Приложения для работы с базами данных

- http://msdn.microsoft.com/ru-ru/library
- http://www.gotdotnet.ru/

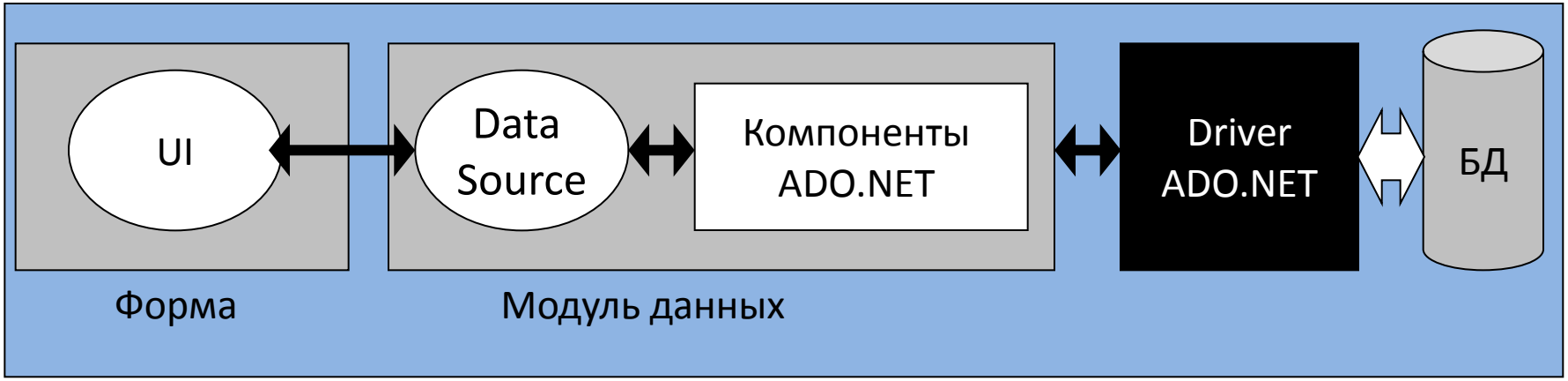
Архитектура приложений для работы с базами данных

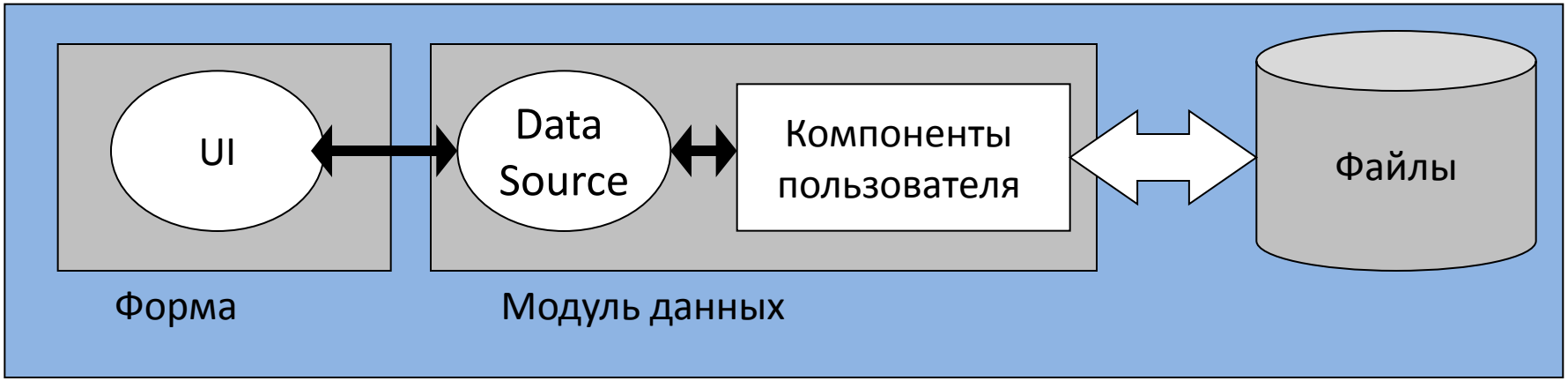
- Одноуровневая (Single-tiered)
- Двухуровневая (Two-tiered)
- Многоуровневая (Multi-tiered)

Одноуровневая архитектура

- Основывается на:
  - Локальных (файловых) базах данных (Paradox, dBase, Access, FoxPro)
  - На типизированных файлах
- Бизнес-логика, как правило, реализуется приложением-клиентом

Одноуровневая архитектура





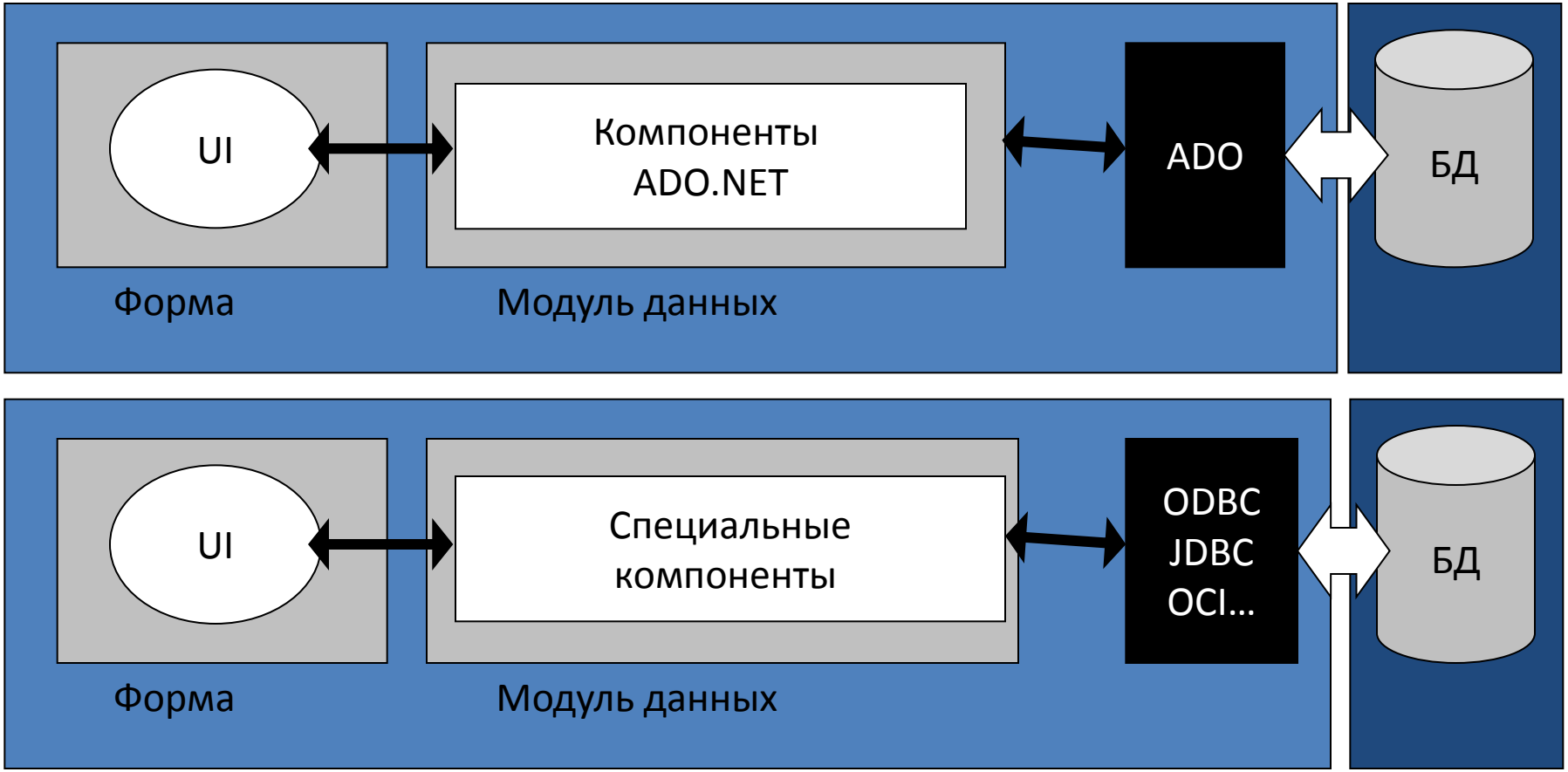
Одноуровневая архитектура

- Достоинства
  - Простота разработки
  - Скорость работы
- Недостатки
  - Ограничения при работе с большими объемами данных
  - Ограничения при работе нескольких пользователей
  - Проблемы с безопасностью

Двухуровневая архитектура

- Основывается на клиент-серверных базах данных (Oracle, DB2, Sybase, MySQL и др.)
- Бизнес-логика распределена между сервером базы данных и приложением-клиентом
- Обмен данными осуществляется в большинстве случаев по локальной сети (LAN), реже по глобальной (WAN)

Двухуровневая архитектура



Типы клиентов

- «Толстый» - содержит в себе логику работы приложения с данными (бизнес-логику)
- «Тонкий» - применяется только для отображения данных, вся бизнес-логика реализуется внутри базы данных

Двухуровневая архитектура

- Достоинства
  - Практически нет ограничений на объем данных
  - Быстрое время реакции приложения
  - Безопасность
- Недостатки
  - Нужен специальный отдельный сервер для хранения БД
  - Проблемы при работе в WAN
  - Бизнес-логика распределена между клиентом и сервером
  - Определенные требования к клиентским местам
  - Проблемы при обновлении версии приложения в случае «толстого» клиента

Многоуровневая архитектура

- Отличается от двухуровневой наличием одного или нескольких слоев, реализующих бизнес-логику
- Для работы с бизнес-логикой применяются серверы приложений
- Используется «тонкий» клиент

Многоуровневая архитектура

Многоуровневая архитектура

- Достоинства
  - Четкое разделение обязанностей между уровнями представления данных, работы с ними и хранения данных
  - Возможность практически неограниченного масштабирования
- Недостатки
  - Трудность в реализации
  - Необходимы отдельные серверы для сервера приложений и базы данных

Средства для работы с базами данных

- ODBC – Open DataBase Connectivity
- ADO – ActiveX Database Object
- Специальные библиотеки от производителя СУБД (например OCI)

Open DataBase Connectivity

- Стандартный способ доступа к базам данных. ODBC – драйверы есть практически для всех СУБД

Open DataBase Connectivity

- Достоинства
  - Независимость от платформы
  - Стандартизированный интерфейс
- Недостатки
  - Сложность использования

ActiveX DataBase Objects

- Набор компонент, позволяющих работать с базами данных, используя OLE DB
- Технология OLE DB была разработана Microsoft и работает только на платформе Windows

ActiveX DataBase Objects

- Достоинства
  - Более высокая скорость работы с базами данных, чем у ODBC
  - Нет необходимости в установке дополнительных библиотек
- Недостатки
  - Привязка к платформе Windows

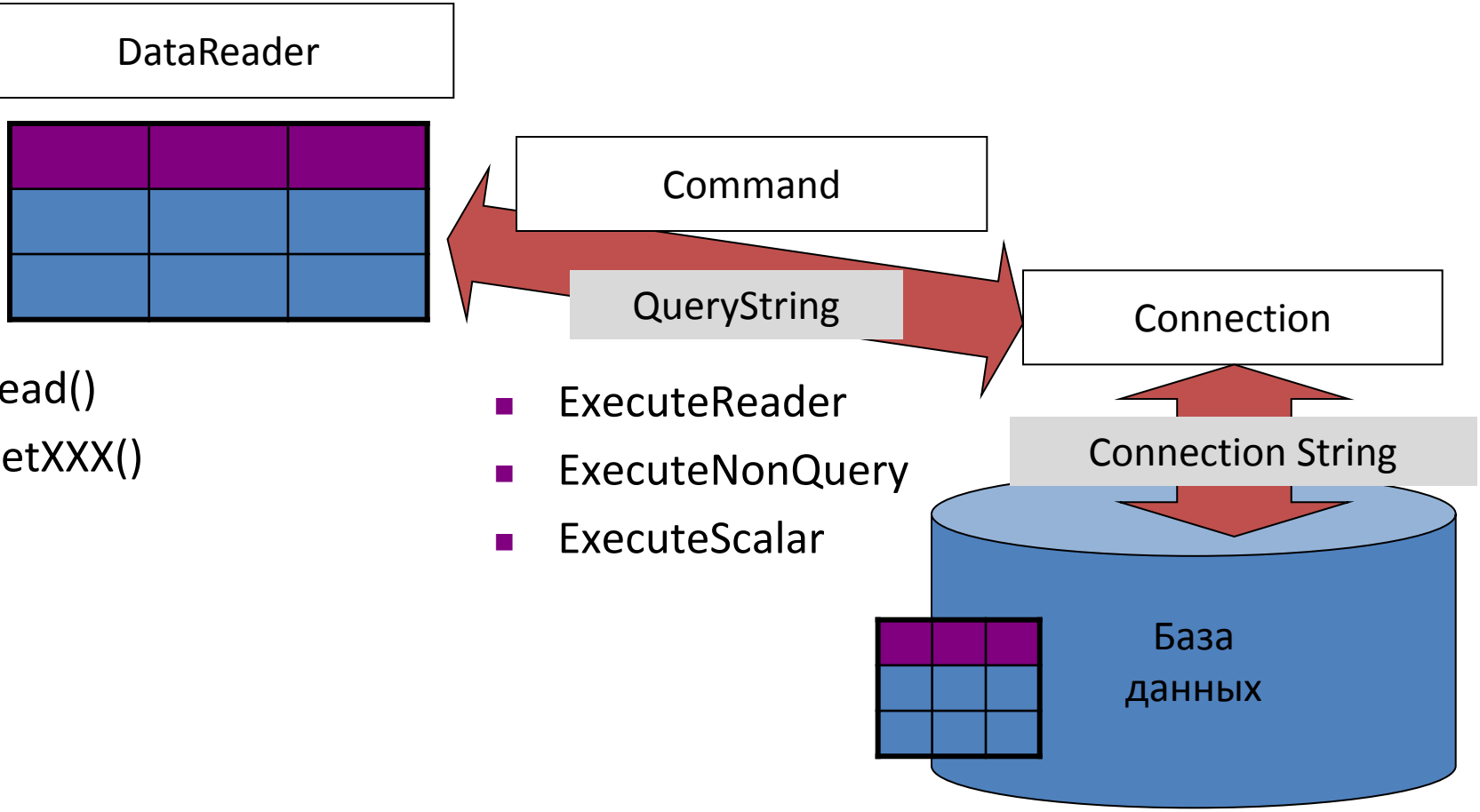
Компоненты ADO.NET

- Connected
  - Connection
  - Transaction
  - Command
    - Parameter
  - DataReader
  - DataAdapter

Data Provider

- Набор классов, предназначенных для соединения с БД. Каждый производитель СУБД реализует свой Data Provider
- Все Data Provider реализуют интерфейсы Connected компонентов

Взаимодействие Connected компонент ADO.NET



- Read()
- GetXXX()

- ExecuteReader
- ExecuteNonQuery
- ExecuteScalar

Соединение с БД

- Для соединения с БД используется класс Connection, которому указывается Connection String

```
OleDbConnection conn = new OleDbConnection();
conn.ConnectionString = “Provider=sqloledb;Data Source=localhost\DB;Initial Catalog=DG;Integrated Security=SSPI”;
conn.Open();
//Делаем что-то здесь
conn.Close();
```

Выборка данных

- Для выборки данных используется экземпляр класса Command и DataReader
- Выполнение команды не обязательно подразумевает возврат данных
  - ExecuteReader
  - ExecuteNonQuery
  - ExecuteScalar

Выборка данных

```
public void ReadMyData(string connectionString)
{
   string queryString = "SELECT OrderID, CustomerID FROM Orders";

   using (OleDbConnection connection = new OleDbConnection(connectionString))
   {

      OleDbCommand command = new OleDbCommand(queryString, connection);

      connection.Open();

      OleDbDataReader reader = command.ExecuteReader();

      while (reader.Read())
      {
         Console.WriteLine(reader.GetInt32(0) + ", " + reader.GetString(1));
      }

      reader.Close();
   }
```

1 2 3 4 5 6 7 8

Работа с подключениями

- Закрывайте Connection по окончании работы с базой (try…finally)
- Закрывайте DataReader по окончании работы с данными

Дополнительные возможности

- Transaction
  - Позволяет группировать набор операторов в транзакцию
  - Транзакция вызывается у экземпляра OleDbConnection
    - Commit
    - Rollback
    - Dispose

Дополнительные возможности

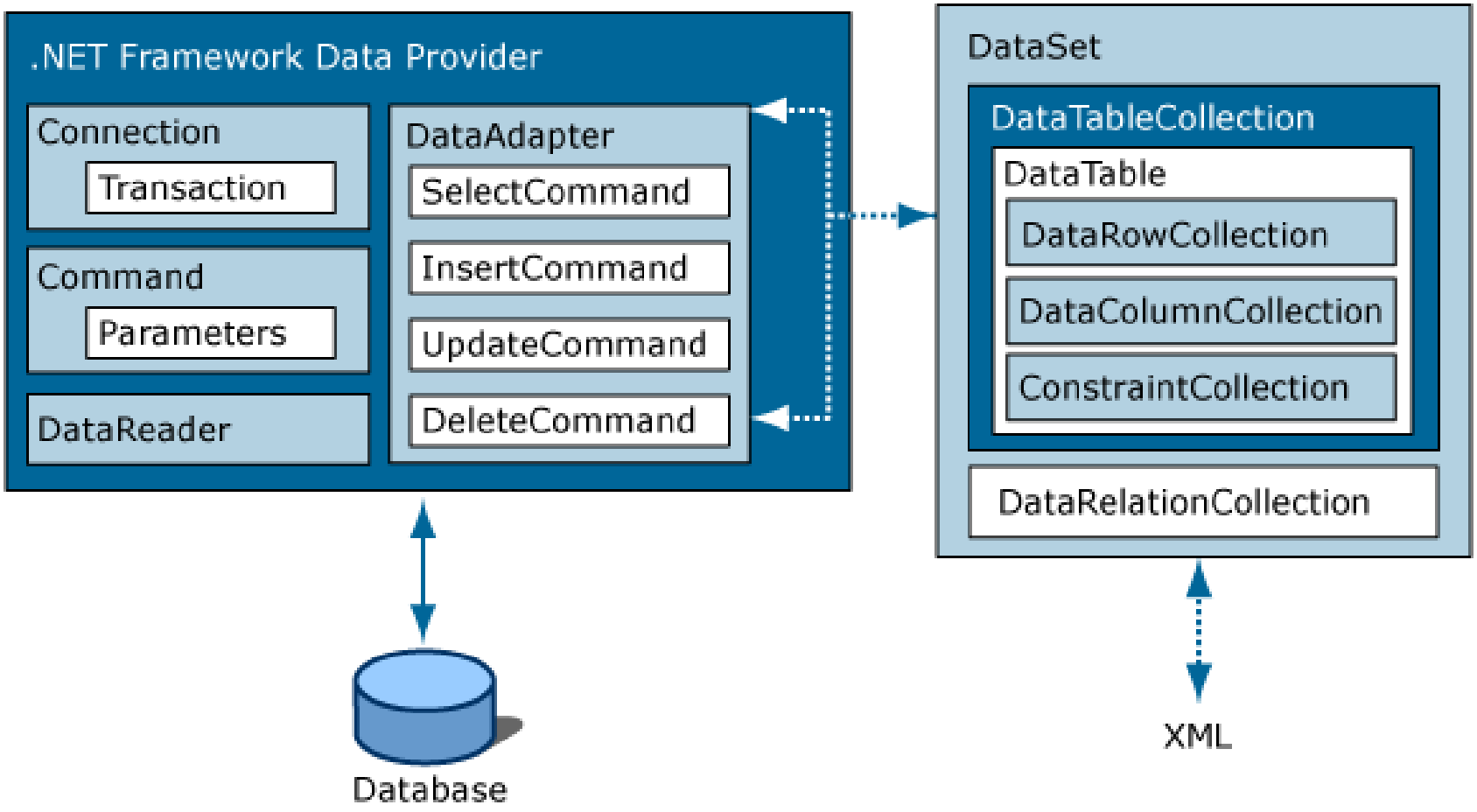
- Parameter
  - Позволяет задавать параметры запроса
  - Предпочтительнее конкатенации строк

```
string queryString = "SELECT * FROM Table1 WHERE Field1 LIKE ?";

OleDbCommand command = new OleDbCommand(queryString, connection);

command.Parameters.Add("@p1", OleDbType.Char, 3).Value = "a";

OleDbDataReader reader = command.ExecuteReader();
```

Компоненты ADO.NET

- Connected
  - DataAdapter
- Disconnected
  - DataSet
    - DataTable -> DataView
      - DataRow
      - DataColumn
      - Constraint
    - DataRelation

Компоненты ADO.NET



DataSet

- Класс, который содержит набор данных. Можно рассматривать как «контейнер» для таблиц. Все данные, которые хранятся в DataSet, отсоединены от БД

- Используйте DataSet, если нужно
  - Локально кешировать данные
  - Подсоединять интерфейсные компоненты к данным
  - Обрабатывать данные без обновления их в БД

DataTable

- Основной класс, который хранит данные
- Состоит из строк и столбцов
- Обычно хранится в составе DataSet
- Select – поиск по выражению

```
DataSet customers = new DataSet();

DataTable customersTable = customers.Tables.Add("CustomersTable");

DataRow[] rows = customersTable.Select (“last_order_date > ’01.01.2009’ ”)
```

DataColumn

- Объект, описывающий столбец таблицы.

```
DataTable table = new DataTable("Product");

DataColumn column = new DataColumn();
column.DataType = System.Type.GetType("System.Decimal");
column.AllowDBNull = false;
column.Caption = "Price";
column.ColumnName = "Price";
column.DefaultValue = 25;

table.Columns.Add(column);
```

DataRow

- Представление строки таблицы

```
DataRow row;

   for(int i = 0; i < 10; i++)
   {
      row = table.NewRow();
      row["Price"] = i + 1;
      table.Rows.Add(row);
   }
```

DataView

- Объект для просмотра содержимого DataTable
  - Find – поиск по первичному ключу
  - Sort – сортировка по столбцам
  - RowFilter – фильтрация строк

DataView

```
DataView view = new DataView();

view.Table = DataSet1.Tables["Suppliers"];
view.AllowDelete = true;
view.AllowEdit = true;
view.AllowNew = true;
view.RowFilter = "City = 'Berlin'";
view.RowStateFilter = DataViewRowState.ModifiedCurrent;
view.Sort = "CompanyName DESC";
```

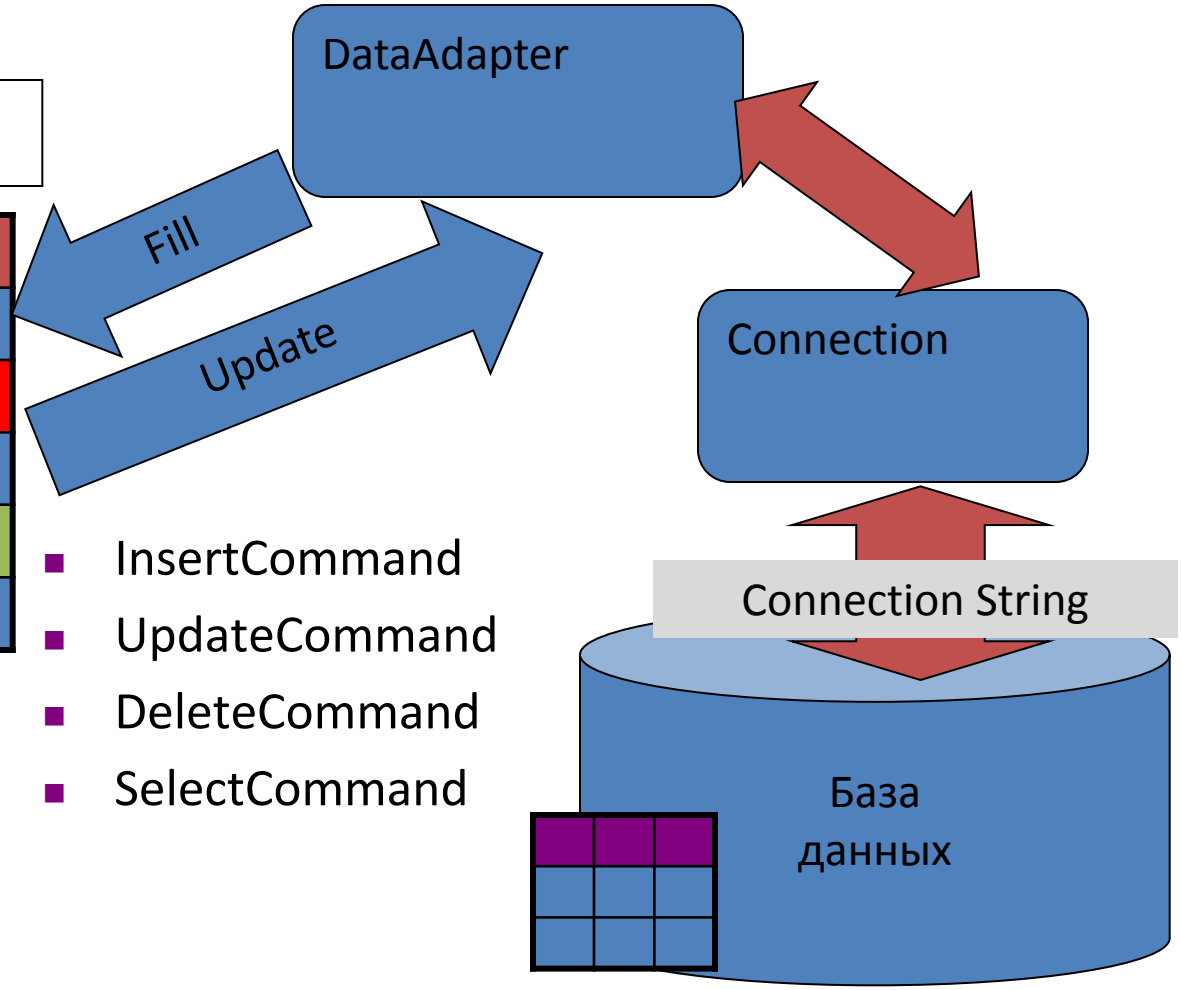
DataAdapter

- Объект для связи Connected и Disconnected объектов
  - InsertCommand
  - UpdateCommand
  - DeleteCommand
  - SelectCommand
  - Fill
  - Update

DataAdapter

DataTable

ID	Name	CITY
1	AB	23
2	JOE	55
3	JACK	22
4	JIM	23
5	BORIS	11



- InsertCommand
- UpdateCommand
- DeleteCommand
- SelectCommand

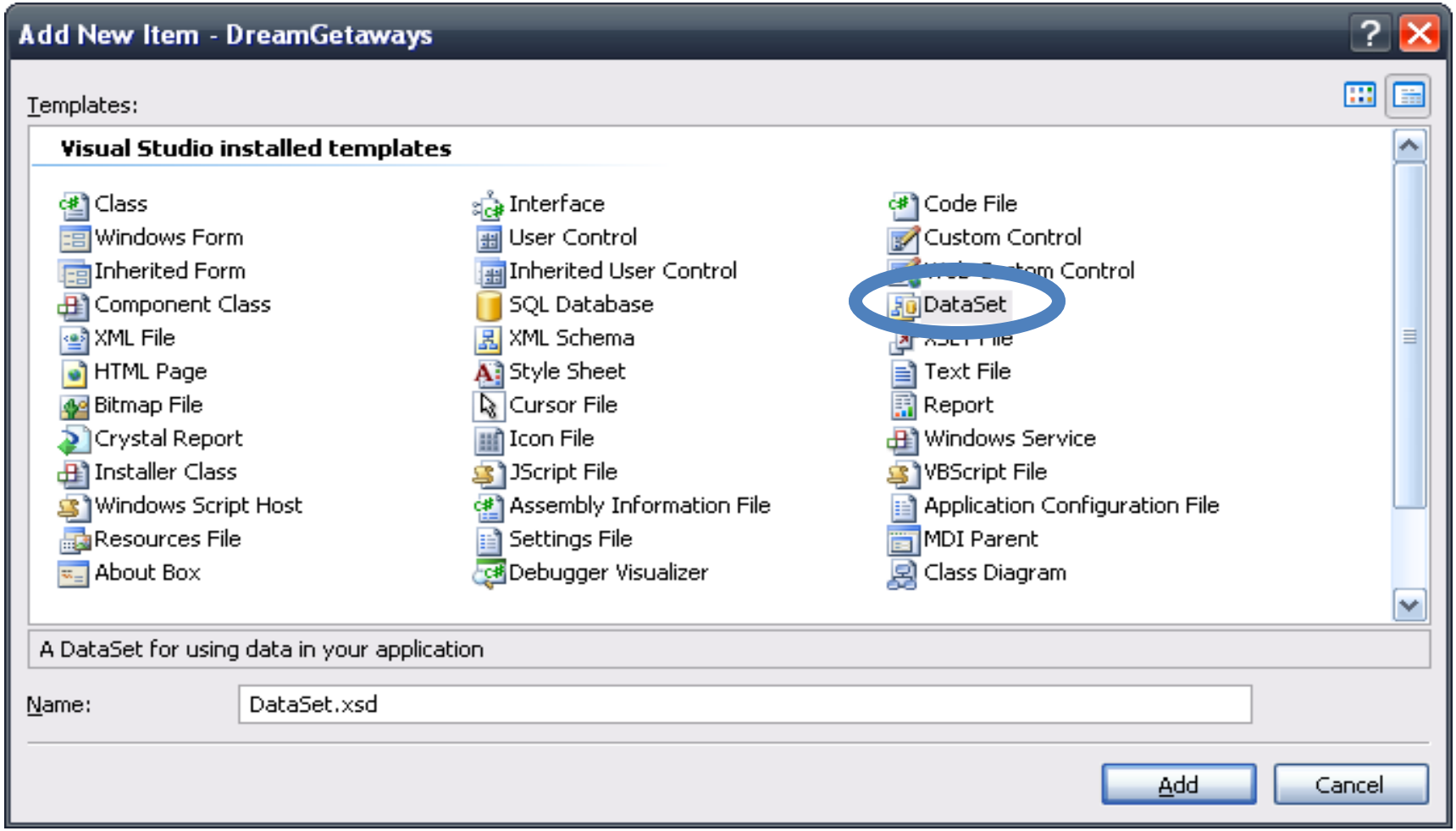
DataSet

- При работе с обычным DataSet существуют некоторые неудобства
  - «Ручное» создание DataSet и таблиц
  - Обращение к таблицам по имени
  - В Design-Time нет никакой информации о таблицах в DataSet

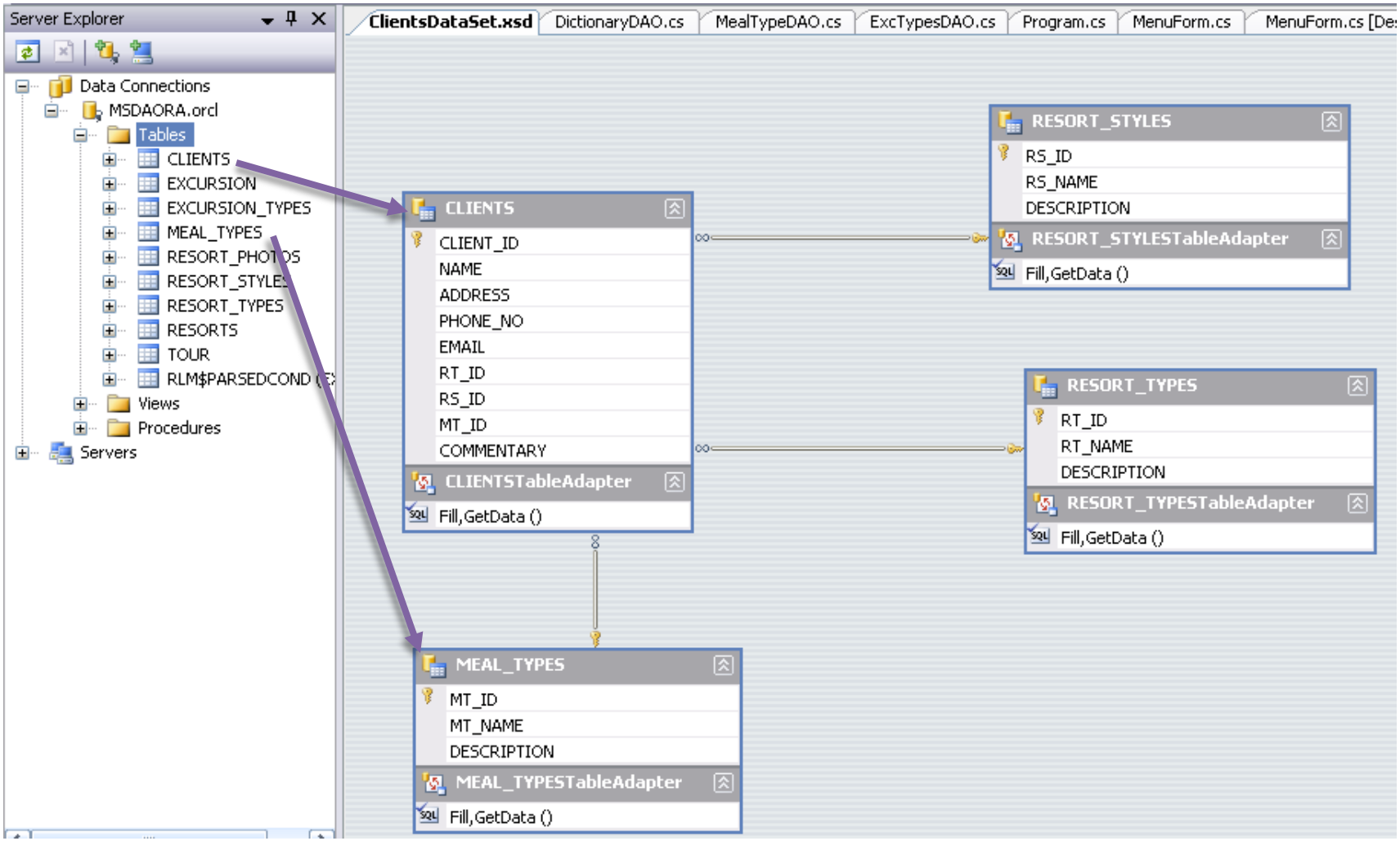
Типизированный DataSet

- Создается при помощи визуального редактора
- Обращение к содержимому с использованием «свойств»
- Генерация вспомогательных типов данных
  - Типизированные таблицы
  - Типизированные строки таблиц
  - Адаптеры

Типизированный DataSet



Типизированный DataSet



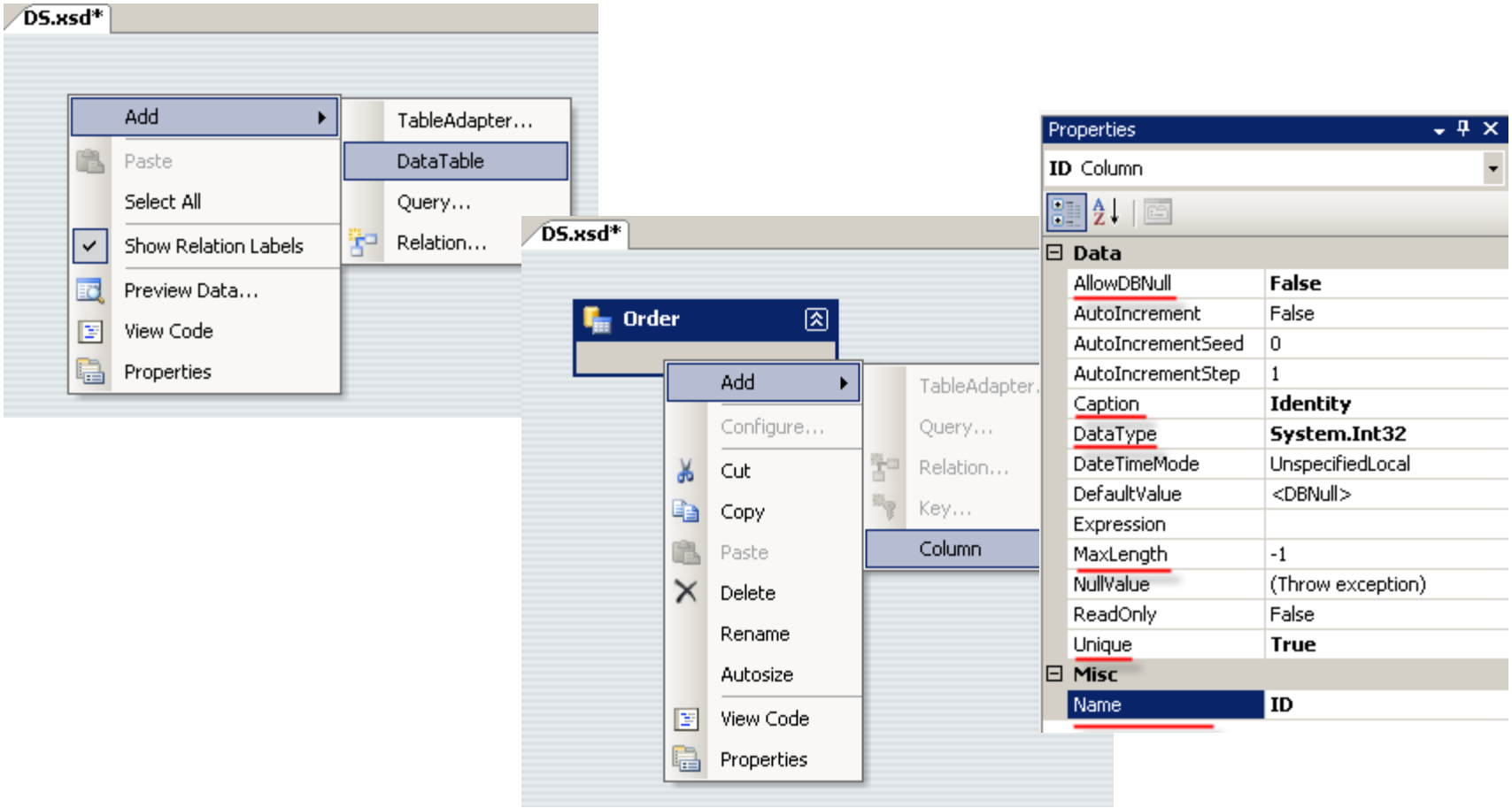
Типизированный DataSet

- Автоматически генерируются классы с поддержкой структуры данных, соответствующих таблицам
  - CLIENTS
  - CLIENTSRow (с полями в виде свойств класса)
  - CLIENTSTableAdapter

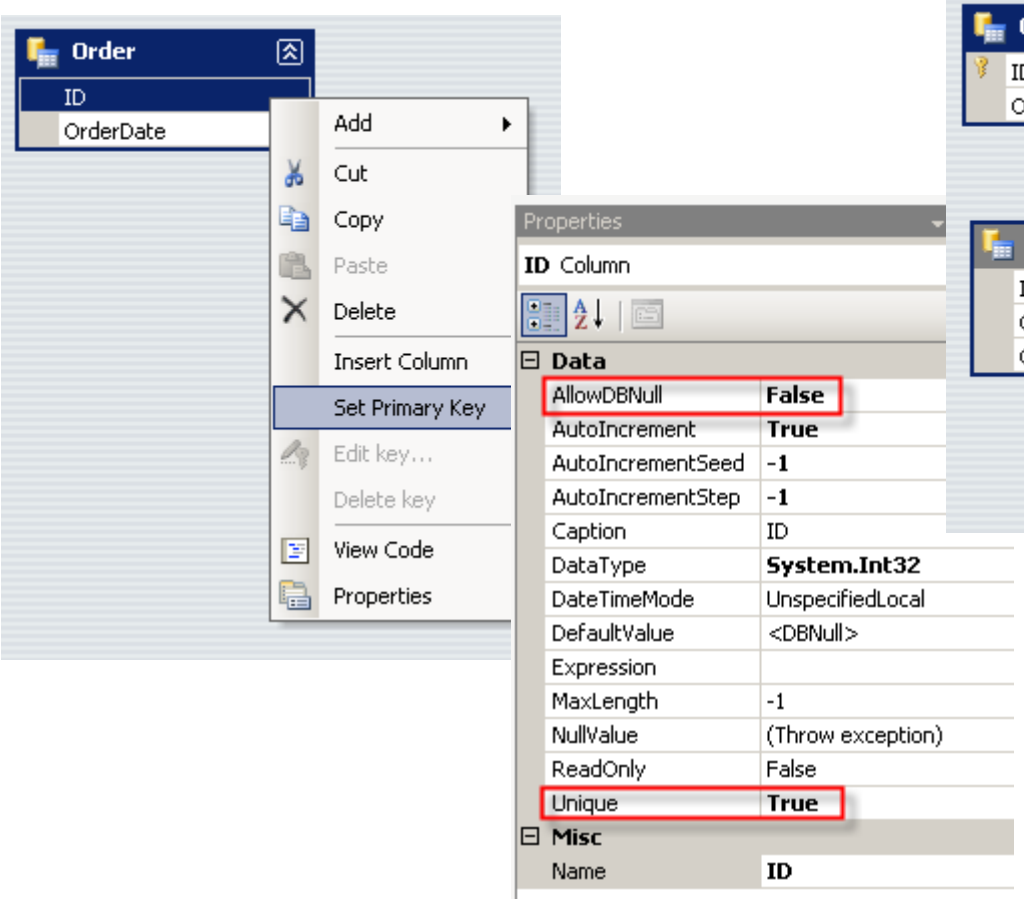
Типизированный DataSet

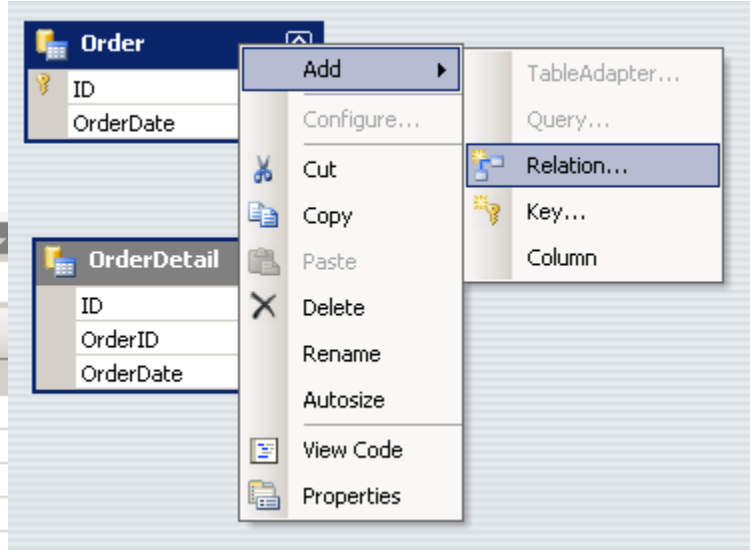
```
ClientsDataSet dataSet = new ClientsDataSet();

ClientsTableAdapter da = new ClientsTableAdapter();

da.Fill(dataSet.Clients);

dataSet.Clients.Rows[0].Name = “Jack”;

//Для обычного DataSet
dataSet.Tables[“CLIENTS”].Rows[0][“NAME”] = “Jack”;
```

DataSet


DS.xsd*
Add
Paste
Select All
Show Relation Labels
Preview Data...
View Code
Properties
TableAdapter...
DataTable
Query...
Relation...
DS.xsd*
Order
Add
Configure...
Cut
Copy
Paste
Delete
Rename
Autosize
View Code
Properties
TableAdapter...
Query...
Relation...
Key...
Column
Properties
ID Column
Data
AllowDBNull False
AutoIncrement False
AutoIncrementSeed 0
AutoIncrementStep 1
Caption Identity
DataType System.Int32
DateTimeMode UnspecifiedLocal
DefaultValue <DBNull>
Expression
MaxLength -1
NullValue (Throw exception)
ReadOnly False
Unique True
Misc
Name ID

DataSet

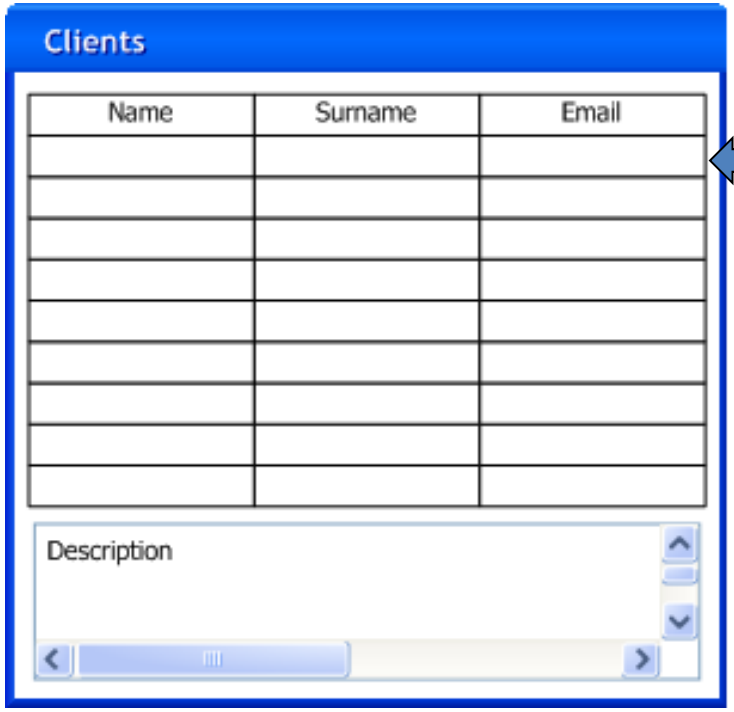

Order
ID
OrderDate
Add
Cut
Copy
Paste
Delete
Insert Column
Set Primary Key
Edit key...
Delete key
View Code
Properties
Properties
ID Column
Data
AllowDBNull False
AutoIncrement True
AutoIncrementSeed -1
AutoIncrementStep -1
Caption ID
DataType System.Int32
DateTimeMode UnspecifiedLocal
DefaultValue <DBNull>
Expression
MaxLength -1
NullValue (Throw exception)
ReadOnly False
Unique True
Misc
Name ID


Order
ID
OrderDate
OrderDetail
ID
OrderID
OrderDate
Add
Configure...
Cut
Copy
Paste
Delete
Rename
Autosize
View Code
Properties
TableAdapter...
Query...
Relation...
Key...
Column

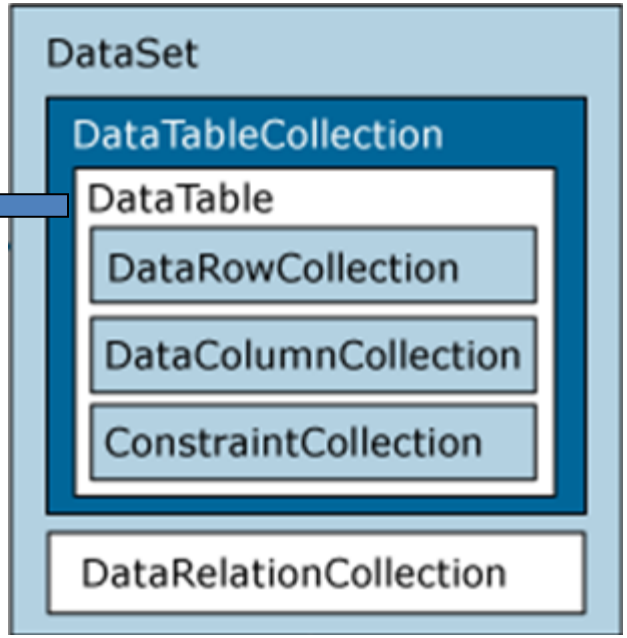
Типизированный DataSet

- FindBy*PrimaryKey*
- New*Table* Row – возвращает типизированную строку
- Add*Table*Row – можно передать экземпляр строки или набор данных

Привязка данных к Grid



DataSource

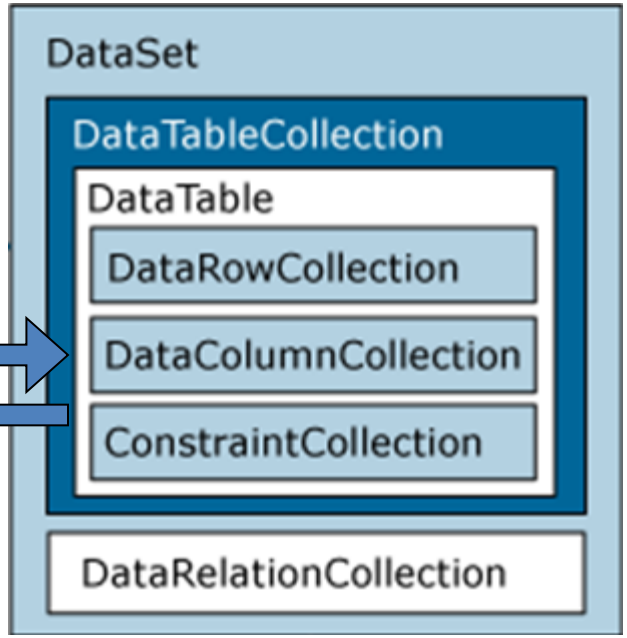


Привязка данных к Grid

```
dictionaryGridView.AutoGenerateColumns = false;

DataGridViewColumn column = new DataGridViewTextBoxColumn();
column.DataPropertyName = “Name”;
column.Name = “Name”;
column.HeaderText = “Имя";

dictionaryGridView.Columns.Add(column);
dictionaryGridView.DataSource = table;
```

Привязка данных к визуальным компонентам

Binding
- Property
- DataSource
- Column
- Formatting
- UpdateMode
  - Validate
  - Changed



Привязка данных к визуальным компонентам

```
TextBox descriptionBox = new TextBox();

Binding bind = new Binding("Text", table, dao.Description);

// Возвращает или задает момент распространения изменений,
// внесенных в источник данных, в свойство привязанного элемента управления
bind.ControlUpdateMode = ControlUpdateMode.OnPropertyChanged;

// Возвращает или задает момент распространения изменений,
// внесенных в свойство привязанного элемента управления, в источник данных.
bind.DataSourceUpdateMode = DataSourceUpdateMode.OnPropertyChanged;

descriptionBox.DataBindings.Add(bind);
```

Обновление данных

- Для сохранения данных
  - Adapter.Update()
- Для отмены изменений
  - DataTable.RejectChanges()
- Получение текущей строки
  - DataGridView.CurrentRow
- Удаление строки
  - DataTable.DefaultView.Delete(index)

Обновление данных

- Все изменения кэшируются в DataSet
- Для передачи изменений необходимо использовать Adapter
- Можно получить список измененных строк
  - DataTable.GetChanges

Обновление данных

- Проблема первичных ключей
  - При вставке строки автоматически генерируется первичный ключ
    - IDENTITY COLUMN
    - Trigger
  - Приложение не знает о сгенерированном значении
- БД-зависимый код

Обновление данных

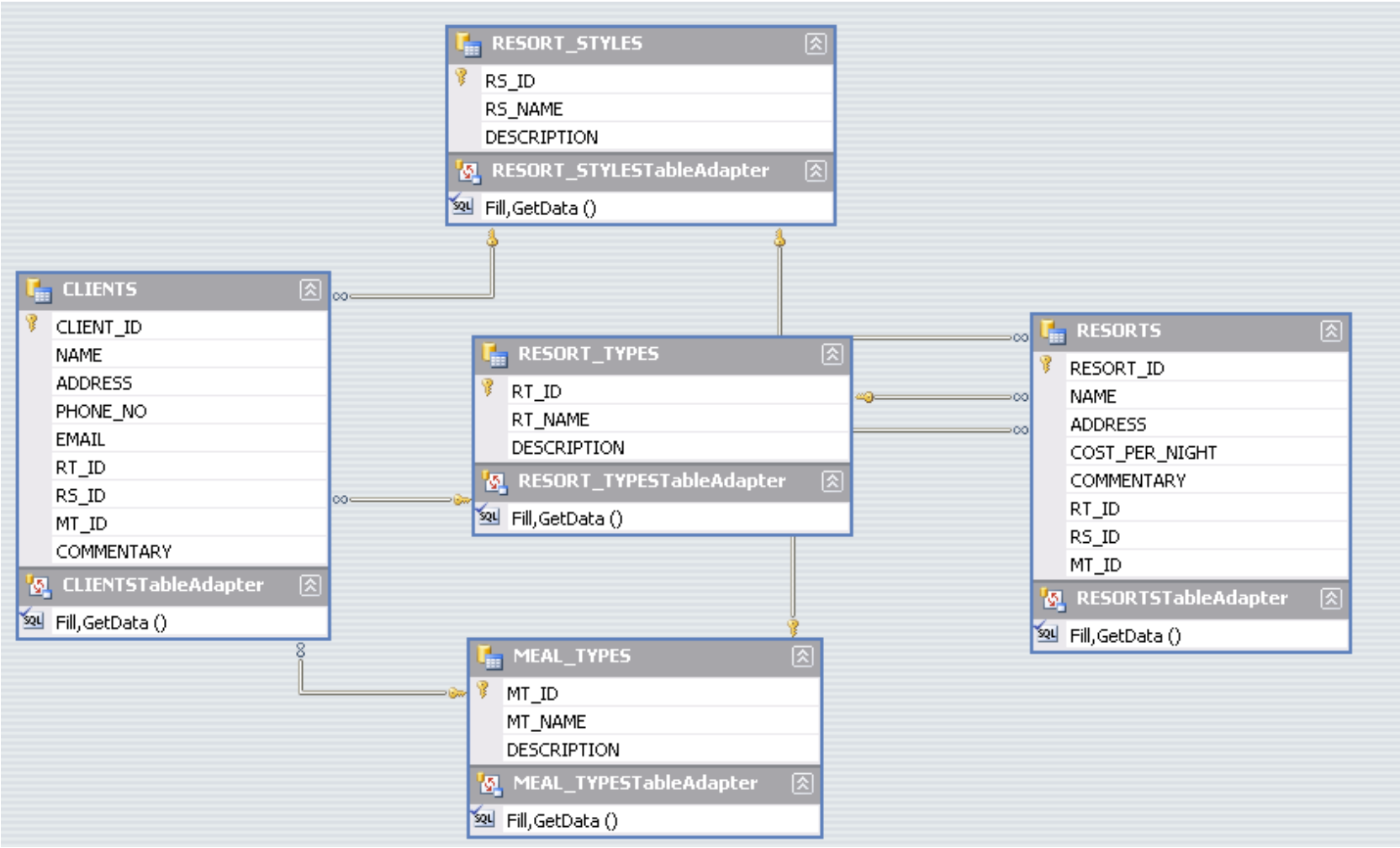
```
protected void OnRowUpdated(object sender, OleDbRowUpdatedEventArgs e)
    {

            if (e.StatementType == StatementType.Insert)
        {

         OleDbCommand cmdNewID =
    new OleDbCommand("SELECT DICT_SEQ.CURRVAL FROM DUAL", Conn);

            e.Row[Id] = cmdNewID.ExecuteScalar();
        e.Status = UpdateStatus.SkipCurrentRow;
      }
    }
```

Фильтрация данных в DataGrid

- Для фильтрации данных используется DataView, который неявно создается для DataGrid

```
 private void Filter(string filter)
{
CurrencyManager cm =
(CurrencyManager)ClientsGridView.
BindingContext[ClientsGridView.DataSource, ClientsGridView.DataMember];

DataView dv = ((DataRowView)cm.Current).DataView;
dv.RowFilter = filter;

}
```

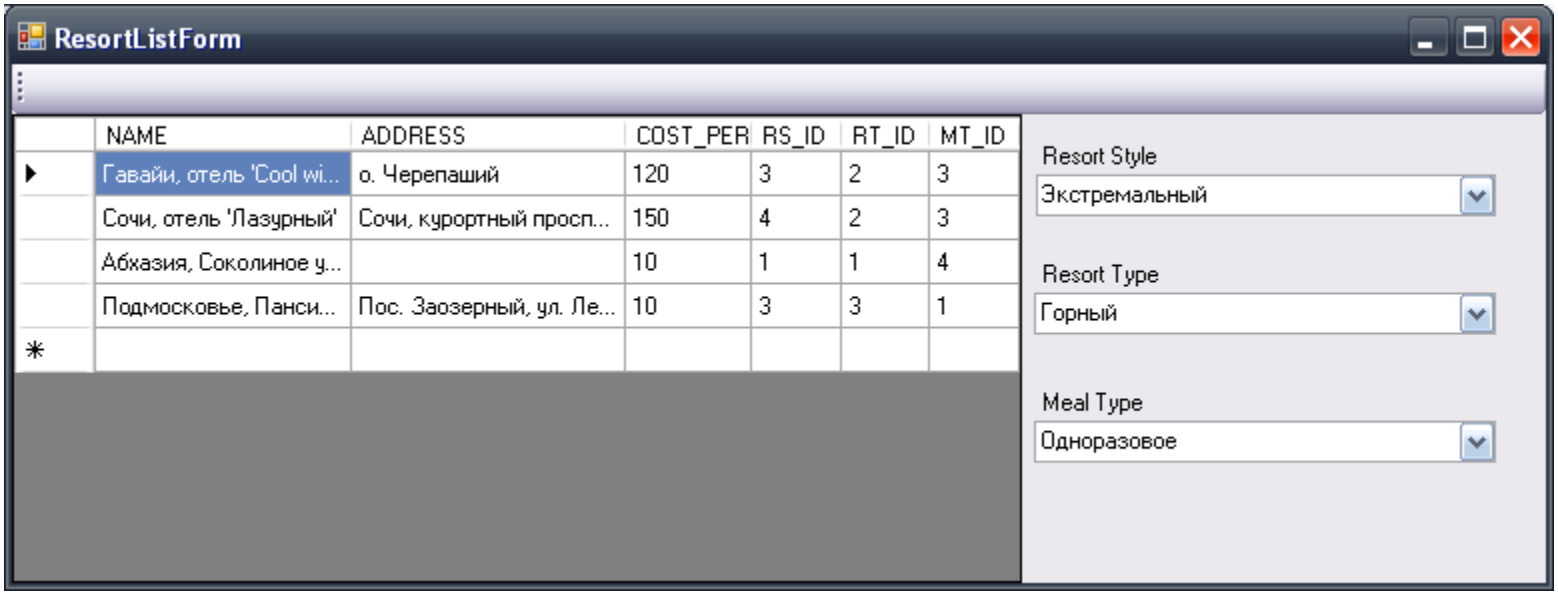
Пример DataSet



Lookup поля

- При заполнении поле внешнего ключа позволяют видеть не ссылку на данные, а значение

Пример формы Resort List



Свойства компонента ComboBox

- DataSource
- DisplayMember
- ValueMember
- SelectedValue

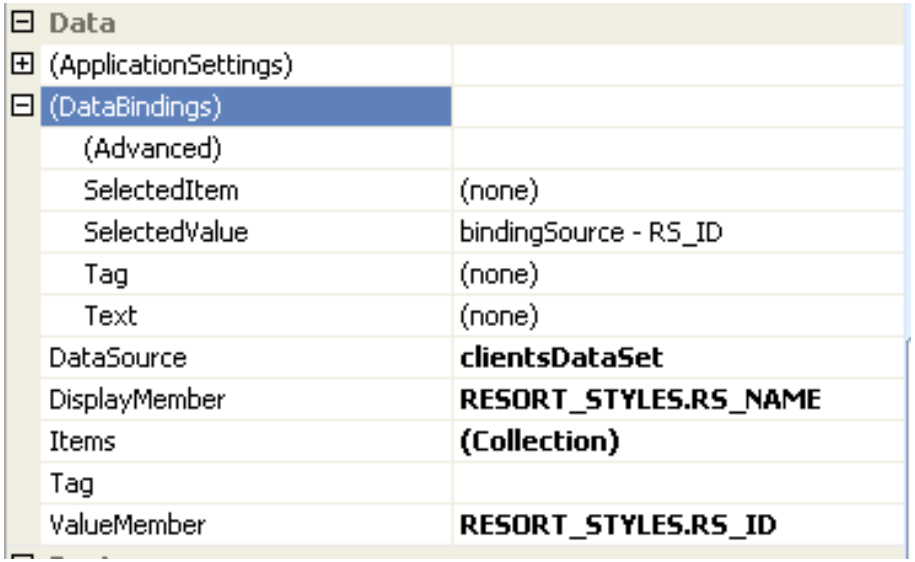
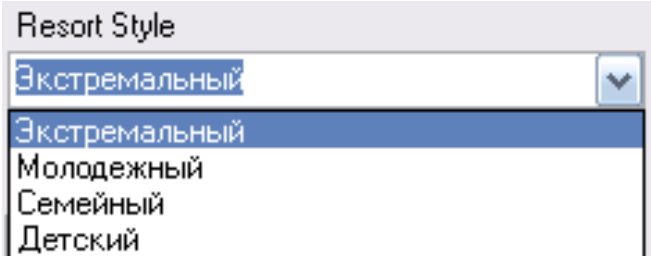


Схема работы ComboBox

DataSet.RESORTS

Resort_ID	Resort_Name	RS_ID
1	Сочи	5
2	Гагры	3
3	Канары	1

- DataSource = DataSet
- SelectedValue=RS_ID



DataSet.RESORT_STYLES

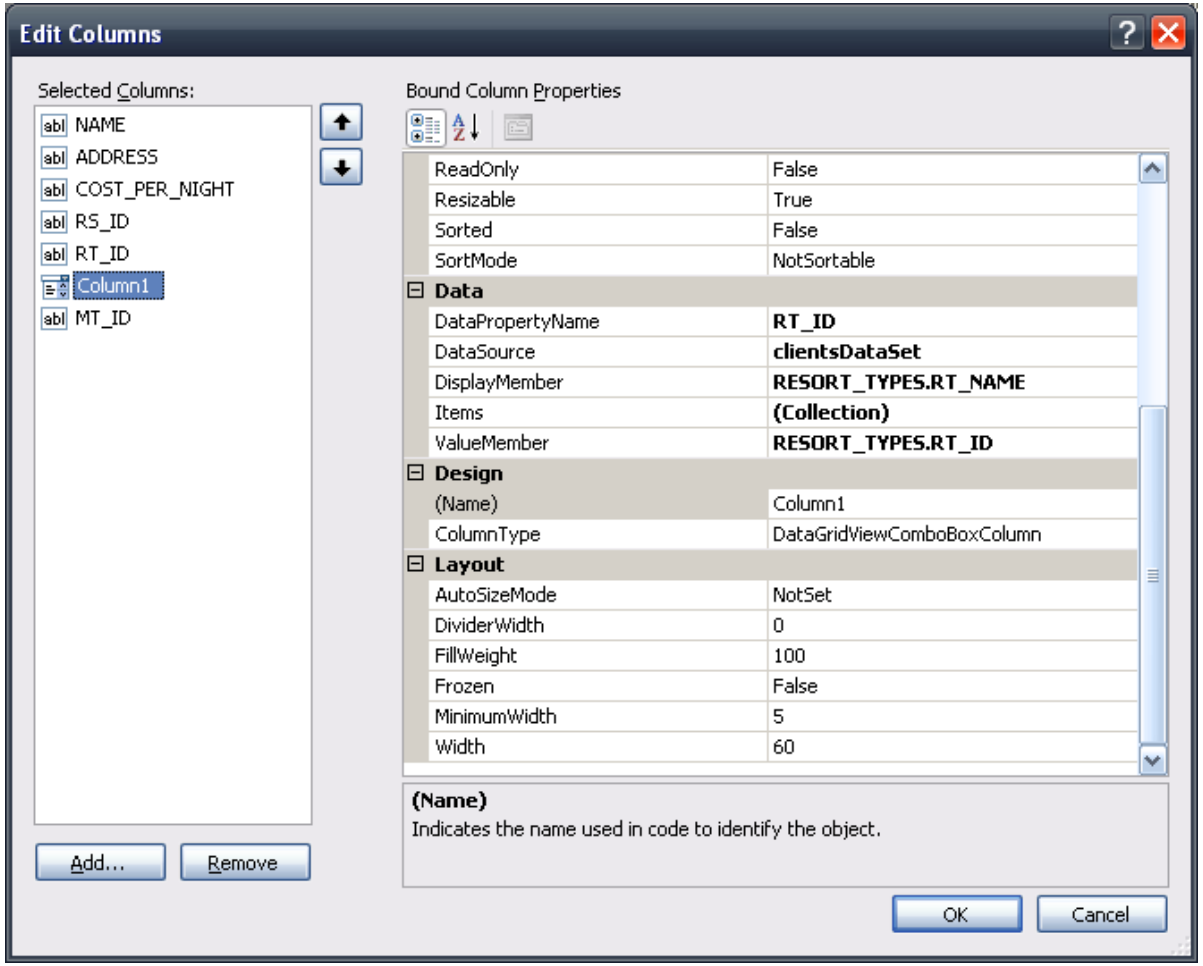
RS_ID	RS_Name	Description
1	Bill	Gates
2	Jack	Doe
3	John	Brown

- DisplayMember=RS_NAME
- ValueMember=RS_ID

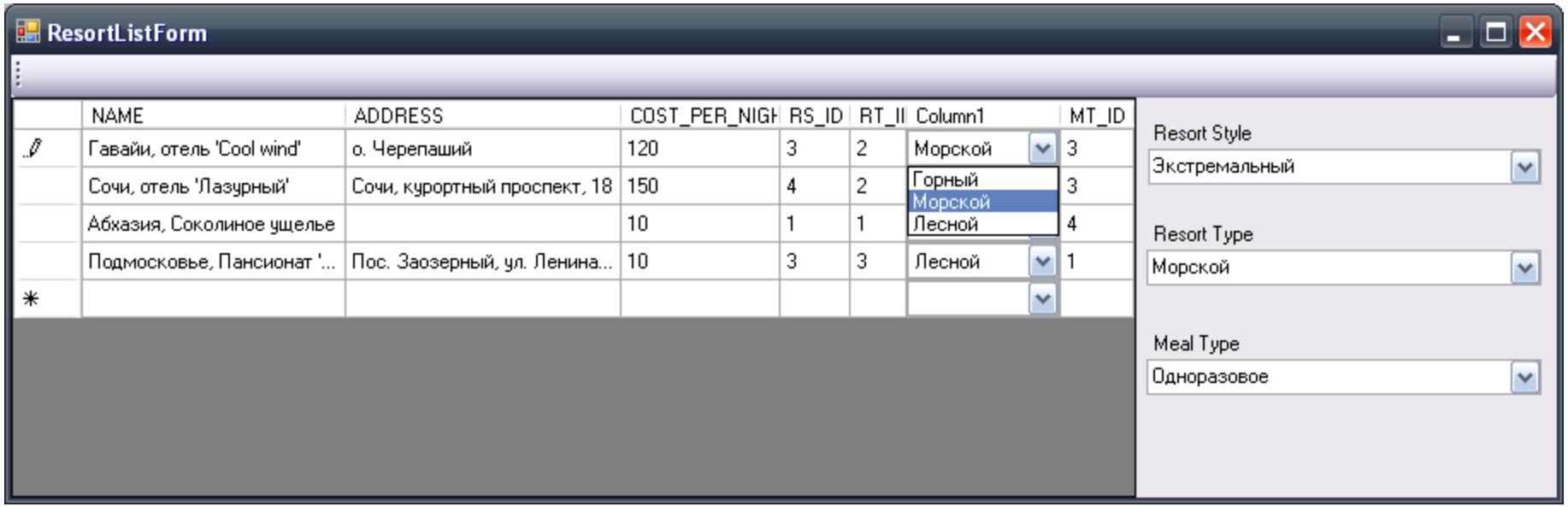
Lookup поля в Grid - формах

- Компонент DataGridVies позволяет создавать Lookup поля внутри сетки
- DataPropertyName
- DataSource
- DisplayMember
- ValueMember

Lookup поля в Grid - формах



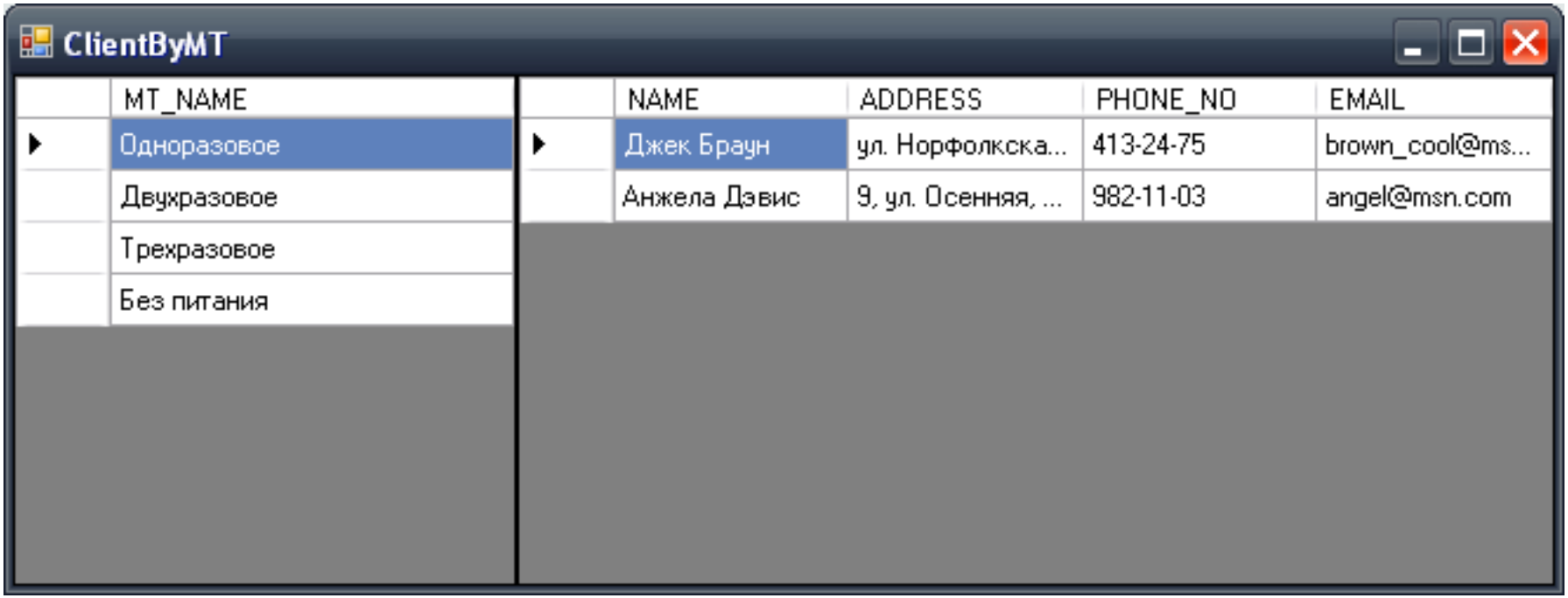
Форма Resorts с lookup полями



Реализация отношения «Мастер-детали»

- Отношение «Мастер - детали» очень часто используется при проектировании форм для работы с базами данных
- Отношение может быть реализовано как с использованием внешних ключей, так и без них

Форма поиска клиентов по предпочтениям



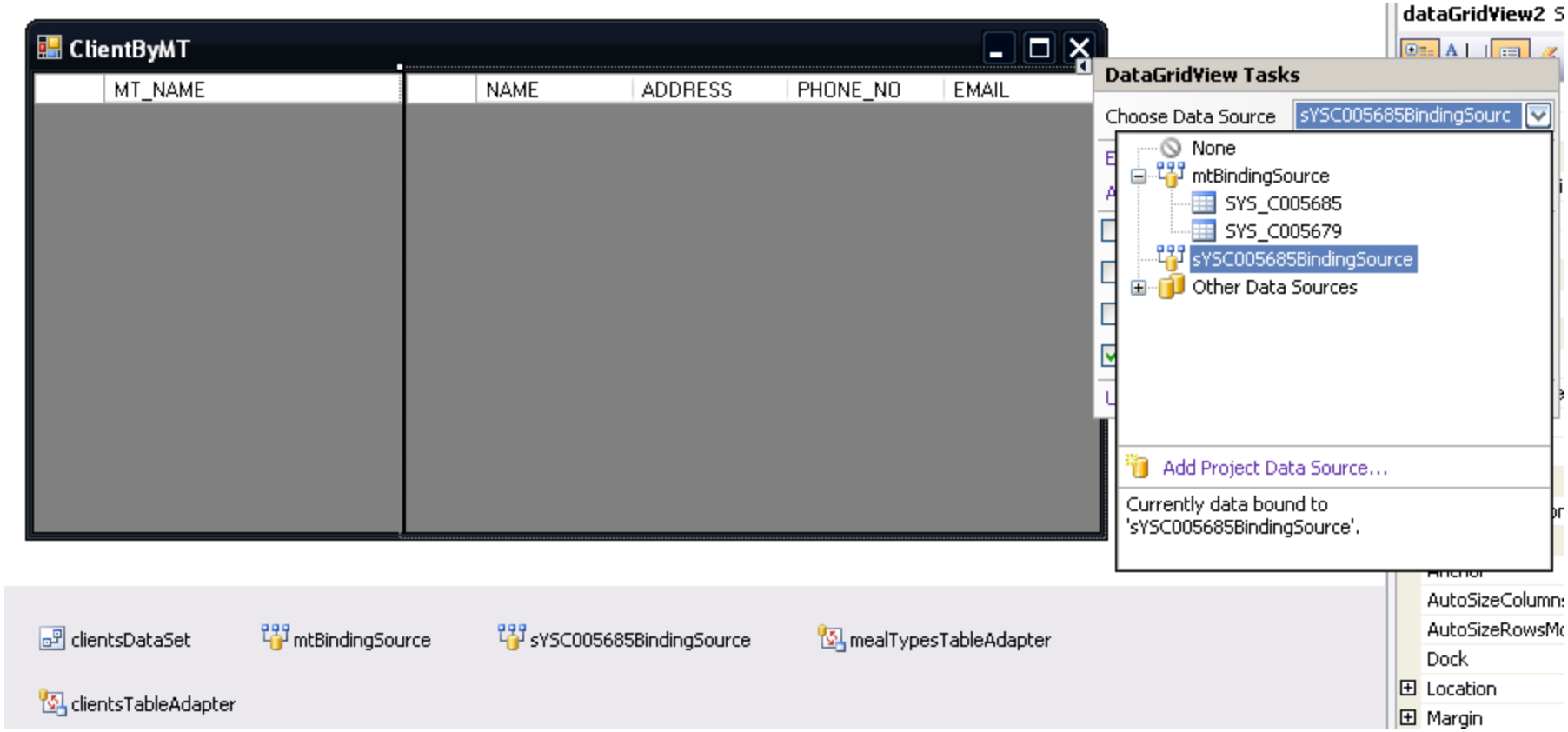
Реализация отношения «Мастер-детали»

- При реализации отношения используются два BindingSource
- Используемые свойства BindingSource
  - DataSource
  - DataMember

Реализация отношения «Мастер-детали»

- DataSource – используется BindingSource родительской таблицы
- DataMember – используется ВНЕШНИЙ КЛЮЧ родительской таблицы

Реализация отношения «Мастер-детали»



Реализация отношения при помощи программного кода

- DataRow
  - ChildRows
  - ParentRow
- BindingSource
  - CurrentChanged
- DataView
  - RowFilter