1) Создание базы данных для сервера MS SQL Server с помощью утилиты Server Explorer. Структура базы данных MS SQL Server. Создание таблиц базы данных.

Для работы с базами данных среда Visual Studio имеет утилиту **Обозреватель серверов**, которая запускается с помощью команды **Вид/Обозреватель серверов**.

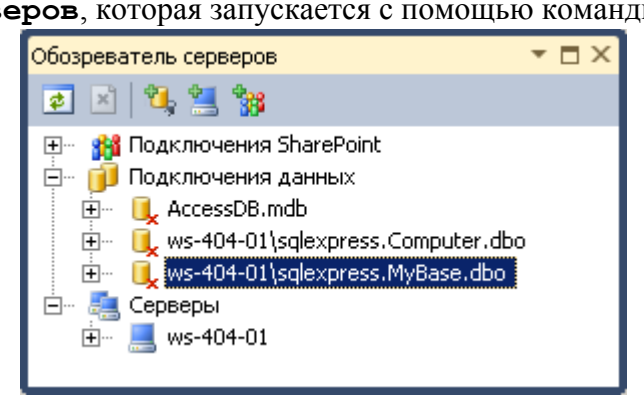


Рис. 1.2. *Фрагмент окна Server Explorer*

Как видно, в этом окне указывается все базы данных, которые соединены на ваш компьютер (на моем компьютере базы данных MyBase, Computer и AccessDB).

Для создания новой базы данных из контекстного меню узла Подключения данных выберите команду **Создать новую базу данных SQL Server**. При этом появляется окно создания новой базы данных.

Как видно, в этом окне в поле Имя сервера следует ввести имя локального SQL-сервера, а в поле Имя новой базы данных – имя базы данных, например, **MyBase**. После нажатия кнопки **OK** в окне Обозреватель серверов появляется новый узел MyBase.

Если на этом узле нажать на символ “+”, то открывается **структура базы данных**.

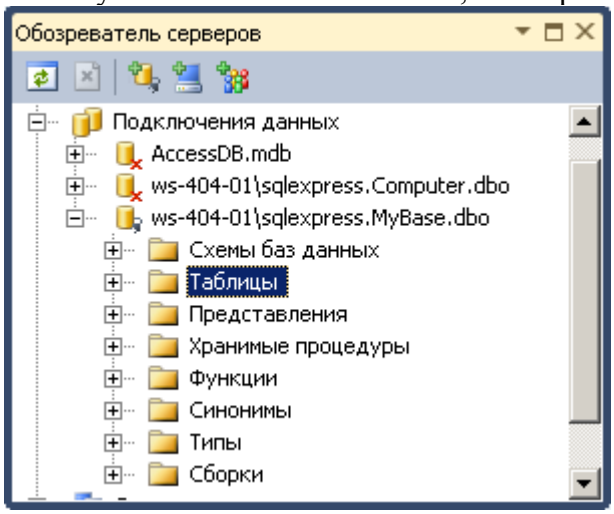


Рис. 1.4. *Структура базы данных MyBase*

При входе в любую базу данных в окне **Обозреватель серверов** открывается список разделов, где отображаются однотипные объекты базы данных:

Схемы базы данных – диаграммы, которые показывают связи между таблицами;

Таблицы – все таблицы базы данных;

Представления – логические таблицы, данные которых отобраны из одного или нескольких таблиц;

Хранимые процедуры – процедуры, которые хранятся в файле базы данных.

Порядок создания таблиц базы данных с помощью утилиты Обозреватель серверов:

Для этого выделяйте узел **Таблицы** и из контекстного меню этого узла вызывайте команду **Добавить новую таблицу**. При этом в рабочей области Visual Studio появляется окно определения структуры таблицы.
После определения структуры таблицы, эту таблицу следует сохранить. Для этого из главного меню выберите команду **Файл/Сохранить Table** и введите имя таблицы, например, Person.
Для заполнения таблицы с данными в окне Обозреватель серверов выберите таблицу и из контекстного меню выполните команду **Показать таблицу данных**. При этом в рабочей области появляется содержимое таблицы, где вы можете ввести данные записей.

2) Провайдеры данных технологии ADO.NET и соответствующие пространства имен. Добавление в проекте ссылку на посторонние сборки (провайдеры). Получение провайдеров данных из Интернета.

Программное обеспечение ADO.NET для подсоединения и взаимодействия с физической базой данных называется провайдером данных ADO.NET. *Провайдер данных (data provider)* — это управляемый код .NET, который эквивалентен провайдеру OLEDB(набор COM-интерфейсов, которые позволяют приложениям унифицировано работать с данными разных источников и хранилищ информации) или драйверу ODBC(программный интерфейс (API) доступа к базам данных). Провайдер данных состоит из нескольких объектов, которые реализуют необходимую функциональность в соответствии с определениями своих классов и интерфейсов.
В настоящее время существует три разных провайдера данных ADO.NET, каждый из которых определен в своем собственном пространстве имен. Для всех объектов в этих пространствах имен используются следующие префиксы: OleDb, Sql и Odbc.

Провайдер данных SqICIient

Оптимизирован для работы с SQL Server версии 7.0 (или выше) и позволяет добиться более высокой производительности по следующим причинам:
• взаимодействует с базой данных непосредственно через собственный протокол табличной передачи данных (Tabular Data Stream — TDS), а не через OLEDB с отображением интерфейса OLEDB на протокол TDS;
• исключает накладные расходы, связанные с использованием COM-служб взаимодействия;
• отсутствуют ненужные функции, которые не поддерживаются в SQL Server (объекты этого провайдера данных находятся в пространстве имен System.Data.SqlClient).

Провайдер данных Oledb

Основан на существующем COM-поставщике OLEDB и COM-службах взаимодействия платформы .NET Framework, предназначенных для доступа к базе данных. Этот провайдер данных используется для работы с SQL Server более ранних версий, чем 7.0. Он позволяет осуществлять доступ к любой базе данных, для которой имеется поставщик OLEDB. Объекты этого провайдера данных находятся в пространстве имен System.Data.Oledb.

Провайдер данных Odbc

Используется для доступа к базам данных, которые не имеют собственного провайдера данных .NET или COM-поставщика OLEDB. Объекты этого провайдера данных находятся в пространстве имен System.Data.Odbc.

Добавление в проекте ссылку на посторонние сборки

Есть два способа в зависимости от того, какие библиотеки тебе надо добавить.

1. Если надо добавить ссылку на сборку, которая есть в стандартной поставке Visual Studio, то в начале кода просто допиши using <название сборки>.
2. Если надо добавить внешнюю ссылку, то в дереве проекта (обозреватель решений) в разделе "Ссылки" нажми правой кнопкой и выбери пункт "Добавить ссылку" и там выбираешь то, что надо.

Провайдеры данных можно загрузить с сайта Microsoft.com.

3) Классы, для работы с удаленной базой данных. Классы для работы с локальной копией базы данных. Компоненты Visual Studio для работы с базой данных.

Выше в примере был указан простейший способ получения данных из MS SQL Server, где для скачивания данных был использован объект класса SqlDataAdapter. В общем случае, имеются различные классы, которые предназначены для соединения к базе данных, формирования и выполнения SQL-запроса. Ниже приведено использование еще двух классов технологии ADO.NET, которые предназначены для работы с базой данных MS SQL Server.

1. Класс **SqlConnection** – используется для установления соединения с базой данных, которая хранится в MS SQL-сервере.

2. Класс **SqlCommand** – применяется для формирования SQL-запроса, либо для вызова хранимой процедуры.

Эти классы имеют различных конструкторов, которые различаются друг от друга количеством и типом параметров. Для демонстрации работы с этими классами объявите следующие глобальные переменные и переопределите код обработчика события Form1_Load следующим образом:

```
public partial class Form1 : Form
{
   SqlConnection conn;
   SqlCommand cmd;

   SqlDataAdapter adapPerson;
   DataTable Person;
 ..............................

private void Form1_Load(object sender, EventArgs e)
{
    string strConn = "Data Source=(local)\\SQLEXPRESS; " +
                     "Initial Catalog=MyBase;" +
                     "Integrated Security=True; Pooling=False";

    conn = new SqlConnection(strConn);
    conn.Open();

    string strCmd = "SELECT Name,Post,Oklad, BirthDay FROM Person";


    cmd = new SqlCommand(strCmd, conn);

    adapPerson = new SqlDataAdapter(cmd);

    Person = new DataTable();
    adapPerson.Fill(Person);

    dataGridView1.DataSource = Person;
}
```

Заметим, что удобством использования объектов этих классов является то, что эти объекты создаются только один раз, а затем они могут применяться в различных участках программы. Поэтому в дальнейших примерах будут использованы объекты этих классов.

В технологии ADO.NET для копирования таблиц из удаленного SQL-сервера на клиентский компьютер применяется объект класса `SqlDataAdapter` ***(адаптер)****. При создании объекта этого класса здесь указан SQL-запрос и строка подключения.*

В этом примере локальная копия таблицы `Person` *(а точнее, часть данных таблицы* `Person`*) хранится в объекте класса* `DataTable`*. Для скачивания данных по сети применяется метод* `Fill` *адаптера.*

В среде `Visual Studio` *для отображения данных таблицы на форме используют элемент управления* `DataGridView`*, на свойство* `DataSource` *этого элемента управления привязывают имя локальной таблицы.*

4) Программа для соединения к удаленной базе данных MS SQL Server. Настройка классов SqlConnection, SqlCommand, SqlDataAdapter и т.д.

SQL Server Management Studio (**SSMS**) — утилита из Microsoft SQL Server 2005 и более поздних версий для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server. Утилита включает скриптовый редактор и графическую программу, которая работает с объектами и настройками сервера[1].

Главным инструментом SQL Server Management Studio является Object Explorer, который позволяет пользователю просматривать, извлекать объекты сервера, а также полностью ими управлять.

Также есть **SQL Server Management Studio Express** для Express версии сервера, которая является бесплатной. Однако в ней нет поддержки ряда компонентов (Analysis Services, Integration Services, Notification Services, Reporting Services) и SQL Server 2005 Mobile Edition.

Свойства классов **SqlConnection, SqlCommand, SqlDataAdapter**

SqlConnection

AccessToken - Возвращает или задает токен доступа для подключения.

CanRaiseEvents - Возвращает значение, показывающее, может ли компонент вызывать событие. (Inherited from Component)

ClientConnectionId - Идентификатор соединения последней попытки подключения, независимо от того, успешно ли выполнена попытка или завершилась ошибкой.

ColumnEncryptionKeyCacheTtl - Возвращает или задает срок жизни для записей ключей шифрования столбцов в кэше ключей шифрования столбцов для функции Always Encrypted. Значение по умолчанию — 2 часа. 0 означает отсутствие кэширования.

ColumnEncryptionQueryMetadataCacheEnabled - Возвращает или задает значение, которое указывает, включено ли кэширование метаданных запроса (True) или нет (False) для параметризованных запросов к базам данных с поддержкой функции Always Encrypted. Значение по умолчанию — true.

ColumnEncryptionTrustedMasterKeyPaths - Позволяет задать список доверенных путей к разделам для сервера базы данных. Если при обработке запроса приложения драйвер получает путь к разделу, которого нет в списке, запрос завершится ошибкой. Это свойство обеспечивает дополнительную защиту от атак на систему безопасности, которые

подразумевают предоставление скомпрометированным сервером SQL Server неверных путей к разделам, что может привести к утечке учетных данных хранилища ключей.

ConnectionString - Получает или задает строку, используемую для открытия базы данных SQL Server.

ConnectionTimeout - Получает время ожидания при попытке установки подключения, по истечении которого попытка подключения завершается и создается ошибка.

Container - Возвращает контейнер IContainer, содержащий компонент Component. (Inherited from Component)

Credential - Возвращает или задает объект SqlCredential для этого подключения.

Database - Получает имя текущей базы данных или базы данных, которая будет использоваться после открытия подключения.

DataSource - Получает имя экземпляра SQL Server, к которому осуществляется подключение.

DesignMode - Возвращает значение, указывающее, находится ли данный компонент Component в режиме конструктора в настоящее время. (Inherited from Component)

Events - Возвращает список обработчиков событий, которые прикреплены к этому объекту Component. (Inherited from Component)

FireInfoMessageEventOnUserErrors - Возвращает или задает свойство FireInfoMessageEventOnUserErrors.

PacketSize - Получает размер сетевых пакетов (в байтах), используемых при взаимодействии с экземпляром SQL Server.

ServerVersion - Получает строку, содержащую версию экземпляра SQL Server, к которому подключается клиент.

Site - Возвращает или задает ISite объекта Component. (Inherited from Component)

State - Отображает состояние SqlConnection во время последней сетевой операции, выполненной по подключению.

StatisticsEnabled - Когда задано значение true, разрешает сбор статистических сведений для текущего подключения.

WorkstationId - Получает строку, определяющую клиента базы данных.

SqlCommand

CanRaiseEvents - Возвращает значение, показывающее, может ли компонент вызывать событие. (Inherited from Component)

ColumnEncryptionSetting - Возвращает или задает параметр шифрования столбца для этой команды.

CommandText - Возвращает или задает инструкцию Transact-SQL, имя таблицы или хранимую процедуру, выполняемую для источника данных.

CommandTimeout - Возвращает или задает время ожидания перед завершением попытки выполнить команду и созданием ошибки.

CommandType - Возвращает или задает значение, определяющее, как будет интерпретироваться свойство CommandText.

Connection - Возвращает или задает объект SqlConnection, используемый этим экземпляром класса SqlCommand.

Container - Возвращает контейнер IContainer, содержащий компонент Component. (Inherited from Component)

DesignMode - Возвращает значение, указывающее, находится ли данный компонент Component в режиме конструктора в настоящее время. (Inherited from Component)

DesignTimeVisible - Возвращает или задает значение, указывающее, будет ли объект команды видимым в элементе управления Windows Forms Designer.

Events - Возвращает список обработчиков событий, которые прикреплены к этому объекту Component. (Inherited from Component)

Notification - Получает или задает значение, указывающее объект SqlNotificationRequest, связанный с данной командой.

NotificationAutoEnlist - Возвращает или задает значение, указывающее, должно ли приложение автоматически получать уведомления о запросах от общего объекта SqlDependency.

Parameters - Возвращает набор SqlParameterCollection.

Site - Возвращает или задает ISite объекта Component. (Inherited from Component)

Transaction - Возвращает или задает транзакцию SqlTransaction, в которой выполняется команда SqlCommand.

UpdatedRowSource - Возвращает или задает способ применения результатов команды к объекту DataRow при использовании метода Update объекта DbDataAdapter.

SqlDataAdapter

AcceptChangesDuringFill - Возвращает или задает значение, указывающее, вызывается ли метод AcceptChanges() в объекте DataRow после его добавления к объекту DataTable при выполнении любой из операций Fill. (Inherited from DataAdapter)

AcceptChangesDuringUpdate - Возвращает или задает, вызывается ли метод AcceptChanges() при вызове метода Update(DataSet). (Inherited from DataAdapter)

CanRaiseEvents - Возвращает значение, показывающее, может ли компонент вызывать событие. (Inherited from Component)

Container - Возвращает контейнер IContainer, содержащий компонент Component. (Inherited from Component)

ContinueUpdateOnError - Возвращает или задает значение, указывающее, следует ли генерировать ли исключение при обнаружении ошибки во время обновления строки. (Inherited from DataAdapter)

DeleteCommand - Возвращает или задает инструкцию Transact-SQL или хранимую процедуру, используемую для удаления записей из набора данных.

DesignMode - Возвращает значение, указывающее, находится ли данный компонент Component в режиме конструктора в настоящее время. (Inherited from Component)

Events - Возвращает список обработчиков событий, которые прикреплены к этому объекту Component. (Inherited from Component)

FillCommandBehavior - Возвращает или задает реакцию команды, использованной для заполнения адаптера данных. (Inherited from DbDataAdapter)

FillLoadOption - Возвращает или задает значение перечисления типа LoadOption, определяющее, как адаптер заполняет объект DataTable из объекта DbDataReader. (Inherited from DataAdapter)

IDataAdapter.TableMappings - Указывает, как исходная таблица сопоставлена с таблицей набора данных. (Inherited from DataAdapter)

InsertCommand - Возвращает или задает инструкцию Transact-SQL или хранимую процедуру, используемую для вставки новых записей в источник данных.

MissingMappingAction - Определяет действие, выполняемое, если входные данные не соответствуют таблице или столбцу. (Inherited from DataAdapter)

MissingSchemaAction - Определяет действие, которое должно быть выполнено, если существующая схема DataSet не соответствует входным данным. (Inherited from DataAdapter)

ReturnProviderSpecificTypes - Возвращает или задает, должен ли метод Fill возвращать зависящие от поставщика значения или обычные CLS-совместимые значения. (Inherited from DataAdapter)

SelectCommand - Возвращает или задает инструкцию Transact-SQL или хранимую процедуру, используемую для выбора записей из источника данных.

Site - Возвращает или задает ISite объекта Component. (Inherited from Component)

TableMappings - Получает коллекцию, обеспечивающую основное сопоставление между исходной таблицей и DataTable. (Inherited from DataAdapter)

UpdateBatchSize - Возвращает или задает число строк, обработанных при каждом обращении к серверу.

UpdateCommand - Возвращает или задает инструкцию Transact-SQL или хранимую процедуру, используемую для обновления записей в источнике данных.

```
  using System.Data;using System.Data.SqlClient;
public partial class Form1 : Form
{
   SqlConnection conn;
   SqlCommand cmd;
   SqlDataAdapter myAdapter;

   // Объекты для локальной копии БД
   DataSet myDataSet;
   DataTable Person;
   BindingSource personBS;
  }
  string connStr ="Data Source =(local)\\SQLEXPRESS;" +"Initial
Catalog = MyBase;" +"Integrated Security = True" ;( при подключении к базе данных используется те же права доступа, что и при загрузке Windows) conn = new
SqlConnection(connectionString); conn.Open();string sqlStr = "SELECT
Name, Post,Oklad FROM Person";cmd = new SqlCommand(sqlStr, conn);
myAdapter=newSqlDataAdapter(cmd);Person=newDataTable();SqlDataAdapter.
myAdapter.Fill(Person);personBS=newBindingSource();personBS.DataSource
= Person; dataGridView1.DataSource = personBS;
```

5) Место нахождения файлов базы данных MS SQL server. Системные базы данных. Подключение к базе данных, указывая путь файла базы данных.

Место нахождения файлов базы данных MS SQL server задается при ее создании.

Системная база данных	Описание
master	В этой базе данных хранятся все данные системного уровня для экземпляра SQL Server.
tempdb	Рабочее пространство для временных объектов или взаимодействия результирующих наборов.
msdb	Используется агентом SQL Server для планирования предупреждений и задач.
model	Используется в качестве шаблона для всех баз данных, создаваемых в экземпляре SQL Server. Изменение размера, параметров сортировки, модели восстановления и других параметров базы данных **model** приводит к изменению соответствующих параметров всех баз данных, создаваемых после

изменения.

resource Системные объекты физически хранятся в базе данных **resource**, но логически отображаются в схеме **sys** любой базы данных.

Строка подключения **strConn** *определяет место нахождения базы данных и параметры подключения к этой базе данных. Для определения строки подключения в окне* `Обозреватель серверов` *выделите базу данных и откройте окно свойств. В окне свойств скопируйте значение параметра* `Строка подключения` *и вставляйте эту строку в код обработчика* `Form1_Load`.

```
private void Form1_Load(object sender, EventArgs e)
{
    string strConn = "Data Source=(local)\\SQLEXPRESS; " +
                     "Initial Catalog=MyBase;" +
                     "Integrated Security=True; Pooling=False";

    string strCmd = "SELECT Name, Post, Oklad, BirthDay FROM Person";
    SqlDataAdapter adapPerson = new SqlDataAdapter(strCmd, strConn);

    DataTable Person = new DataTable();
    adapPerson.Fill(Person);

    dataGridView1.DataSource = Person;
}
```

6) Команда SELECT языка SQL. Определение вычисляемых полей. Операнд WHERE команды SELECT. Формирование условий отбора записей с помощью операций сравнения.

Команда SELECT языка SQL.

Команда SELECT предназначена для отбора нужных записей.
Команда SELECT имеет следующий упрощенный формат:

SELECT {<Список полей> }
  FROM <Список таблиц>
  WHERE <Условия отбора записей>

В этой команде обязательно должен быть указан только список полей и операнд FROM, а остальные операнды могут отсутствовать. В списке операнда FROM перечисляются имена таблиц, из которых отбираются записи. Этот список должен содержать как минимум одну таблицу. Например, для отбора всех записей таблицы следует выполнить SQL-запрос:

  SELECT Name, Oklad, Post FROM Person

Определение вычисляемых полей.

Кроме физических полей таблиц, в `SELECT`-запрос можно включать вычисляемые поля. Для получения вычисляемого поля в списке полей указывается не имя этого поля, а выражение, по которому рассчитывается его значение. Например,

```
SELECT Name, Post, Oklad, Oklad*1.5  AS Зарплата
       FROM   Person
```

Здесь через ключевое слово **AS** вводится имя вычисляемого поля. Ввод имени поля не обязателен, но использование имени поля позволяет представлять информацию более наглядно.

Операнд WHERE команды SELECT. Формирование условий отбора записей с помощью операций сравнения.

На практике требуется отобрать записей, которые удовлетворяют каким-либо условиям отбора записей. Это условие отбора записей задается с помощью операнда **WHERE**.

Критерий отбора записей представляет собой логическое выражение, в котором можно использовать следующих операторов сравнения.

= - равно; *<= - меньше или равно;*

> - больше; *<> - или != - не равно;*

< - меньше; *!> - не больше;*

>= - больше или равно; *!< - не меньше.*

В качестве примера можно привести следующий SQL-запрос.

```
SELECT Name, Oklad  FROM  Person
       WHERE   Oklad  > 20000;
```

Эта команда определяет получение списка сотрудников, имеющих оклад более 20000 рублей.

```
private void sql_load2_Click(object sender, EventArgs e)
    {
        /*string str = "SELECT Id, Name, Age, Position, Experience, Salary FROM Person
Where Salary between 37000 and 50000";
        com.CommandText = str;
        // Скачивание данных таблицы
        Person.Clear();
        myAdapter.Fill(Person);*/
    }
```

7) Соединение с базой данных MS Access. Провайдеры и классы для доступа к базе данных MS Access. Программный доступ к базе данных Access.

Вначале запустите **Access** и создайте базу данных с именем **AccessDB**. В этой базе данных создайте таблицу **Person** и заполняйте эту таблицу с несколькими записями. Пользовательский интерфейс MS Access достаточно прост, как выполнить создание таблиц и заполнение их данными вы наверно, догадаетесь без проблем.

Теперь переходите в среду Visual Studio.NET и подключитесь на эту базу данных. Для этого выполните команду **Сервис/Подключиться к базе данных**. При этом появляется окно выбора источника данных. В этом окне выберите строку **Файл базы данных Microsoft Access** и нажмите на кнопку **Продолжить.** Тогда появится окно подключения к базе данных, где следует выбрать имя файла базы данных
С помощью кнопки **Обзор** выберите файл базы данных, например, D:\AccessDB.mdb и нажмите на кнопку. **OK**. Тогда в окне Обозреватель серверов появится выбранная база данных.

Наборы классов для работы с базами данных зависят от типа используемой СУБД. В случае использования базы данных Access эти классы находятся в пространстве имен System.Data.OleDb. Подключите в приложение это пространство имен:

```
using System.Data.OleDbClient;
```

Имена классов для доступа к базам данных различных СУБД похоже друг другу и зависят от поставщика данных. Например, класс для соединения базе данных MS SQL Server называется SqlConnection, а аналогичный класс для соединения базе данных

Access называется OleDbConnection. Как видно, имена классов для работы с базой данных Access различаются от имен классов для работы с базой данных MS SQL Server только приставкой "OleDb" и "Sql".

Заметим, что по функциональности эти классы имеют одинаковые возможности. Для примера, ниже приведен порядок использования классов для работы с базой данных Access. Вначале объявляйте глобальных переменных - объектов классов для работы с базой данных Access:

```
public partial class Form1 : Form
{
   OleDbConnection conn;
   OleDbCommand cmd;

   OleDbDataAdapter adapPerson;
   DataTable Person;
```

Теперь переопределите код обработчика события Form1_Load. Для получения строки подключения в окне Обозреватель серверов вначале выделяйте базу данных AccessDB, затем из окна свойств скопируйте строку подключения.

```
string strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
                 "Data Source=E:\\AccessDB.mdb";

   conn = new OleDbConnection(strConn);
   conn.Open();
```

8) Операнд WHERE команды SELECT. Формирование условий отбора записей с помощью операторов LIKE, IN, BETWEEN, NOT, IS NULL.

Для составления более сложных условий отбора с помощью операнда **WHERE** можно использовать следующих операторов:

1. Оператор LIKE.

Этот оператор позволяет формировать условие отбора записей по шаблону. В строке шаблона можно использовать один или несколько символов подстановки. Назначение этих символов представлено ниже:

_ - замещение одного символа. Например, **J_y** соответствует **Joy** и **Jay**.

% - замещение любого количества символов, в том числе и нулевого. Например, **Иван%** соответствует фамилиям Иван**,** Иванов**,** Иванчук и т.п.

[символы] – соответствует одному любому символу перечисленных в скобках. Например, **[sm]ay** соответствует say и may.

[^символы] – соответствует одному любому символу, не включенному в скобки. Например, **[^a]** соответствует любому символу кроме **a**.

[Символ1 – Символ2] – соответствует диапазону символов. Например, **[a-c]bc** соответствует словам abc, bbc и cbc.

- соответствует одной любой цифре. Например, A# соответствует от A0 до A9.

В качестве примера, в коде предыдущей программы замените SQL-запрос на следующий запрос:

```
SELECT Name, Post, Oklad    FROM  Person
       WHERE   Name LIKE 'Иван%'
```

В результате выполнения этого SQL-запроса будет получен список служащих, фамилии которых начинаются на *Иван*.

2. Оператор IN

Чтобы извлечь записи, значения полей которых содержатся в некотором заданном списке, в условии WHERE можно использовать оператор **IN**. Например, следующий SELECT-запрос использует оператор IN для получения записей таблицы Person, у которых ID равен 1, 3 и 15.

```
SELECT    ID, Name, Post, Oklad
           FROM  Person
          "WHERE   ID   IN   (1, 3, 15)";
```

3. Оператор BETWEEN

Оператор **BETWEEN** в условии WHERE можно использовать для извлечения записей, значения полей которых находятся в указанном диапазоне. Например, следующий запрос извлекает записей о служащих, у которых **DepID** находится между 2 и 4:

```
SELECT   Name, Oklad, DepID
         FROM   Person
         WHERE  DepID  BETWEEN 2 AND 4
```

4. Использование оператора NOT

Оператор **NOT** с другим оператором в условии WHERE можно использовать для инвертирования значения оператора. Например, следующий SELECT-запрос использует оператор NOT для инвертирования значения оператора BETWEEN в примере, который был показан ранее:

```
SELECT   Name, DepID,  Oklad, DepID
         FROM   Person
         WHERE DepID NOT BETWEEN 2 AND 4
```

5. Оператор IS NULL

Поля могут иметь значение NULL. Значение NULL отличается от пустой строки или нуля – это значение, которое либо еще не задано, либо неизвестно. Можно использовать оператор IS NULL в условии WHERE для определения, равно ли значение поля NULL.

Например, следующий SELECT-запрос использует оператор IS NULL для получения записей о служащих, у которых поле Oklad содержит значение NULL:

```
SELECT  Name, Post  FROM  Person
       WHERE   Oklad  IS  NULL;
```

string str = "SELECT Id, Name, Age, Position, Experience, Salary FROM Person Where Experience > 4 and Salary between 37000 and 50000";

com.CommandText = str;

// Скачивание данных таблицы

Person.Clear();
myAdapter.Fill(Person);*/

9) Использование свойства Filter класса BindingSource для фильтрации записей таблицы. Сортировка записей операндом ORDER BY и свойством Sort класса BindingSource.

Использование свойства Filter класса BindingSource для фильтрации записей таблицы *:*

Для хранения имени поля для фильтрации определите глобальную переменную `filterField`. Предполагается, что по умолчанию фильтрация происходит по полю **`Name`**.

```
string filterField = "Name";
```

Для текстового поля определите обработчик события **`TextChanged`**, где определяется условие фильтрации:

```
private void textBox1_TextChanged(...)
{
   bsPerson.Filter = filterField + " LIKE '" + textBox1.Text + "%'";
}
```

Как видно, при изменении текста в текстовом поле формируется новое условие фильтрации. Например, когда фильтрация происходит по полю `Name`, то условие фильтрации может быть следующего вида:

```
Name LIKE    'Иван%'
```

ИЛИ

Задать Filter свойства необходимое выражение.

В примере выражение является именем столбца, за которым следует значение для столбца.

```
BindingSource1.Filter = "ContactTitle='Owner'";
```

Сортировка записей операндом ORDER BY

Оператор **ORDER BY** позволяет отсортировать извлекаемые значения по определенному столбцу:

SELECT * FROM <имя таблицы> ORDER BY <название столбца >

Сортировка записей свойством Sort класса BindingSource:

1. Задайте Sort свойства к имени столбца, который будет следуют ASC или DESC для указания по возрастанию или убыванию.
2. Несколько столбцов разделяются запятыми.

```
BindingSource1.Sort = "Country DESC, Address ASC";
```

10) Перемещение по записям таблицы с помощью свойств и методов класса BindingSource.

Свойство **Position** класса BindingSource содержит номер текущей записи таблицы, которое доступно для чтения и записи. Изменив значение этого свойства можно перемещаться на нужную запись.

Для перемещения по записям таблицы класс **BindingSource** имеет следующие методы:

MoveNext() – переход на следующую запись;
MovePrevious() – переход на предыдущую запись;
MoveFirst() – переход на первую запись;
MoveLast() – переход на последнюю запись.

```
private void button1_Click(...)
{
  // Переход на следующую запись
  personBS.MoveNext();
  int curRecord = personBS.Position;
  MessageBox.Show(
       Convert.ToString(curRecord));
}
private void button2_Click(...)
{
   // Переход на запись с номером
   int curRecord =
          Convert.ToInt32(textBox1.Text);
   personBS.Position = curRecord;
}
```

11) Доступ к полям текущей записи с помощью класса BindingSource

Для доступа к полям текущей записи также можно использовать свойство **Current** класса BindingSource. Это свойство дает указатель на текущую запись таблицы, который затем может использоваться для доступа к полям этой записи. Для демонстрации работы с этим свойством на форме разместите два текстовых поля и определите обработчик события **CellClick** элемента управления dataGridView1.

```
private void dataGridView1_CellClick(...)
{
   DataRowView dataRow =(DataRowView)personBS.Current;
   textBox1.Text = dataRow["Name"].ToString();
   textBox2.Text = dataRow["Oklad"].ToString();
}
```

Как видно, вначале с помощью свойства Current берется ссылка на текущую запись, а затем эта ссылка преобразуется на тип **DataRowView**. Этот тип представляет собой запись таблицы в объекте DataGridView.

Далее, для обращения к полям текущей записи используются составные имена полей:

```
dataRow["Name"]
dataRow["Oklad"]
```

***Замечание**. Свойство **Current** предназначено только для чтения. Поэтому с помощью этого свойства нельзя изменить значения поля текущей записи.*

12) Доступ к полям текущей записи с помощью DataGridView

При отображении таблицы на `DataGridView` только одна запись таблицы является доступной (*текущая запись*). Имеется различные способы считывания значений полей текущей записи и изменения их. Ниже для этой цели использованы следующие свойства элемента управления `DataGridView`:

`CurrentRow` – содержит указатель на текущую запись;

`CurrentCell` - содержит указатель на выделенную ячейку текущей записи.

Для демонстрации работы с этими полями измените форму предыдущего приложения, как это показано на рисунке 1.16.

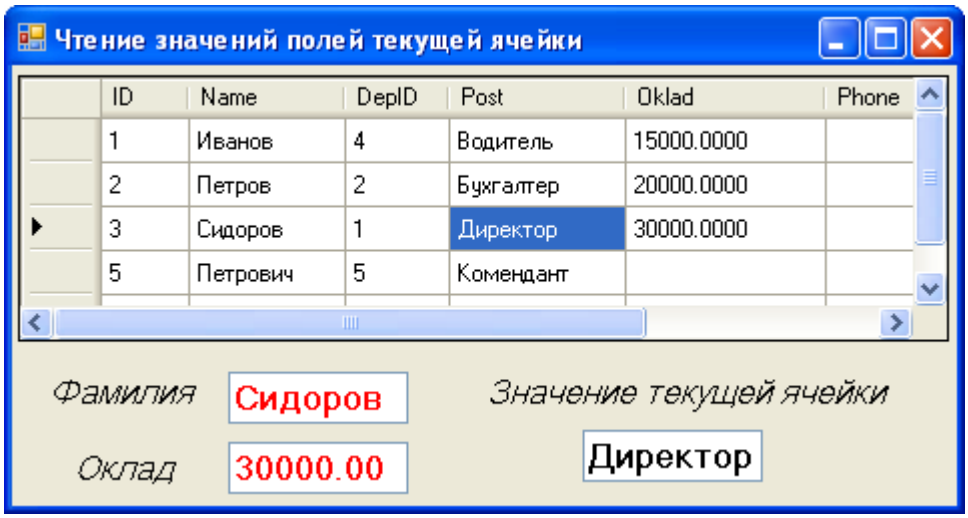


***Рис. 1.16.** Отображение полей текущей записи*

При выборе записи в `dataGridView1` в текстовых полях `textBox1` и `textBox2` должны появиться фамилия служащего и его оклад, а значение выбранной ячейки должно отображаться в текстовом поле `textBox3`. Для реализации этой функциональности напишите обработчик события **`CellClick`** для элемента управления `dataGridView1`.

```
private void dataGridView1_CellClick(...)
{
    // Выбор текущей записи
    DataGridViewRow curRow = dataGridView1.CurrentRow;

    textBox1.Text = curRow.Cells["Post"].Value.ToString();
    textBox2.Text = curRow.Cells["Oklad"].Value.ToString();

    // Выбор текущей ячейки
    DataGridViewCell curCell = dataGridView1.CurrentCell;
    textBox3.Text = curCell.Value.ToString();
}
```

В этом примере для обращения к ячейкам текущей записи используется свойство **`Cells`**, которое представляет собой коллекцию полей. Для обращения к конкретному полю следует использовать название поля или индекс поля. Например,

`curRow.Cells["Oklad"].Value` – обращению по названию поля

`curRow.Cells[4].Value` – обращению по индексу поля.

Индексы полей нумеруются начиная с нуля, в зависимости от того, как эти поля перечислены в SQL-запросе. Например, пусть определен `SELECT`-запрос следующего вида:

```
SELECT  * FROM  Person
```

Здесь символ **`"*"`** означает выбор всех полей таблицы. В этом случае индексы полей таблицы `Person` имеют следующие значения, которые представлены в таблице 1.2.

Таблица 1.2*.Индексы полей таблицы* `Person`

Имя поля	***Индекс***
ID	*0*
Name	*1*
DepID	*2*
Post	*3*
Oklad	*4*

Как видно из нумерации полей таблицы `Person`, выражение `curRow.Cells[4]` дает ссылку на поле `Oklad`.

13) Доступ к записям таблицы с помощью класса DataTable

Объект класса `DataTable` предназначен для сохранения локальной копии таблицы. Этот класс имеет свойство **`Rows`**, который представляет собой коллекцию записей таблицы. Это свойство удобно использовать в том случае, когда необходимо перемещаться по всем записям таблицы.

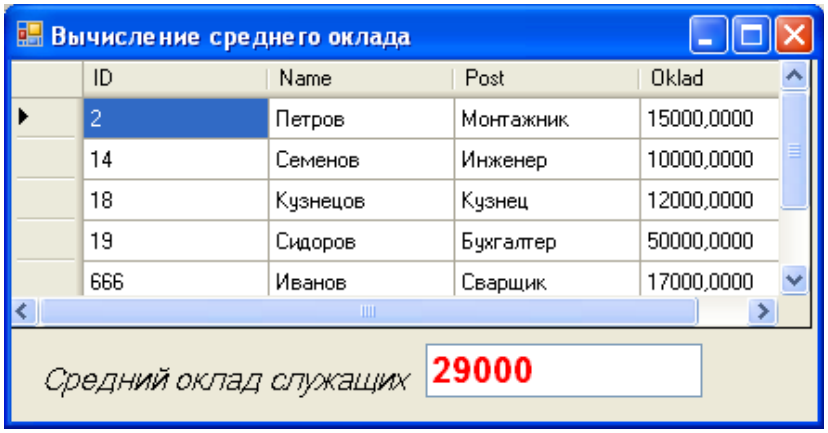


Рис. 1.17. *Вычисление среднего оклада служащих*

Как видно, здесь оставлено только одно текстовое поле для вывода среднего оклада служащих. Код вычисления среднего оклада можно реализовать в обработчике события **`Form_Load`**. Для этого в конец этого обработчика добавляйте следующий код:

```
double Sum = 0, Sr;
foreach (DataRow person in Person.Rows)
{
    Sum = Sum + Convert.ToDouble(person["Oklad"]);
}

Sr = Sum / Person.Rows.Count;
textBox1.Text = Convert.ToString(Sr);
```

Как видно, внутри тела оператора **foreach** вычисляется сумма окладов. Текущая запись набора данных присваивается объекту класса DataRow. К полю текущей записи можно обращаться, по названию поля:

```
person["Oklad"]
```

14) Команды языка SQL для редактирования, добавления и удаления записей. Общий формат команд UPDATE, INSERT и DELETE.

UPDATE – изменение существующей записи;

```
UPDATE имя_таблицы
SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN
[FROM выборка AS псевдоним_выборки]
    UPDATE  Person SET  Oklad = Oklad + 2000
            WHERE  Oklad < 15000
[WHERE условие_обновления]
```

INSERT – вставка новой записи;

```
INSERT [INTO] имя_таблицы [(список_столбцов)] VALUES (значение1, значение2, ... значениеN)
```

```
SqlCommand command = new SqlCommand("INSERT INTO [PriceList] (Product, Brends_Id, Id_Product, Price, Quantity)VALUES(@Product, @Brends_Id, @Id_Product, @Price, @Quantity)", formForAdmin.conn);
            command.Parameters.AddWithValue("Product", textBox1.Text);
            command.Parameters.AddWithValue("Brends_Id", textBox2.Text);
            command.Parameters.AddWithValue("Id_Product", textBox3.Text);
            command.Parameters.AddWithValue("Price", textBox4.Text);
            command.Parameters.AddWithValue("Quantity", textBox5.Text);
            command.ExecuteNonQuery();
            // Вывод данных обновленной таблицы
            formForAdmin.PriceLists.Clear();
            formForAdmin.adapPriceLists.Fill(formForAdmin.PriceLists);
```

DELETE – удаление записи.

```
DELETE [FROM] имя_таблицы  WHERE условие_удаления
//удаление данных по вводу их айди
            SqlCommand command = new SqlCommand("DELETE FROM [PriceList] WHERE [Id]=@Id", conn);
            command.Parameters.AddWithValue("Id", textBox1.Text);
            command.ExecuteNonQuery();
            PriceLists.Clear();
            adapPriceLists.Fill(PriceLists);
```

15) Использование команды UPDATE для изменения должности и оклада служащего.

Для изменения значения полей группы записи на языке SQL имеется команда **UPDATE**. Синтаксис использования этой команды ниже рассматривается в различных

примерах. Например, для изменения оклада группы служащих, у которых оклад менее 15000 рублей можно выполнить следующую команду:

```
UPDATE  Person SET  Oklad = Oklad + 2000
        WHERE  Oklad < 15000
```

Пример использования этой команды при создании приложения базы данных.

Напишите обработчик события Click для кнопки **Изменить данные служащего**. При нажатии на эту кнопку должна измениться значения полей Post и Oklad таблицы Person.

```
private void button1_Click(...)
{
   string Post = "'" + comboBox1.Text + "', ";
   string Oklad = textBox3.Text;
   string curID =  textBox1.Text;

   string updateStr = "UPDATE Person SET " +
                      " Post = " + Post +
                      " Oklad = " + Oklad +
                      " WHERE  ID = " + curID;

   SqlCommand updateCmd = new SqlCommand(updateStr, conn);
   updateCmd.ExecuteNonQuery();

   // Вывод данных обновленной таблицы
   Person.Clear();
   adapPerson.Fill(Person);
}
```

16) Использование параметрического SQL-запроса для изменения должности и оклада служащего.

При решении практических задач одни и те же SQL-запросы отличаются друг от друга только некоторыми параметрами. Например, в предыдущем примере, при удалении записи по номеру ID, удаляемая запись отличается от другой удаленной записи только по номеру ID. Поэтому появляется необходимость создания параметрических SQL-запросов. Параметр SQL-запроса выделяется префиксом "@". Например,

```
DELETE FROM  Person WHERE  ID = @ID
```

В этом SQL-запросе **@ID** означает параметр SQL-запроса. Заметим, что формирование SQL-запроса в виде символьной строки является опасным способом работы с данными с точки зрения хакерских атак, так как эта строка содержит некоторую информацию о данных таблицы. Поэтому запросы следует формировать в виде параметрических SQL-запросов, которые компилируются в стадии создания исполнимого файла.

Пример использования при создании приложения базы данных.

При нажатии на эту кнопку должны измениться значения полей Post и Oklad таблицы Person. Параметр SQL-запроса выделяется префиксом "@".

```
private void button1_Click(...)
{
   string ID = textBox1.Text;
```

```
    string Oklad = textBox3.Text;
    string Post = comboBox1.Text;

    string updateStr = "UPDATE Person SET  " +
                           "Post = @Post,  " +
                           "Oklad = @Oklad  " +
                           "WHERE  ID = @ID ";
    SqlCommand updateCmd = new SqlCommand(updateStr, conn);
    //Добавление параметров в коллекцию Parameters
    updateCmd.Parameters.Add("@Post",SqlDbType.NChar, 10);
    updateCmd.Parameters.Add("@Oklad", SqlDbType.Money);
    updateCmd.Parameters.Add("@ID",SqlDbType.Int);
    // Определение значений параметров
    updateCmd.Parameters["@Post"].Value = Post;
    updateCmd.Parameters["@Oklad"].Value = Oklad;
    updateCmd.Parameters["@ID"].Value = ID;
    updateCmd.ExecuteNonQuery();
    // Вывод данных обновленной таблицы
    Person.Clear();
    adapPerson.Fill(Person);
}
```

17) Использование команды INSERT для добавления новой записи в таблицу Person.

Для добавления нового служащего можно выполнить следующую команду:

```
INSERT INTO Person (Name, Oklad, Post)
            VALUES("Иванов", 40000, "Директор");
```

Пример. Для добавления новой записи в удаленную базу данных определите следующий обработчик события для кнопки **Добавить нового служащего**:

```
private void button2_Click(...)
{
    string ID = textBox1.Text + ", ";
    string Name = "'" + textBox2.Text + "', ";
    string Post = "'" + comboBox1.Text + "', ";
    string Oklad = textBox3.Text;

    string insertStr = "INSERT INTO Person "+
                           "(ID, Name, Post, Oklad) " +
                       "VALUES (" + ID + Name + Post + Oklad + ")";

    SqlCommand insertCmd = new SqlCommand(insertStr, conn);
    insertCmd.ExecuteNonQuery();

    // Вывод данных обновленной таблицы
    Person.Clear();
    adapPerson.Fill(Person);
}
```

Как видно, вначале формируется строка SQL-запроса. В SQL-запросе значения полей ID, Name, Post и Oklad разделены запятыми, эти запятые определяются в одноименных переменных строкового типа. Значения строковых полей (Name и Post) должны быть взяты в одинарные кавычки. Эти одинарные кавычки также определены в строковых переменных.

18) Использование параметрического SQL-запроса для добавления новой записи в таблицу Person

```
private void button2_Click(...)
```

```
{
    string ID = textBox1.Text;
    string Name = textBox2.Text;
    string Oklad = textBox3.Text;
    string Post = comboBox1.Text;

    string insertStr = "INSERT INTO Person (ID, Name, Post, Oklad)" +
                              " VALUES (@ID, @Name, @Post, @Oklad)";

    SqlCommand insertCmd = new SqlCommand(insertStr, conn);

    //Добавление параметров в коллекцию Parameters
    insertCmd.Parameters.Add("@ID", SqlDbType.Int);
    insertCmd.Parameters.Add("@Name", SqlDbType.NChar, 10);
    insertCmd.Parameters.Add("@Post", SqlDbType.NChar, 10);
    insertCmd.Parameters.Add("@Oklad", SqlDbType.Float);

    // Определение значений параметров
    insertCmd.Parameters["@ID"].Value = ID;
    insertCmd.Parameters["@Name"].Value = Name;
    insertCmd.Parameters["@Post"].Value = Post;
    insertCmd.Parameters["@Oklad"].Value = Oklad;

    insertCmd.ExecuteNonQuery();

    // Вывод данных обновленной таблицы
    Person.Clear();
    adapPerson.Fill(Person);
```

Комментарии кода этих процедур.

1. *Как видно из кода, каждый параметр должен быть добавлен в коллекцию параметров* ***Parameters****. Для этого используется метод* ***Add****. При вызове этого метода указываются имя и тип параметра, и если параметр строкового типа, то указывается еще ширина поля.*

2. *Далее, с помощью свойства* ***Value*** *присваивается значение параметра. При многократном использовании параметрического SQL-запроса, следует определить только значения параметров, что значительно упрощает формирование SQL-запроса.*

19) Использование новой формы для заполнения данных новой записи

Для пункта 19 и 20

==========

Шаг 1. Вначале командой **Project/Add Windows Form** добавляйте в проект новую форму `Form2`. На этой форме разместите текстовых полей для ввода или редактирования значений полей `ID`, `Name` и `Oklad` (Рис. 1.22). Для выбора должности служащего на форме разместите раскрывающиеся список `comboBox1` и заполните свойство `Items` со списком должностей. Для сохранения введенных данных в таблице `Person` удаленной базы данных разместите кнопку **Сохранить**.

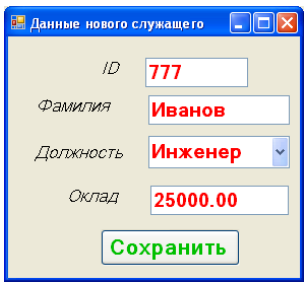


Рис. 1.22. *Вспомогательная форма приложения*

В коде `Form2` приходится обращаться к объектам, размещенным на форме `Form1`. Для обращения к этим объектам вначале следует получить ссылку на форму `Form1`. Для сохранения этой ссылки в коде класса `Form2` объявляйте глобальную переменную:

```
public partial class Form2 : Form
{
   public Form1 form1;
   . . . . . .
```

В коде формы `Form1` приходится использовать текстовых полей и раскрывающиеся список `comboBox1`, размещенных на форме `Form2`. Поэтому этих объектов следует объявлять с модификатором доступа `public`. Объявления этих объектов находятся в файле **`Form2.Designer.cs`**. С помощью окна `Solution Explorer` откройте этот файл и измените модификаторов доступа.

```
public System.Windows.Forms.TextBox textBox1;
public System.Windows.Forms.TextBox textBox2;
public System.Windows.Forms.TextBox textBox3;
public System.Windows.Forms.ComboBox comboBox1;
```

Шаг 2. Теперь из формы `Form1` удалите текстовых полей и раскрывающиеся список **`comboBox1`**. Таким образом, на форме остается только две кнопки **Добавить служащего** и **Редактировать** (Рис. 1.23).

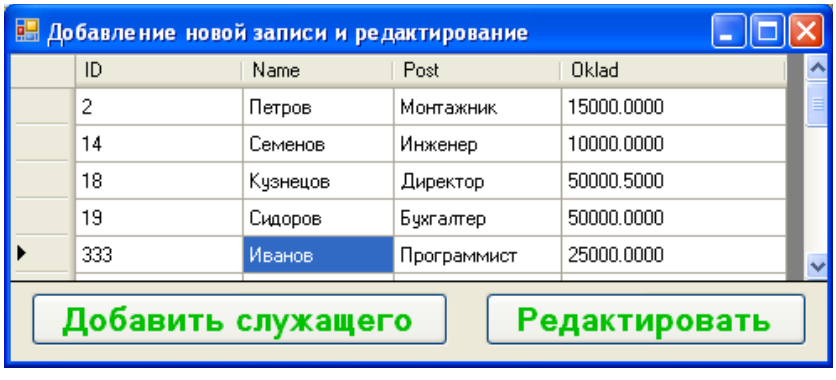


Рис. 1.23. *Главная форма приложения*

Для того, чтобы переменные `Form1` были доступны в коде формы `Form2`, переменных `Form1` следует объявлять с модификатором доступа `public`:

```
public partial class Form1 : Form
{
  public SqlConnection conn;
  public SqlDataAdapter adapPerson;
  public DataTable Person;
```

. . . . . .

Вспомогательная форма Form2 предназначена для добавления нового служащего. Для хранения режима перехода на эту форму в классе **Form1** объявляйте глобальную переменную:

```
public int Option;
```

При нажатии на одну из кнопок главной формы Form1 вы должны создавать объект класса Form2 и передавать ссылку на главную форму. С учетом сказанного, обработчик события кнопки **Добавить служащего** имеет следующий вид:

```
private void button1_Click(. . .)
{
    // Код кнопки Добавить служащего
    Option = 1;
    Form2 form2 = new Form2();
    form2.form1 = this;

    form2.Show();
}
```

20) Использование новой формы для редактирования записи.**(см. п. 19)**

Вспомогательная форма Form2 предназначена для редактирования существующей записи. Для хранения режима перехода на эту форму в классе **Form1** объявляйте глобальную переменную:

```
public int Option;
```

При нажатии на кнопку **Редактировать** значения полей ID, Name**,** Post и Oklad текущей записи должны отображаться на соответствующих полях формы Form2. Ниже приведен код обработчика события этой кнопки:

```
private void button2_Click(...)
{
  // Код кнопки Редактировать
  Option = 2;

  Form2 form2 = new Form2();
  form2.form1 = this;

  DataGridViewRow curRow = dataGridView1.CurrentRow;
  form2.textBox1.Text = curRow.Cells["ID"].Value.ToString();
  form2.textBox2.Text = curRow.Cells["Name"].Value.ToString();
  form2.textBox3.Text = curRow.Cells["Oklad"].Value.ToString();
  form2.comboBox1.Text =curRow.Cells["Post"].Value.ToString();

  form2.Show();
}
```

21) Проектирование структуры базы данных. Правила первой, второй и третьей нормальной формы. Типы отношений между записями связанных таблиц.

Проектирование базы данных, как правило, играет одну из ключевых ролей в большинстве проектов. Грамотно спроектированная база позволяет без ошибок вносить изменения данных, быстро найти нужную информацию. Проектирование базы данных

сводится к разделению базы данных на несколько взаимосвязанных таблиц, которые удовлетворяют **правилам нормальных форм** *(условиям)*. Ниже на примере базы данных `MyBase` рассматривается правила нормальных форм и способы удовлетворения этих требований.

Первая нормальная форма

Первая и главная нормальная форма требует от таблицы (а точнее, от ее проектировщика) следования следующим правилам:

1. *Каждый столбец в строке должен быть атомарным.* Примером, который не удовлетворяет этому требованию, является поле `Name` таблицы `Person`, если в это поле вводить фамилию, имя и отчество служащего. Чтобы эти данные были атомарными для фамилии, имени и отчеству следует определить отдельные поля.

2. *Каждая строка в таблице обязана содержать одинаковое количество столбцов.* Если данные определены в виде таблицы, то это условие выполняется автоматически, так как таблица всегда имеет одинаковое число столбцов для всех записей таблицы.

3. *Все строки в таблице должны быть уникальны.* Например, может быть два служащих однофамильцев *Ивановых*, которые имеют одинаковые должности, даты рождения, номера телефона и т.д. Для ввода различия следует ввести идентификатор служащего (поле **`ID`**).

Таким образом, условия первой нормальной формы выполняются очень легко. Заметим, что каждая последующая нормальная форма предполагает выполнение правил предыдущей нормальной формы, поэтому при изучении последующих нормальных форм, предположим, что это условие выполняется.

Вторая нормальная форма

Условия второй нормальной формы представляет собой следующие два условия, которые исключают избыточных данных.

1. *Для повторяющихся значений поля в различных записях следует создать отдельную таблицу*
2. *Связать эти таблицы с помощью внешнего ключа.*

Третья нормальная форма.

Третья нормальная форма требует от таблицы базы данных выполнения следующего правила: *база данных будет находиться в третьей нормальной форме, если каждый не ключевой столбец независим друг от друга.*

Типы отношений между записями связанных таблиц.

• связь один к одному;

При установлении связи "один к одному" каждой строке таблицы А может соответствовать только одна строка таблицы Б и наоборот. Связь "один к одному"

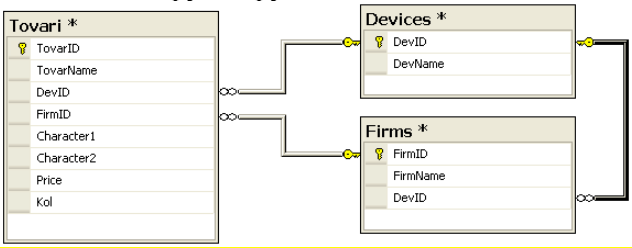
создается в том случае, когда оба связанные столбца являются первичными ключами или на них наложены ограничения уникальности.

• *связь один ко многим;*
Является наиболее часто используемым типом связи. В такой связи каждой записи в таблице А (первичный ключ) соответствует несколько записей в таблице В (внешний ключ), а запись в таблице В не может иметь более одной соответствующей ей записи в таблице А. Таблица А называется главной ,а таблица В подчиненной таблицей. Или иногда таблицу А называют родительской таблицей, а В – дочерней .

• *связь многие ко многим.*
При установлении связи "многие ко многим" каждой строке таблицы А может соответствовать множество строк таблицы Б и наоборот. Такая связь создается при помощи третьей таблицы, называемой соединительной, первичный ключ которой состоит из внешних ключей, связанных с таблицами А и Б.

Создаём таблицу Devices(процессор, материнская плата, ОЗУ, дисплей и т.д.), Firms(название фирмы, адрес, телефон, название руководителя, адрес сайта и т.п.), Tovari(имя устройства, код типа устройства и фирмы-производителя, характеристику устройства (поле **Character1**), цену устройства и количества устройств, которые имеются в складе). Эти таблицы взаимосвязаны с помощью полей `DevID` и `FirmID`. Для определения этих связей в окне `Server Explorer` выберите узел `Database Diagrams` и выполните команду **`Add New Diagram`**. При этом появляется окно со списком таблиц базы данных.

Для добавления таблицу в диаграмму следует выбрать имя таблицы и нажать на кнопку **`Add`**. Таким образом, выберите все три таблицы. Тогда на рабочей области появятся три прямоугольные области, которые содержат структуру таблиц `Devices`, `Firms` и `Tovari`. Заметьте, что таблицы `Devices` и `Firms` связаны с таблицей `Tovari`. Кроме этого, таблицы `Devices` и `Firms` взаимосвязаны друг с другом



22) Вывод записей по категориям с помощью ComboBox.

Для вывода товаров по категориям, на практике обычно применяют раскрывающиеся список **`ComboBox`**. Ниже показано использование этого элемента для вывода товаров по категориям. Создайте новое приложение и разместите на форме элементов управления `DataGridView` и `ComboBox` (Рис. 1.28).

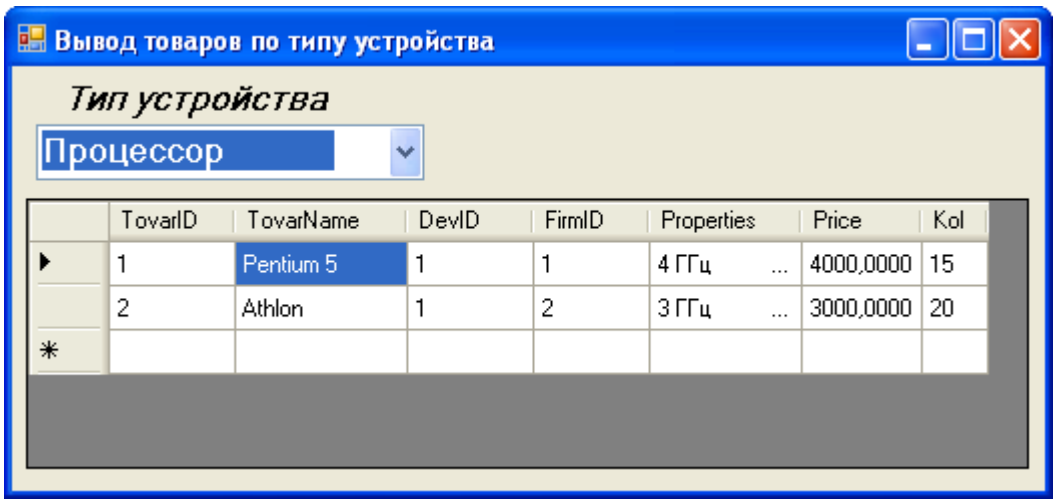


Рис. 1.28. *Вывод записей по типу устройства*

В коде этого приложения будет использоваться объекты классов `SqlCommand`, `SqlDataAdapter` и `DataTable` для всех трех таблиц базы данных `Computer`. Эти объекты будут использоваться в разных обработчиках событий. Поэтому их следует объявлять как глобальных переменных класса `Form1`:

```
public partial class Form1 : Form
{
    string conStr;
    SqlConnection conn;

    string strTovari, strDevices, strFirms;
    SqlCommand cmdTovars, cmdDevices, cmdFirms;

    SqlDataAdapter adapTovars, adapDevices, adapFirms;

    DataTable Devices, Firms,Tovars;
    BindingSource TovariBS;

    int index1, index2;
    string DevID, FirmID;

    DataRow[] curRow;
    ..........................
```

Раскрывающиеся список **`comboBox1`** должен заполняться со значениями поля `DevName` таблицы `Devices`. При запуске приложения в этом списке должна быть выбрана строка “`Все типы устройств`”, а в элементе управления `dataGridView1` должны отображаться все записи таблицы `Tovars`. Все эти работы следует выполнить в обработчике события `Form1_Load`:

```
private void Form1_Load(...)
{
    // Соединение с базой данных Computer
    conStr = "Data Source=(local)\\SQLEXPRESS;" +
             "Initial Catalog = Computer;" +
             "Integrated Security = True";
    conn = new SqlConnection(conStr);
    conn.Open();

    // Вывод всех записей таблицы Tovars
```

```
        strTovars = "SELECT *  FROM   Tovars";
        cmdTovars = new SqlCommand(strTovars, conn);

        // Скачивание данных в локальную таблицу
        adapTovars = new SqlDataAdapter(cmdTovars);
        Tovars = new DataTable();
        adapTovars.Fill(Tovars);

        // Привязка источника данных на dataGridView1
        bsTovars = new BindingSource();
        bsTovars.DataSource = Tovars;
        dataGridView1.DataSource = bsTovars;

        // ТАБЛИЦА   Devices
        strDevices = "SELECT  * FROM  Devices";
        cmdDevices = new SqlCommand(strDevices, conn);
        adapDevices = new SqlDataAdapter(cmdDevices);

        Devices = new DataTable();
        adapDevices.Fill(Devices);

        // Заполнение ComboBox
        comboBox1.Items.Add("Все типы устройств");
        foreach (DataRow dev in Devices.Rows)
        {
           comboBox1.Items.Add(dev["DevName"]);
        }
        comboBox1.SelectedIndex = 0;
    }
```

При выборе из списка comboBox1 типа устройства, в элементе управления dataGridView1 должны выводиться только те товары, которые относятся к выбранному типу устройства. Для реализации этой функциональности напишите обработчик события SelectedIndexChanged для comboBox1.

```
private void comboBox1_SelectedIndexChanged(...)
{
   strTovars = "SELECT * FROM   Tovars ";
   index1 = comboBox1.SelectedIndex;

   // Если выбран конкретный тип устройства
   if (index1 > 0)
   {
      curRow = Devices.Select("DevName = '" + comboBox1.Text + "'");
      DevID = curRow[0]["DevID"].ToString();

      strTovars = strTovars + " WHERE  DevID = " + DevID;
   }

   cmdTovars.CommandText = strTovars;
   Tovars.Clear();
   adapTovars.Fill(Tovars);
}
```

В раскрывающемся списке comboBox1 пользователь выбирает название типа устройства - DevName. Для поиска записи в таблице Devices по типу устройства применяется метод **Select** класса DataTable. Этот метод возвращает набор записей, удовлетворяющих условию поиска.

23) Перемещение записей из одной таблицы в другую (например, в Корзину).

В компьютерном салоне клиент выбирает компонентов компьютера, и тем самым определяет конфигурацию своего компьютера. Обычно выбранные товары следует переместить в таблицу-корзину.

Для организации этой работы добавьте в базу данных новую таблицу `Korzina`. Эта таблица должна содержать основных характеристик товаров, поэтому для него определите следующих полей: `TovarID`, `TovarName`, `Price` и `Kol`. Чтобы отображать данных таблицы `Korzina` на форме, разместите элемент управления `dataGridView2`. Также разместите два текстовых поля для ввода количества товаров и для отображения стоимости набранных товаров (Рис. 1.30).

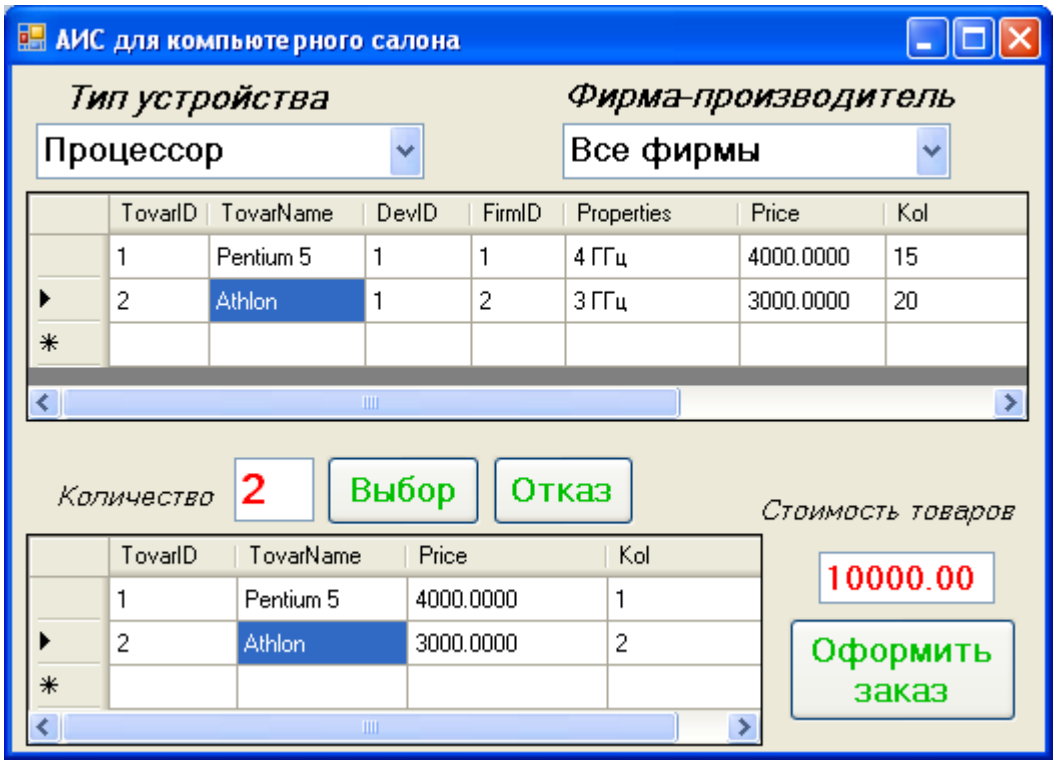


Рис. 1.30. *Добавление товаров в Корзину*

Для работы с таблицей `Korzina` объявите глобальных переменных.

```
public partial class Form1 : Form
{
  string strKorzina;
  SqlCommand cmdKorzina;

  SqlDataAdapter adapKorzina;
  DataTable Korzina;

  BindingSource bsKorzina;
  . . . . . .
```

В обработчике события `Form_Load` добавьте код для вывода таблицы `Korzina` в элементе управления `dataGridView2`.

```
private void Form1_Load(...)
{
    . . . . . .
     strKorzina = "SELECT *  FROM   Korzina ";
```

```
            cmdKorzina = new SqlCommand(strKorzina, conn);

            // Копирование данных в локальную таблицу
            adapKorzina = new SqlDataAdapter(cmdKorzina);
            Korzina = new DataTable();
            adapKorzina.Fill(Korzina);

            // Привязка данных на DataGridView
            bsKorzina = new BindingSource();
            bsKorzina.DataSource = Korzina;
            dataGridView2.DataSource = bsKorzina;
        }
```

Для реализации перемещения товара из таблицы `Tovars` в таблицу `Korzina` на форме разместите кнопку **Выбор**.

В обработчике события кнопки **Выбор** вы должны формировать и выполнить SQL-команду `INSERT` для таблицы `Korzina`. Ниже приведен код этого обработчика события.

```
private void button1_Click(...)
{
  DataGridViewRow curRow = dataGridView1.CurrentRow;

  string TovarID = curRow.Cells["TovarID"].Value.ToString() + ", ";
  string TovarName =
            "'" +curRow.Cells["TovarName"].Value.ToString() + "', ";
  string Price = curRow.Cells["Price"].Value.ToString() + ", ";
  string Kol = textBox1.Text;

  string insertStr = "INSERT INTO Korzina " +
                       "(TovarID, TovarName, Price, Kol) " +
            "VALUES (" + TovarID + TovarName + Price + Kol + ")";

  SqlCommand insertCmd = new SqlCommand(insertStr, conn);
  insertCmd.ExecuteNonQuery();

  // Вывод данных обновленной таблицы
  Korzina.Clear();
  adapKorzina.Fill(Korzina);

  textBox2.Text = SumTovars().ToString();
}
```

24) Использование транзакций. Команды для формирования тела транзакции, для подтверждения и отказа от транзакции

Для реализации группового выполнения `SQL`-запросов предназначены транзакции. `SQL`-запросы можно сгруппировать в **транзакцию**, а затем зафиксировать результат или откатить эту транзакцию как одно целое. Например, команд `INSERT` и `UPDATE` в примере компьютерного салона можно поместить в транзакцию, а затем подтвердить или откатить эту транзакцию как одно целое, в зависимости от того, выполнились ли успешно оба `SQL`-запроса.

Для начала транзакции используется оператор `BEGIN TRANSACTION` (сокращенно `BEGIN TRANS`), а далее следуют составляющие транзакцию `SQL`-команды. Чтобы зафиксировать транзакцию, исполняется запрос `COMMIT TRANSACTION` (сокращенно

COMMIT), а для отката транзакции выполняется запрос ROLBACK TRANSACTION (сокращенно ROLLBACK).

измените код кнопки выбора товаров в Корзину.

```
private void button1_Click(...)
{
   DataGridViewRow curRow = dataGridView1.CurrentRow;

    string TovarID = curRow.Cells["TovarID"].Value.ToString();
    string TovarName = "'" + (string)curRow.Cells["TovarName"].Value
                                                          + "', ";
    string Price = curRow.Cells["Price"].Value.ToString() + ", ";
    string Kol = textBox1.Text;

    string insertStr = "INSERT INTO Korzina " +
                          "(TovarID, TovarName, Price, Kol) " +
        "VALUES (" + TovarID + ", " + TovarName + Price + Kol + ")";
    SqlCommand insertCmd = new SqlCommand(insertStr, conn);


    // Объекты для обновления таблицы Tovari
    int curKol = (int)curRow.Cells["Kol"].Value;
    curKol = curKol - Convert.ToInt32(Kol);

    string updateStr = "UPDATE Tovars SET  Kol = " + curKol +
                           " WHERE TovarID =" + TovarID;
    SqlCommand updateCmd = new SqlCommand(updateStr, conn);

    // Создание объектов для транзакции
    SqlCommand transCmd = new SqlCommand("BEGIN TRANSACTION", conn);
    transCmd.ExecuteNonQuery();

    insertCmd.ExecuteNonQuery();
    updateCmd.ExecuteNonQuery();

    transCmd.CommandText = "COMMIT";
    transCmd.ExecuteNonQuery();
 . . . . . .
}
```

25) Использование хранимых процедур и триггеров для отбора скалярных и векторных данных.

Хранимые процедуры представляют собой набор команд, которые могут храниться в SQL-сервере в компилированном виде. Таким образом, вместо того, чтобы написать используемый запрос, клиентские приложения могут вызывать нужную хранимую процедуру. Это обеспечивает лучшую производительность, поскольку данный запрос должен анализироваться только однажды и уменьшается трафик между сервером и клиентом. Еще одним преимуществом использования хранимых процедур является то, что за счет создания в SQL-сервере библиотеки хранимых процедур, можно создать набор правил для всей организации.

Триггер представляет собой хранимую процедуру, которая активизируется при наступлении определенного события. Например, можно определить процедуру, которая

срабатывает каждый раз при наступлении нового дня, чтобы пересчитать процентные начисления кредита.

Хранимые процедуры и триггеры создаются на языке, зависящем от производителя СУБД. Например, в случае MS SQL Server хранимые процедуры создаются на языке, известном под названием **Transact-SQL**.

Хранимые процедуры можно создавать для возврата, как набора строк, так и скалярных типов данных. В результате получается обычная процедура, которая хранится в файле удаленной базы данных и выполняется на компьютере-сервере.

Для создания хранимой процедуры в окне Обозреватель серверов выделите узел Хранимые процедуры и из контекстного меню выполните команду Добавить новую хранимую процедуру. При этом появляется окно редактора, которое содержит заголовок хранимой процедуры. Измените этот код на следующий код:

```
CREATE PROCEDURE GetTovar
      @TovarID int = 5,
      @TovarName char(10) OUTPUT
AS
    SELECT @ TovarName = TovarName FROM  Tovars
                                WHERE TovarID = @ TovarID
```

Заметьте, что в этом примере хранимая процедура в качестве параметра получает один скалярный параметр @TovarID, и возвращает другой скалярный параметр @TovarName.

Сохраните эту процедуру командой **Файл/ Сохранить...**. При сохранении этой процедуре автоматически будет присвоено имя GetTovar, взятое из оператора CREATE PROCEDURE. После этого новая хранимая процедура будет видна в окне Обозреватель серверов.

```
SqlCommand cmd = new SqlCommand("GetTovar", conn);
cmd.CommandType = CommandType.StoredProcedure;

//Входной параметр
SqlParameter param = new SqlParameter();
param.ParameterName = "@TovarID";
param.SqlDbType = SqlDbType.Int;
param.Value =Convert.ToInt32(textBox1.Text);
param.Direction = ParameterDirection.Input;
cmd.Parameters.Add(param);


//Выходной параметр
param = new SqlParameter();
param.ParameterName = "@TovarName";
param.SqlDbType = SqlDbType.Char;
param.Size = 10;
param.Direction = ParameterDirection.Output;
cmd.Parameters.Add(param);

// Вызов хранимой процедуры
cmd.ExecuteNonQuery();
textBox2.Text = (string)cmd.Parameters["@TovarName"].Value;
```

Теперь запустите приложение, введите в текстовое поле textBox1 значение ID товара и нажмите на кнопку. Тогда в текстовом поле **textBox2** появляется фамилия служащего

окне Обозреватель серверов создайте хранимую процедуру **GetTovars**, которая должна возвращать значения полей TovarName и Price всех записей таблицы Tovars. Код этой хранимой процедуры имеет следующий вид:

```
CREATE PROCEDURE "GetTovars" AS
      SELECT TovarName, Price  FROM   Tovars
GO

private void button1_Click(...)
{
    SqlCommand cmd = new SqlCommand("GetTovars", conn);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.ExecuteNonQuery();

    adapTovars.SelectCommand = cmd;
    DataTable Tovars2 = new DataTable();
    adapTovars.Fill(Tovars2);
    dataGridView2.DataSource = Tovars2;
}
```