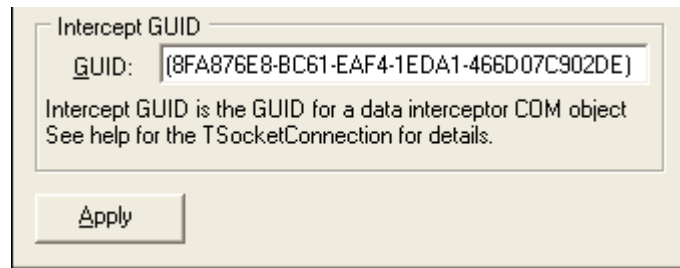


Рис.2.5. Регистрация объекта-перехватчика COM в сокет-сервере

Метод *function* **GetInterceptorList: OleVariant; virtual** возвращает список зарегистрированных на сервере объектов-перехватчиков.

Для организации передачи данных между клиентом и сервером компонент `TSocketConnection` предоставляет интерфейс **IAppServer** *property* `AppServer: Variant`, который также может быть получен методом *function* **GetServer: IAppServer; override**.

Свойство *property* **ObjectBroker: TCustomObjectBroker** позволяет использовать экземпляр компонента `TSimpleObjectBroker` для получения списка доступных серверов во время выполнения.

## 2.9. Компонент `TWebConnection`

Компонент `TWebConnection` предоставляет клиенту соединение на основе транспорта HTTP. Для работы компонента на клиентском компьютере должна быть зарегистрирована библиотека `wininet.dll`. Обычно это не требует специальных усилий, т. к. этот файл уже имеется в системной папке Windows, если на компьютере установлен Internet Explorer.

На компьютере сервера должен быть инсталлирован Internet Information Server версии не ниже 4.0 или Netscape Enterprise версии не ниже 3.6. Перечисленное ПО обеспечивает доступ компонента `TWebConnection` к динамической библиотеке `HTTPsrvr.dll`, которая также должна находиться на сервере.

Например, если файл `HTTPsrvr.dll` расположен в папке `Scripts US 4.0` на Web-сервере **www.someserver.com**, то свойство *property* **URL: string** должно содержать следующее значение: **http://someserver.com/scripts/httpsrvr.dll**

Если URL задан верно и сервер настроен правильно, то в списке свойства *property* **ServerName: string** в инспекторе объектов появляется перечень зарегистрированных серверов приложений. Имя одного из них должно содержаться в свойстве `ServerName`.

После выбора имени сервера в свойстве *property* **ServerGUID: string** автоматически появляется GUID сервера.

Свойства *property* **UserName: string** и *property* **Password: string** при необходимости могут содержать имя и пароль пользователя, которые будут использованы при авторизации.

Свойство *property* **Proxy: string** содержит имя используемого прокси-сервера.

В заголовок сообщений HTTP можно поместить имя приложения. Для этого используется свойство *property* **Agent: string**.

Соединение открывается и закрывается при помощи свойства *property* **Connected: Boolean**. Аналогичные операции выполняют методы *procedure* **Open** и *procedure* **Close**.

Доступ к интерфейсу **IAppServer** предоставляет свойство *property* **AppServer: Variant** или метод *function* **GetServer: IAppServer; override**.

Список доступных соединению серверов приложений возвращает метод *function* **GetServerList: OleVariant; virtual**.

Свойство *property* **ObjectBroker: TCustomObjectBroker** позволяет использовать экземпляр компонента **TSimpleObjectBroker** для получения списка доступных серверов во время выполнения.

Методы-обработчики событий компонента **TWebConnection** полностью совпадают с методами-обработчиками компонента **TDCOMConnection** (см. табл. 2.1).

## 2.10. Провайдеры данных

Компонент-провайдер **TDataSetProvider** представляет собой мост между набором данных сервера приложений и клиентским набором данных. Он обеспечивает формирование и передачу пакетов данных клиентскому приложению и прием от него сделанных изменений (см. рис.2.2).

Все необходимые операции компонент выполняет автоматически. Разработчику необходимо лишь разместить компонент **TDataSetProvider** и связать его с набором данных сервера приложений. Для этого предназначено свойство *property* **DataSet: TDataSet**.

Если соединение в клиентском приложении настроено правильно, то в списке выбора свойства **ProviderName** компонента **TClientDataSet** в Инспекторе объектов появляются имена всех компонентов-провайдеров сервера приложе-

ний. Если связать клиентский набор данных с компонентом-провайдером, а затем открыть его, в клиентский набор данных будут переданы записи из набора данных сервера приложений, указанного в свойстве DataSet компонента-провайдера TDataSetProvider.

Компонент также содержит свойства, помогающие настроить процесс обмена данными.

Свойство *property* ResolveToDataSet: Boolean управляет передачей данных от клиента серверу БД. Если оно имеет значение True, все изменения передаются в набор данных сервера приложений, заданный свойством DataSet. Иначе изменения направляются напрямую серверу БД. Если сервер приложений не должен отображать сделанные клиентом изменения, то свойству ResolveToDataSet можно присвоить значение False, что ускорит работу приложения.

Свойство *property* Constraints: Boolean управляет передачей ограничений серверного набора данных клиентскому. Если свойство имеет значение True, ограничения передаются.

Свойство *property* Exported: Boolean позволяет использовать в клиентском наборе данных интерфейс IAppServer. Для этого свойство должно иметь значение True.

Параметры компонента-провайдера задаются свойством **type**:

- TProviderOption = (poFetchBlobsOnDemand, poFetchDetailsOnDemand, poIncFieldProps, poCascadeDeletes, poCascadeUpdates, poReadOnly, poAllowMultiRecordUpdates, poDisableInserts, poDisableEdits, poDisableDeletes, poNoReset, poAutoRefresh, poPropagateChanges, poAllowCoinmandText, poRetainServerOrder);

- TProviderOptions = set of TProviderOption;

- (набор параметров свойства задается присвоением элементам значения True)

- property Options: TProviderOptions;

- poFetchBlobsOnDemand — включает передачу в клиентский набор данных значений полей типа BLOB. По умолчанию эта возможность, отключена для ускорения работы;

- poFetchDetailsOnDemand — включает передачу в клиентский набор данных подчиненных записей для отношения «один-ко-многим». По умолчанию эта возможность отключена для ускорения работы;

- `poIncFieldProps` — включает передачу в клиентский набор данных нескольких свойств для объектов полей: `Alignment`, `DisplayLabel`, `DisplayWidth`, `Visible`, `DisplayFormat`, `EditFormat`, `MaxValue`, `MinValue`, `Currency`, `EditMask`, `DisplayValues`;
- `poCascadeDeletes` — включает автоматическое удаление подчиненных записей в отношении «один-ко-многим» на стороне сервера, если главная запись была удалена в клиентском наборе данных;
- `poCascadeUpdates` — включает автоматическое обновление подчиненных записей в отношении «один-ко-многим» на стороне сервера, если главная запись была изменена в клиентском наборе данных;
- `poReadOnly` — включает режим «только для чтения» для набора данных сервера;
- `poAllowMultiRecordUpdates` — включает режим внесения изменений сразу в несколько записей одновременно. Иначе все записи изменяются последовательно, одна за одной;
- `poDisableInserts` — запрещает клиенту вносить в набор данных сервера новые записи;
- `poDisableEdits` — запрещает клиенту вносить в набор данных сервера изменения;
- `poDisableDeletes` — запрещает клиенту удалять записи в наборе данных сервера;
- `poNoReset` — запрещает обновление набора данных сервера перед передачей записей клиенту (перед вызовом метода `AS_GetRecrds` интерфейса `IAppServer`);
- `poAutoRefresh` — включает автоматическое обновление записей клиентского набора данных. По умолчанию эта возможность отключена для ускорения работы;
- `poPropagateChanges` — если в методах-обработчиках `BeforeUpdateRecord` или `AfterUpdateRecord` клиентского набора данных были сделаны дополнительные изменения, то после их записи в наборе данных сервера, изменения снова направляются клиенту для обновления записи. Во включенном состоянии эта возможность позволяет полностью контролировать сохранение изменений на сервере;

- `poAllowCommandText` — позволяет изменять текст запроса SQL, имена хранимых процедур или таблиц в компоненте набора данных на сервере приложений;
- `poRetainServerOrder` — включает запрет на изменение порядка сортировки записей клиентом. Если этот параметр отключить, возможны ошибки отображения набора данных, проявляющиеся в появлении двойных записей.

## 2.11. Пример создания многопользовательской распределенной системы

Целью разработки данного приложения является приобретение навыков создания многопользовательской распределенной системы трехзвенной архитектуры: **BDE → Сервер приложения → Тонкий клиент**, на примере ведения электронного документа «НАКЛАДНАЯ». Структура электронного документа представляет собой единственную таблицу, внутренние отношения в которой реализуются в клиентской части системы.

### Создание таблицы базы данных

Создайте личную папку, например папку с именем **M3\_LAB**, в которой создайте три вложенных папки с именами **DATA**, **SERVER**, **CLIENT**. Используя утилиту **DataBaseDesktop**, создайте **Alias** с уникальным именем (например, **M3\_TEST**) к папке **DATA**. Создайте таблицу **Paradox 7**, структура которой приведена на рис.2.6 и сохраните ее в папке **DATA** под именем **Invoice**.

	Field Name	Type	Size	Key
1	AutoNum	+		*
2	INX	N		
3	Caption	A	30	
4	Count	N		
5	Salary	\$		

Рис. 2.6. Структура таблицы



## Разработка MIDAS сервера приложения

Создайте новый проект в среде Delphi. Присвойте форме имя **SVRForm**. Свойство **FormStyle** установите в состояние **fsStayOnTop**. Выполните дизайн формы в соответствии рис.2.7.



Рис. 2.7. Дизайн формы сервера

В обработчике события активизации формы напишите код, соответствующий положению формы при запуске сервера.

```
procedure TSVRForm.FormActivate(Sender: TObject);  
begin  
    Left := 0; Top := 0;  
end;
```

Сохраните созданный проект в папке **SERVER** с именами **Unit1** → **M3\_SRV**, **Project1** → **M\_SERVER**.

Скомпилируйте проект.

Выберите пункт меню **File / New / Other** и на странице **Multier** выберите **Remote Data Module** (рис.2.8.).

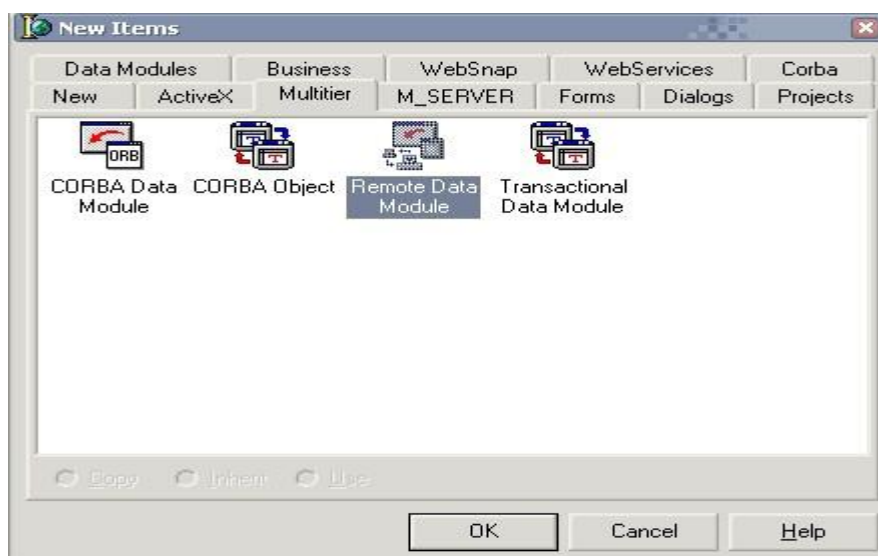


Рис. 2.8. Выбор удаленного модуля

В окне регистрации класса модуля введите уникальное имя нового класса, например **NICSVR** (рис.2.9).



Рис. 2.9. Объявление нового класса

После выполнения команды объявления класса появится окно модуля (рис.2.10).



Рис. 2.10. Окно удаленного модуля

Сохраните проект, выполнив команду **SaveAll**, при этом измените имя **Unit'a** модуля класса на уникальное имя, например **TSR**.

Скомпилируйте проект.

Установите в окно удаленного модуля компонент **Table** и компонент **DataSetProvider** (рис.2.11).

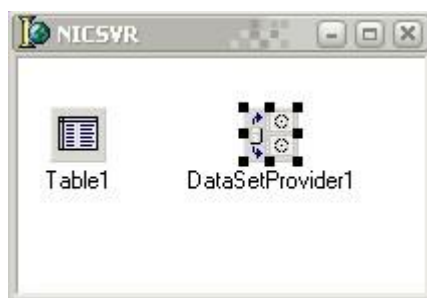


Рис. 2.11. Установка компонентов доступа к данным

Соедините компонент **Table1** с базой данных, в нашем случае **M3\_TEST** и таблицей **Invoice.db**. Соедините **DataSetProvider** с **Table1** посредством свойства **DataSet**. Для активизации обмена данными сначала щелкните правой кнопкой мыши, а затем дважды щелкните левой кнопкой мыши по компоненту **DataSetProvider**.

```
procedure TNICSVR.DataSetProvider1AfterApplyUpdates(Sender: TObject;  
    var OwnerData: OleVariant);  
begin  
  
end;
```

Сохраните и скомпилируйте проект. Закройте разработанное приложение.

## Разработка MIDAS тонкого клиента

Создайте новый проект в Delphi и выполните предварительный дизайн формы, согласно рис.2.12.



Рис.2.12. Предварительный дизайн приложения Client

Сохраните созданный проект в папке **CLIENT** с именами **Unit1** → **CLN**, **Project1** → **CLIENT**.

Сохраните и скомпилируйте проект.


Установите в форму компонент **DCOMConnection** и выберите в свойстве **ServerName** имя зарегистрированного сервера, в нашем случае **M\_SERVER.NICSERVER** (рис. 2.13).



Рис. 2.13. Выбор имени сервера

Установите в форму два компонента **ClientDataSet** которые поочередно свяжите с сервером приложения посредством выбора значения свойства **RemoteServer := DCOMConnection1**. В свойствах **ProviderName** компонентов **ClientDataSet** выберите значение **DataSetProvider1**.

Обратите внимание на тот факт, что при выборе значения свойства **ProviderName** автоматически вызовется приложение сервера, форма которого будет выведена в верхнем левом углу монитора.

***Примечание:** Если свойство **ProviderName** пустое, то сохраните проект и закройте приложение. Затем откройте ранее скомпилированный проект сервера и повторно выполните операции щелчков мышью по компоненту **DataSetProvider**. Также обратите внимание на тот факт, что если открылась утилита регистрации (рис.2.14), то необходимо вызвать повторную регистрацию, выполнив команду **Register Type Library** . В случае если утилита регистрации не появилась, необходимо ее предварительно вызвать, выполнив пункт меню **View / Tuype Library Delphi**. После выполнения указанных действий скомпилируйте приложение Сервер, сохраните его и вернитесь к разработке клиентского приложения, повторив указанные ранее действия для компонентов **ClientDataSet**.*

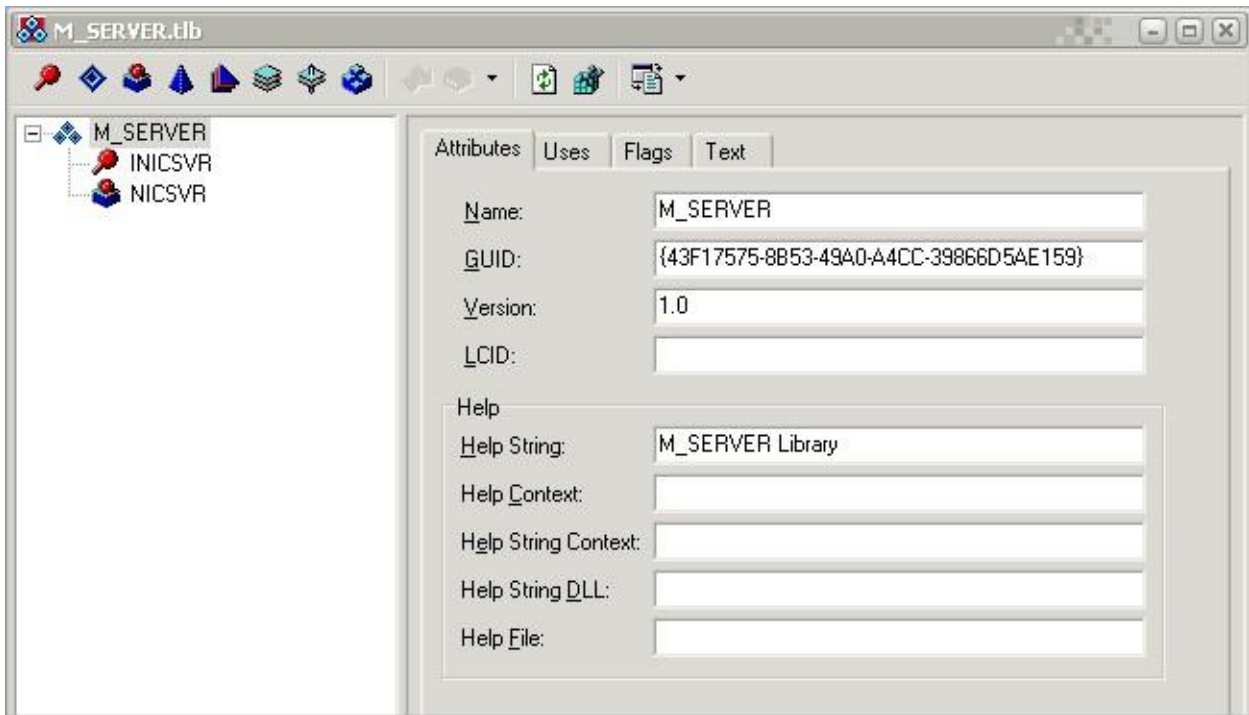


Рис.2.14. Утилита регистрации классов.

После удачного соединения с сервером установите в форму элементы связи и управления (рис.2.15) которые соедините между собой. Для компонентов **ClientDataSet** создайте доменную структуру для представлений.

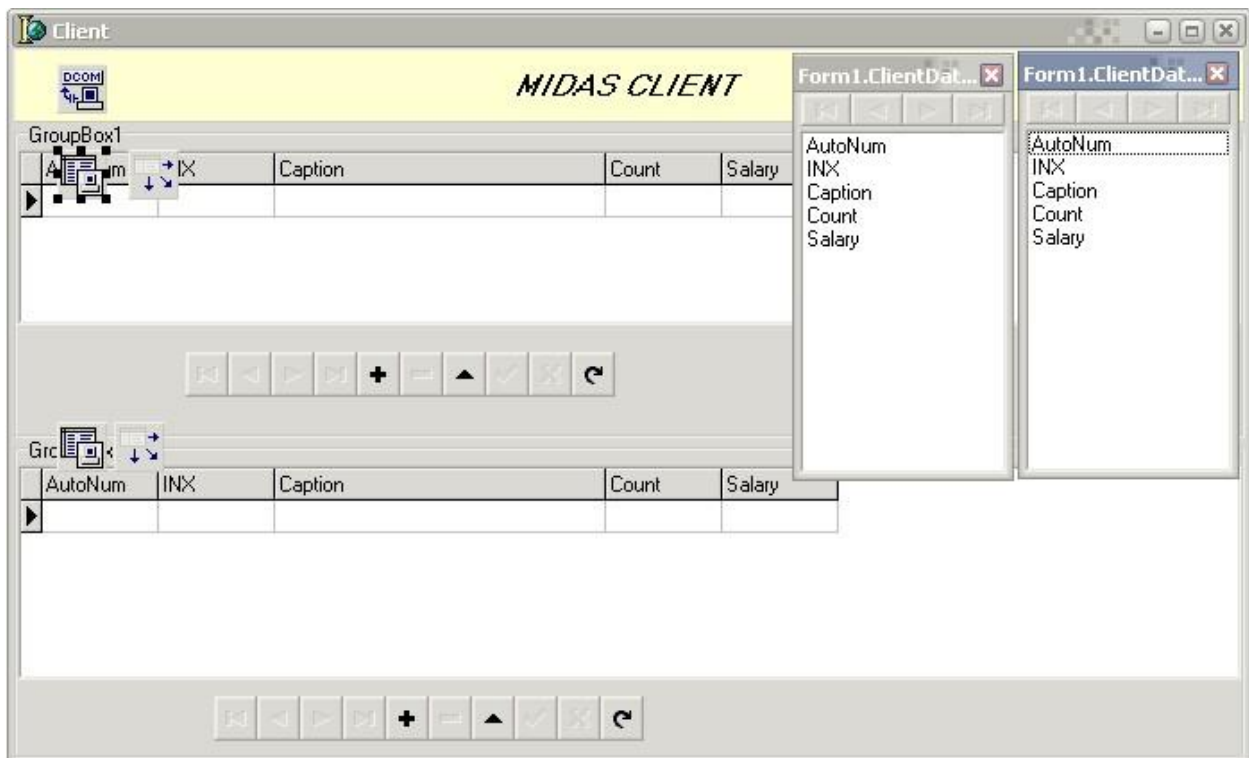


Рис.2.15. Дизайн приложения **Client**

Установите динамическое отношение 1:M для представлений **ClientDataSet** используя методы фильтра, для чего введите значение свойства **Filter** для **ClientDataSet1** ( $\text{Filter} \rightarrow \text{INX} = 0$ ) и свойство **Filtered := True**, а для **ClientDataSet2** напишите личную процедуру **LinkFilter**, которую декларируйте в разделе **Private**.

```

private
  { Private declarations }
procedure LinkFilter;
public
  { Public declarations }
end;

procedure TForm1.LinkFilter;
begin
  ClientDataSet2.Filtered := false;
  ClientDataSet2.Filter := 'INX =' + QuotedStr(ClientDataSet1.AutoNum.AsString);
  ClientDataSet2.Filtered := true;
end;

```

В соответствующих обработчиках событий **ClientDataSet1** реализуйте методы предварительной установки значений и взаимодействий с сервером приложений, а также связь с **ClientDataSet1** посредством вызова процедуры личного фильтра (**LinkFilter**).

```

procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
  ClientDataSet1.INX.Value := 0;
end;

procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
  ClientDataSet1.ApplyUpdates(-1);
  ClientDataSet1.Refresh;
end;

procedure TForm1.ClientDataSet1AfterDelete(DataSet: TDataSet);
begin
  ClientDataSet1.ApplyUpdates(-1);
  ClientDataSet1.Refresh;
end;

procedure TForm1.ClientDataSet1AfterScroll(DataSet: TDataSet);
begin
  if ClientDataSet1.AutoNum.IsNull = false then LinkFilter;
end;

```

В соответствующих обработчиках событий **ClientDataSet2** также реализуйте методы предварительной установки значений и взаимодействий с сервером приложений и фильтрации данных, относящихся к выбранной записи в представлении **ClientDataSet1**:

```
procedure TForm1.ClientDataSet2AfterDelete(DataSet: TDataSet);
begin
  ClientDataSet2.ApplyUpdates(-1);
  ClientDataSet2.Refresh;
  LinkFilter;
end;

procedure TForm1.ClientDataSet2AfterPost(DataSet: TDataSet);
begin
  ClientDataSet2.ApplyUpdates(-1);
  ClientDataSet2.Refresh;
  LinkFilter;
end;

procedure TForm1.ClientDataSet2AfterInsert(DataSet: TDataSet);
begin
  ClientDataSet2.INX.Value := ClientDataSet1.AutoNum.Value;
end;
```

Скомпилируйте проект и сохраните приложение.

Выйдите из Delphi и запустите клиентское приложение из папки **CLIENT**. Обратите внимание на тот факт, что при запуске клиентского приложения автоматически запускается приложение **SERVER** (рис.2.16), а при закрытии клиентского приложения экземпляр приложения также **SERVER** закрывается.

Введите несколько записей в поле **Caption** представления «Предприятия» (рис.2.17). Обратите внимание на тот факт, что поля **AutoNum** и **INX** заполняются автоматически.

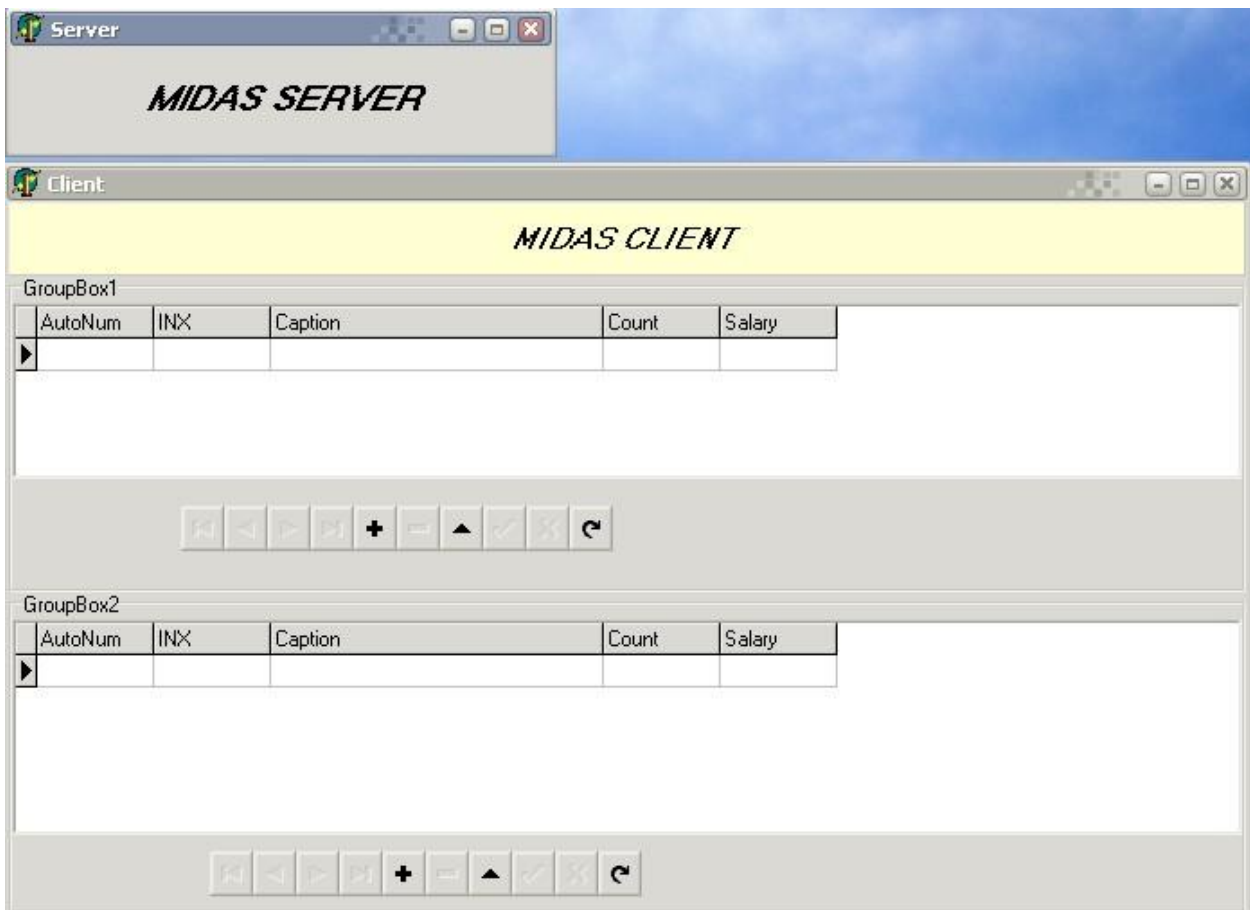


Рис. 2.16. Общий вид тонкого клиента и экземпляра сервера

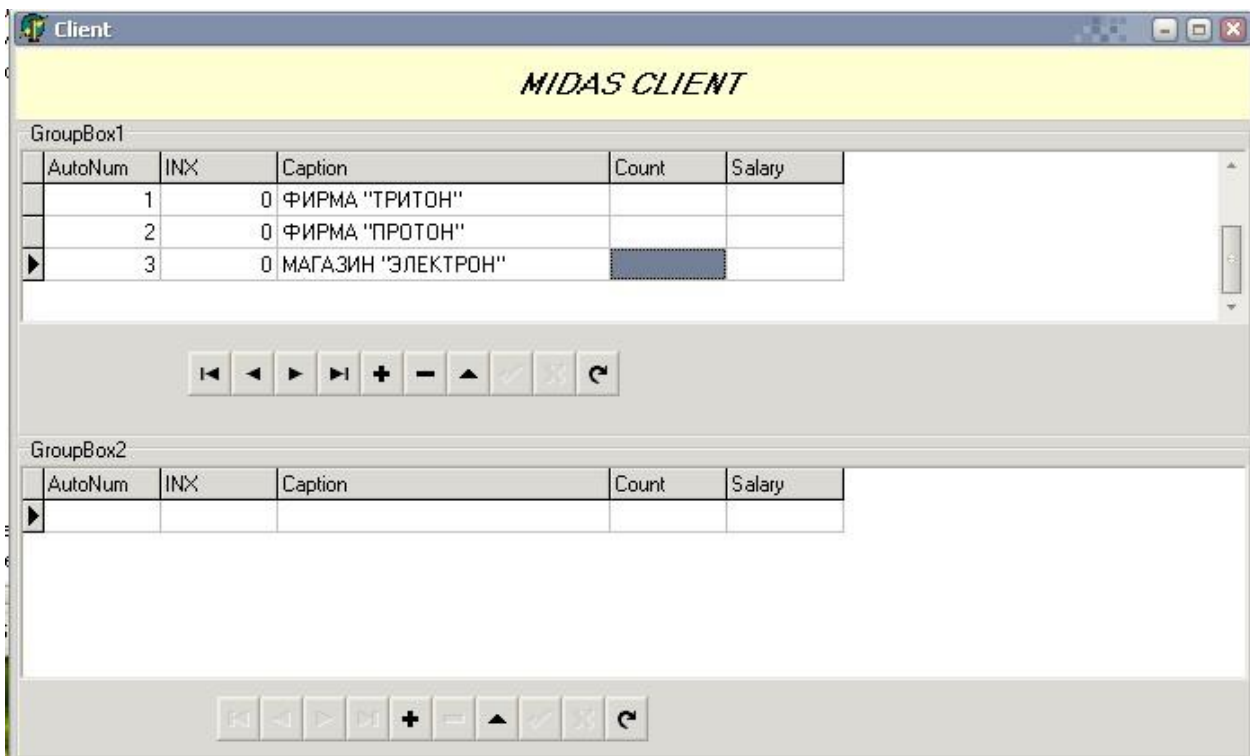


Рис. 2.17. Заполнение полей Caption представления «Предприятия»



Относительно каждой записи представления предприятия создайте несколько записей в представлении «Отпущенный товар» (рис.2.18). Обратите внимание на тот факт, что поля **AutoNum** и **INX** представления «Отпущенный товар» также заполняются автоматически.

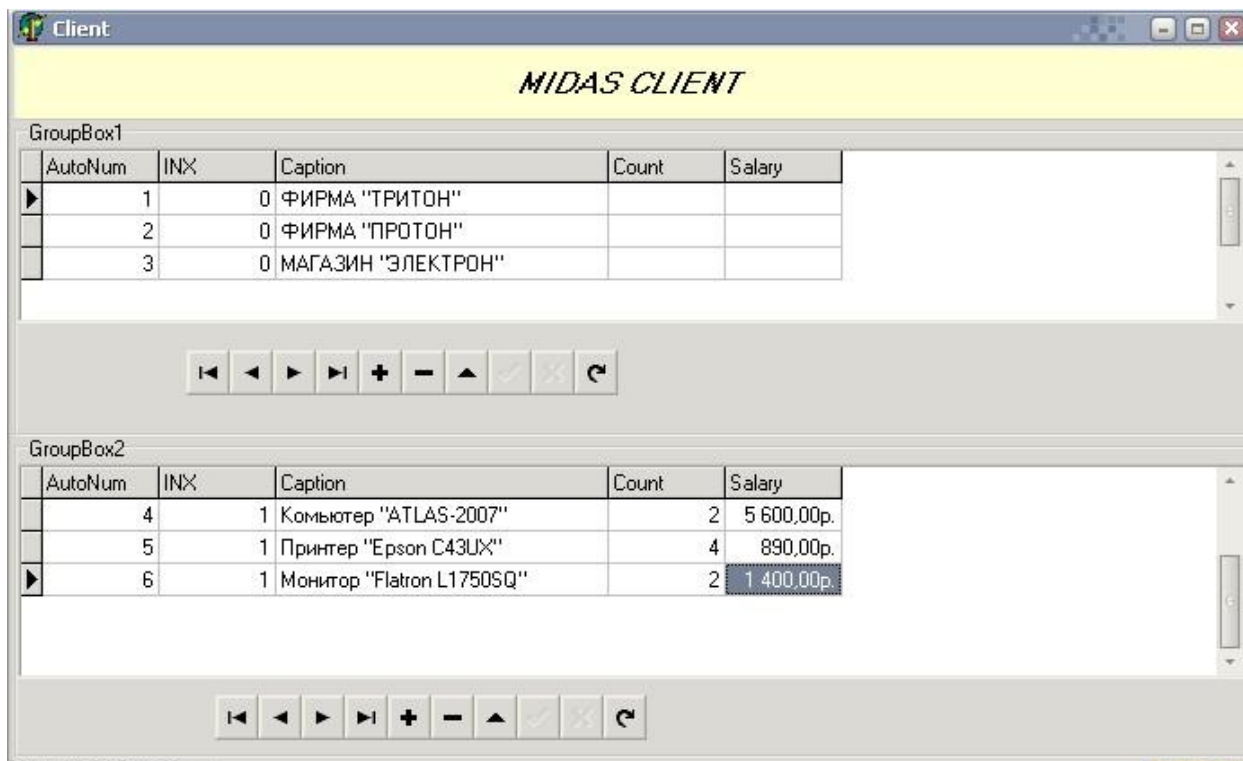


Рис. 2.18. Заполнение полей Caption, Count, Salary представления ” Отпущенный товар ”

Вернитесь в среду Delphi и продолжите разработку клиентского приложения.

Закройте доступ к данным компонента **ClientDataSet2**. Вызовите редактор полей набора данных **ClientDataSet2**, выполнив двойной щелчок по компоненту и пункт контекстного меню New Field. Создайте в редакторе вычисляемое поле **AllSalary** (рис.2.19), а затем переведите **ClientDataSet2** в активное состояние.

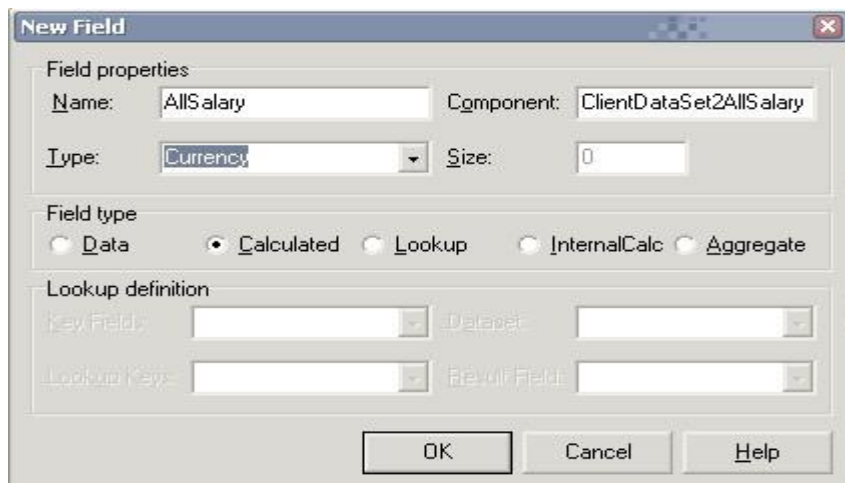


Рис. 2.19. Создание вычисляемого поля AllSalary

Для вычисления общей суммы создайте в обработчике события OnCalcFields следующий метод:

```

procedure TForm1.ClientDataSet2CalcFields(DataSet: TDataSet);
begin
    ClientDataSet2AllSalary.Value :=
    ClientDataSet2Salary.Value * ClientDataSet2Count.Value;
end;

```

Завершите дизайн приложения, приведя его к виду, показанному на рис.2.20.

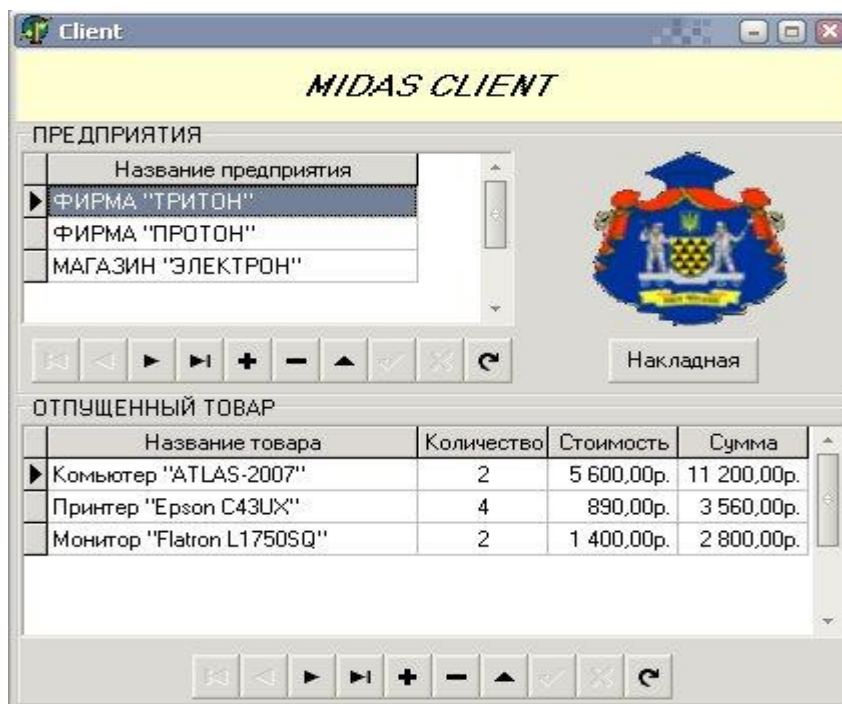


Рис. 2.20. Вид завершеного приложения

Предлагается самостоятельно разработать печатный документ накладной на товар.

### Сетевой режим работы

Для запуска приложения на другом компьютере, участнике сети, необходимо создать на удаленном компьютере сетевой диск, используя пункт меню *Сервис/Подключить сетевой диск* проводника Windows. Сетевым диском должен быть диск компьютера – сервера, на котором находится разработанные приложения **SERVER** и **CLIENT**. Для регистрации экземпляра класса сервера на удаленном компьютере необходимо первоначально запустить приложение **SERVER**, а затем **CLIENT**. В дальнейшем, приложение **SERVER** будет запускаться автоматически при запуске приложения **CLIENT**, а для **CLIENT**а можно создать ярлык на рабочем столе.

## ЛИТЕРАТУРА

1. Архангельский А. Программирование в Delphi 7. – К.: Каравелла, 2008. – 832 с.
2. Бобровский С. Delphi 7. Учебный курс. – Ростов на Дону: Феникс, 2008. – 452 с.
3. Культин Н. С. Delphi 7. Программирование на Object Pascal. – СПб.: БХВ - Петербург, 2009. – 342 с.
4. Марко Кэнтю. Delphi 7 для профессионалов. – СПб: Питер, 2008. – 453 с.
5. Стивенс Род. Delphi. Готовые алгоритмы. – СПб.: Питер, 2009. – 312 с.
6. Шпак Ю. А. Delphi 7 на примерах. – СПб.: БХВ - Петербург, 2008. – 287 с.
7. Фаронов В.В. Delphi программирование на языке высокого уровня. СПб.: – Питер, 2010. – 544 с.
8. Фленов М., Библия Delphi (3-е издание). – СПб.: ВНУ, 2010. – 740 с.
9. Хомоненко А., Гофман В., Мещеряков Е., Никифоров В. Delphi 7. Наиболее полное руководство. – СПб.: БХВ-Петербург, 2010. – 616 с.
10. Юркин А. Г. Задачник по программированию. – СПб.: Питер, 2009. – 310 с.

## СОДЕРЖАНИЕ

1. ОСНОВЫ ТЕХНОЛОГИИ ADO .....	3
1.1. Провайдеры ADO .....	4
1.2. Компоненты ADO .....	5
1.3. Механизм соединения с хранилищем данных ADO .....	6
1.4. Компонент TADOConnection .....	6
1.5. Наборы данных ADO .....	7
1.6. Компонент TADODataSet .....	8
1.7. Компонент TADOTable .....	8
1.8. Компонент TADOQuery .....	9
1.9. Компонент TADOStoredProc .....	9
1.10. Пример приложения ADO клиент-сервер СУБД на основе ADOTable	10
1.11. Пример приложения ADO с использованием компонента ADOConnection .....	25
2. ТЕХНОЛОГИЯ MIDAS .....	32
2.1. Структура многозвенного приложения в Delphi .....	33
2.2. Трехзвенное приложение в Delphi .....	35
2.3. Сервер приложений .....	37
2.4. Структура сервера приложения .....	39
2.5. Клиентское приложение .....	39
2.6. Механизм удаленного доступа к данным DataSnap .....	40
2.7. Компонент TDCOMConnection .....	40
2.8. Компонент TSocketConnection .....	42
2.9. Компонент TWebConnection .....	44
2.10. Провайдеры данных .....	45
2.11. Пример создания многопользовательской распределенной системы .....	48
ЛИТЕРАТУРА .....	60

Навчальне видання

Швачич Геннадій Григорович  
Овсянніков Олександр Васильович  
Петречук Ліна Миколаївна

# КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ В ДІЛОВОДСТВІ

## Розділ 1. ADO та MIDAS технології

Навчальний посібник

Тем. план 2012, поз.243

Підписано до друку 26.06.2012. Формат 60x84 1/16. Папір друк. Друк плоский.  
Облік.-вид. арк. 3,64. Умов. друк. арк. 3,59. Тираж 100 пр. Замовлення № 99.

Національна металургійна академія України  
49600, м. Дніпропетровськ-5, пр. Гагаріна, 4

---

Редакційно-видавничий відділ НМетАУ