ОСНОВЫ БАЗ ДАННЫХ

Основы баз данных

- **База данных** – это совокупность данных, которая предназначена для машинной обработки и служит для удовлетворения нужд многих пользователей в рамках одной или нескольких организаций.
- **База данных** состоит из одной или нескольких таблиц: строки таблицы называются записями, столбцы – полями.
- Все записи таблицы разные.

Основы баз данных

- **Первичный ключ** – это одно или несколько полей таблиц, по которым можно однозначно найти запись.
- Чаще в качестве первичного ключа используют целое число, которое увеличивается при добавлении новой записи в таблицу.

Основы баз данных

- Совокупность таблиц, связанных между собой определенным образом, называется **реляционной базой данных** (РБД).
- Виды **связей** между таблицами БД:
  - Один к одному (1 : 1)
  - Один ко многим (1 : M)
  - Многие к одному (M : 1)
  - Многие ко многим (M : N)

Основы баз данных

- Основное достоинство РБД – повышение эффективности использования БД за счет
  - Применения специальных языков манипулирования данными (SQL)
  - Ликвидации избыточности представления информации

Основы баз данных

- **Локальные БД** – располагаются на одном компьютере вместе с обращающимся к ним приложением.
- Работа с такими БД производится обычно в однопользовательском режиме.
- **Локальная БД** может также работать в сети. В таком случае файлы БД и приложения располагаются на сервере, и при запуске этого приложения на компьютере пользователя запускается его копия.
- Такой принцип работы с БД соответствует архитектуре **файл-сервер**.

Основы баз данных

- **Удаленные** БД размещаются на сервере сети, а приложение, работающее с этой БД, располагается на компьютере пользователя, что соответствует архитектуре **клиент-сервер (двухуровневая архитектура)**.
- Клиентом является приложение пользователя, которое формирует запрос (на языке SQL) для получения данных и посылает его на удаленный сервер, где находится БД.
- При получении такого запроса, удаленный сервер отправляет его серверу БД (SQL-серверу).
- Сервер БД представляет собой программу, с помощью которой осуществляется управление удаленной БД и обеспечивает выдачу клиенту результатов выполнения поступившего запроса.
- Вся работа происходит непосредственно на удаленном сервере.

Модель «файл-север»

Файл-сервер

Функции:
физическое хранение данных

Большая нагрузка на сеть,
передаются данные

Клиенты

Функции:
интерфейс пользователя,
логика обработки,
управление данными

Двухуровневая модель «клиент-север»

Сервер баз данных

Функции:
физическое хранение данных,
логика обрабртки,
управление данными

Малая нагрузка на сеть,
передаются запросы и результаты

Клиенты

Функции:
интерфейс пользователя,
логика обработки

Основы баз данных

- В **двухуровневом** клиент-серверном приложении, как правило, все функции по формированию пользовательского интерфейса реализуются на клиенте, все функции по управлению данными - на сервере, а вот бизнес-правила можно реализовать как на сервере используя механизмы программирования сервера (хранимые процедуры, триггеры, представления и т.п.), так и на клиенте.
- В **трехуровневом** приложении появляется третий, промежуточный уровень, реализующий бизнес-правила, которые являются наиболее часто изменяемыми компонентами приложения
- Наличие не одного, а нескольких уровней позволяет гибко и с минимальными затратами адаптировать приложение к изменяющимся требованиям бизнеса.

Трехуровневая модель «клиент-север»

Функции:
физическое хранение данных,
управление данными

Функции:
логика обрабртки,
реализация бизнес-правил

Функции:
интерфейс пользователя

МЕХАНИЗМЫ ДОСТУПА К ДАННЫМ

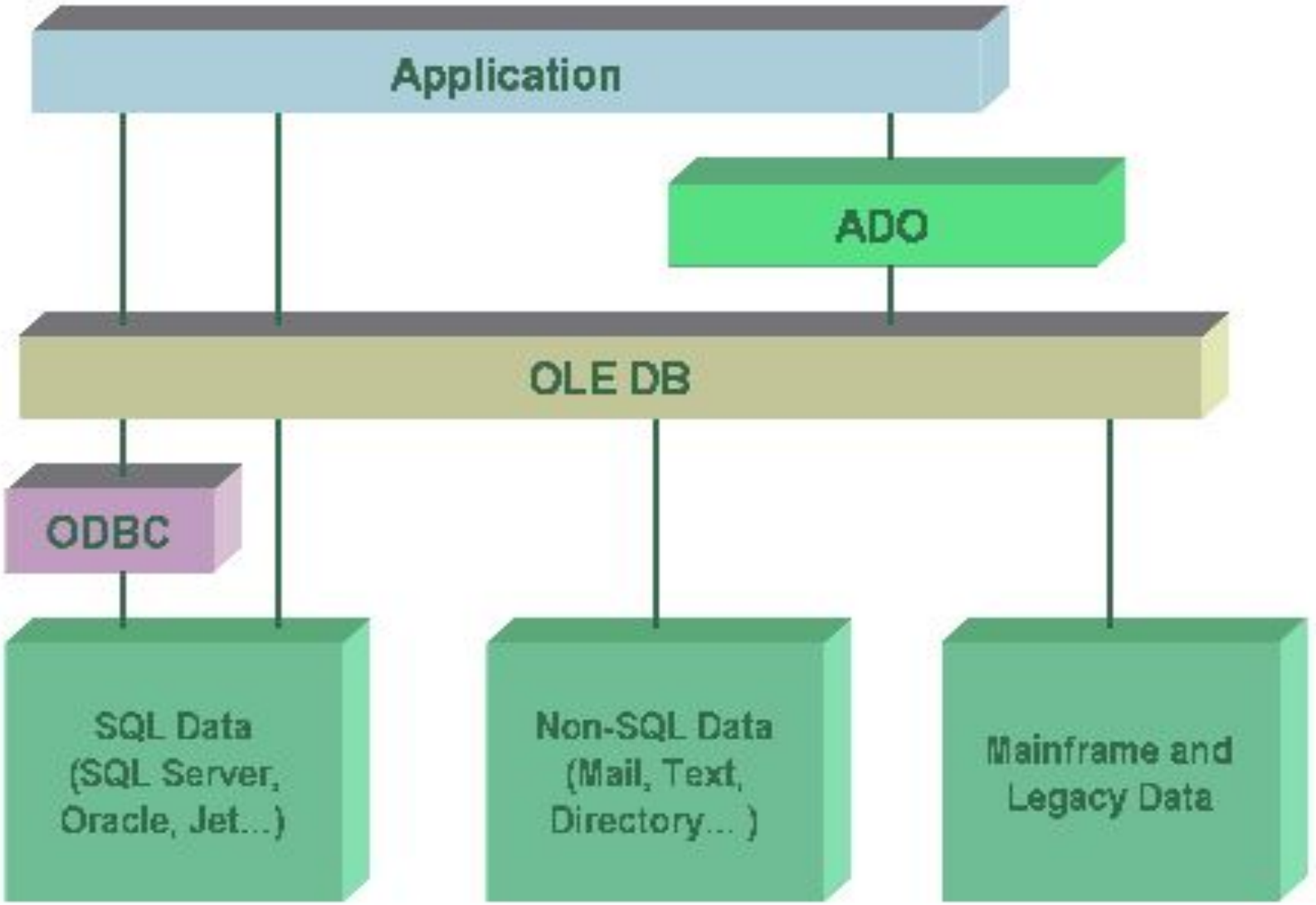
Механизмы доступа к данным

- Универсальный механизм доступа обычно реализован в виде библиотек и дополнительный модулей, называемых **драйверами** или **провайдерами**
  - *Библиотеки* содержат стандартный набор функций и классов для работы с данными
  - *Дополнительные модули*, специфичные для той или иной СУБД, реализуют непосредственное обращение к функциям клиентского API конкретных СУБД.

Механизмы доступа к данным

- Наиболее популярными среди универсальных механизмов доступа к данным являются следующие:
  - **ODBC** (Open DataBase Connectivity)
  - **OLE DB** (Object Linking and Embedding, DataBase)
  - **ADO** (ActiveX Data Objects)
  - **BDE** (Borland Database Engine)
- **ODBC, OLE DB, ADO** – (Microsoft) промышленные стандарты
- **BDE** – механизм, применяемый в средствах разработки фирмы Borland (устаревший механизм, применяется для обратной совместимости с ранее разработанными БД)

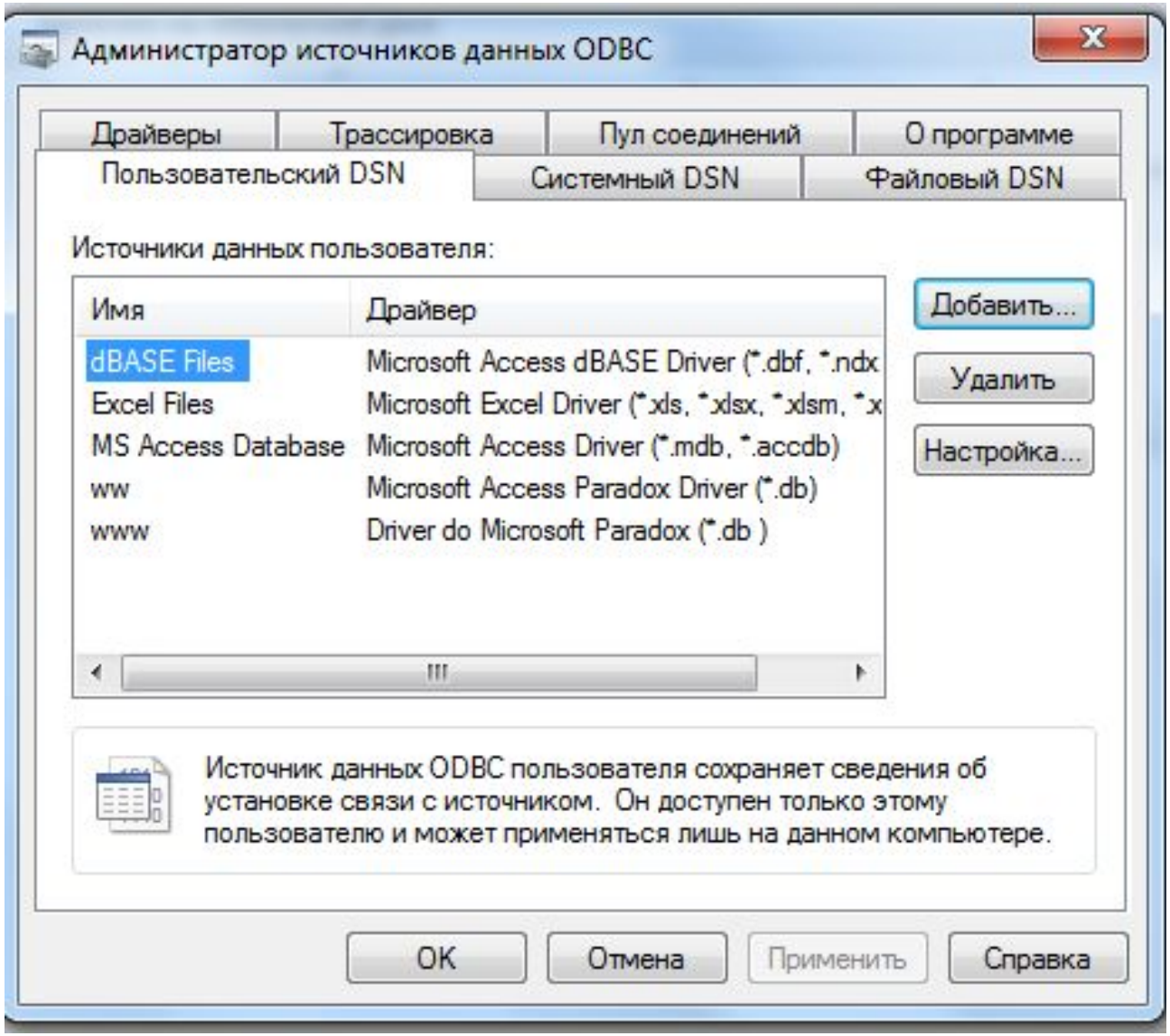
- **ODBC, OLE DB, ADO** – (Microsoft) промышленные стандарты



ODBC

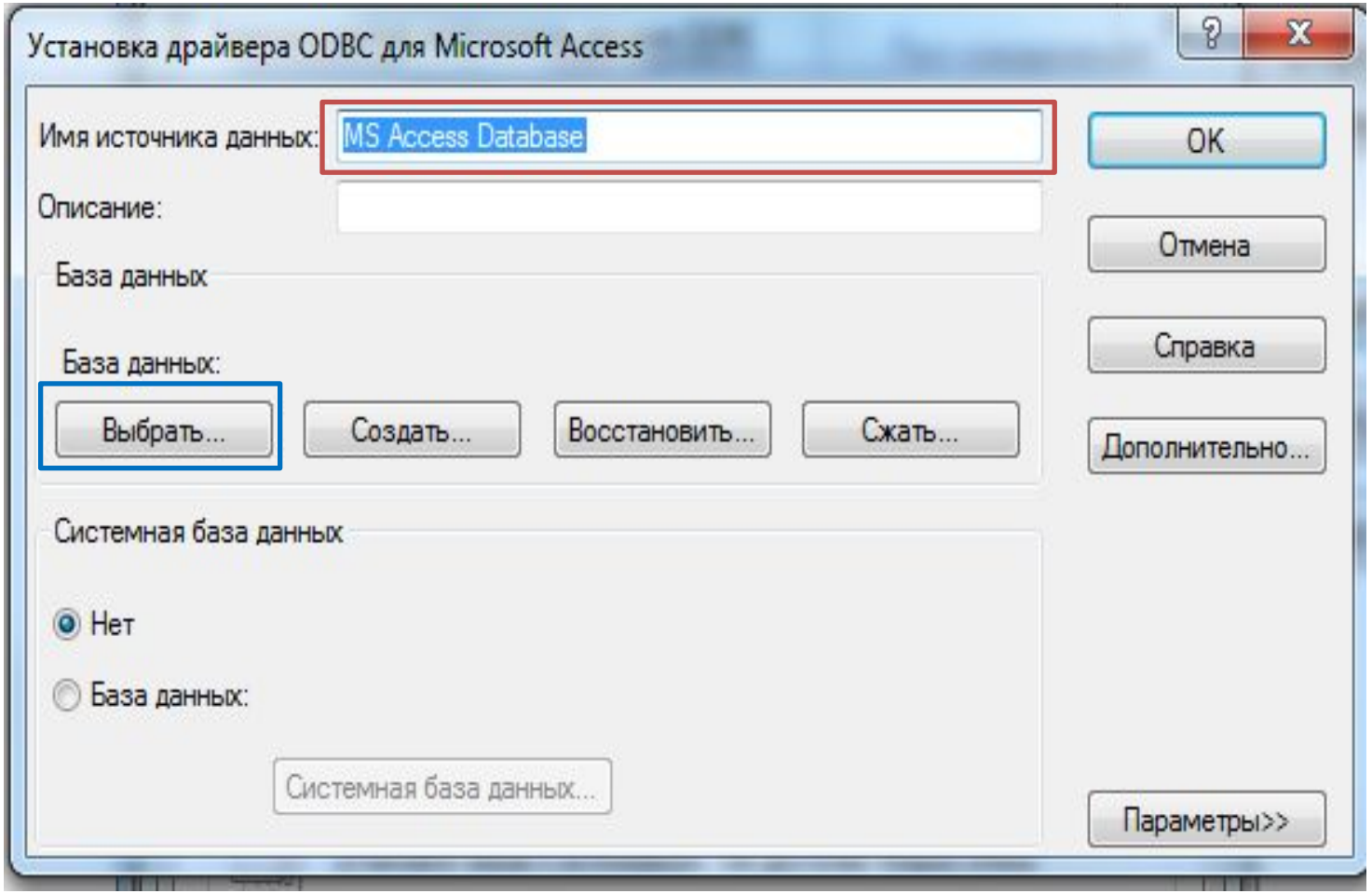
- Для доступа к конкретным СУБД через **ODBC** нужен ODBC-администратор – приложение, позволяющее определить:
  - Какие источники данных доступны для данного компьютера с помощью ODBC
  - **ODBC**-драйвер для доступа к этой БД
- Для каждой используемой СУБД нужен соответствующий ODBC-драйвер

ODBC


Администратор источников данных ODBC
Драйверы
Трассировка
Пул соединений
О программе
Пользовательский DSN
Системный DSN
Файловый DSN
Источники данных пользователя:
Имя
Драйвер
dBASE Files
Microsoft Access dBASE Driver (*.dbf, *.ndx
Excel Files
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.x
MS Access Database
Microsoft Access Driver (*.mdb, *.accdb)
ww
Microsoft Access Paradox Driver (*.db)
www
Driver do Microsoft Paradox (*.db)
Добавить...
Удалить
Настройка...
Источник данных ODBC пользователя сохраняет сведения об установке связи с источником. Он доступен только этому пользователю и может применяться лишь на данном компьютере.
OK
Отмена
Применить
Справка

ODBC

- Настройка источника данных ODBC

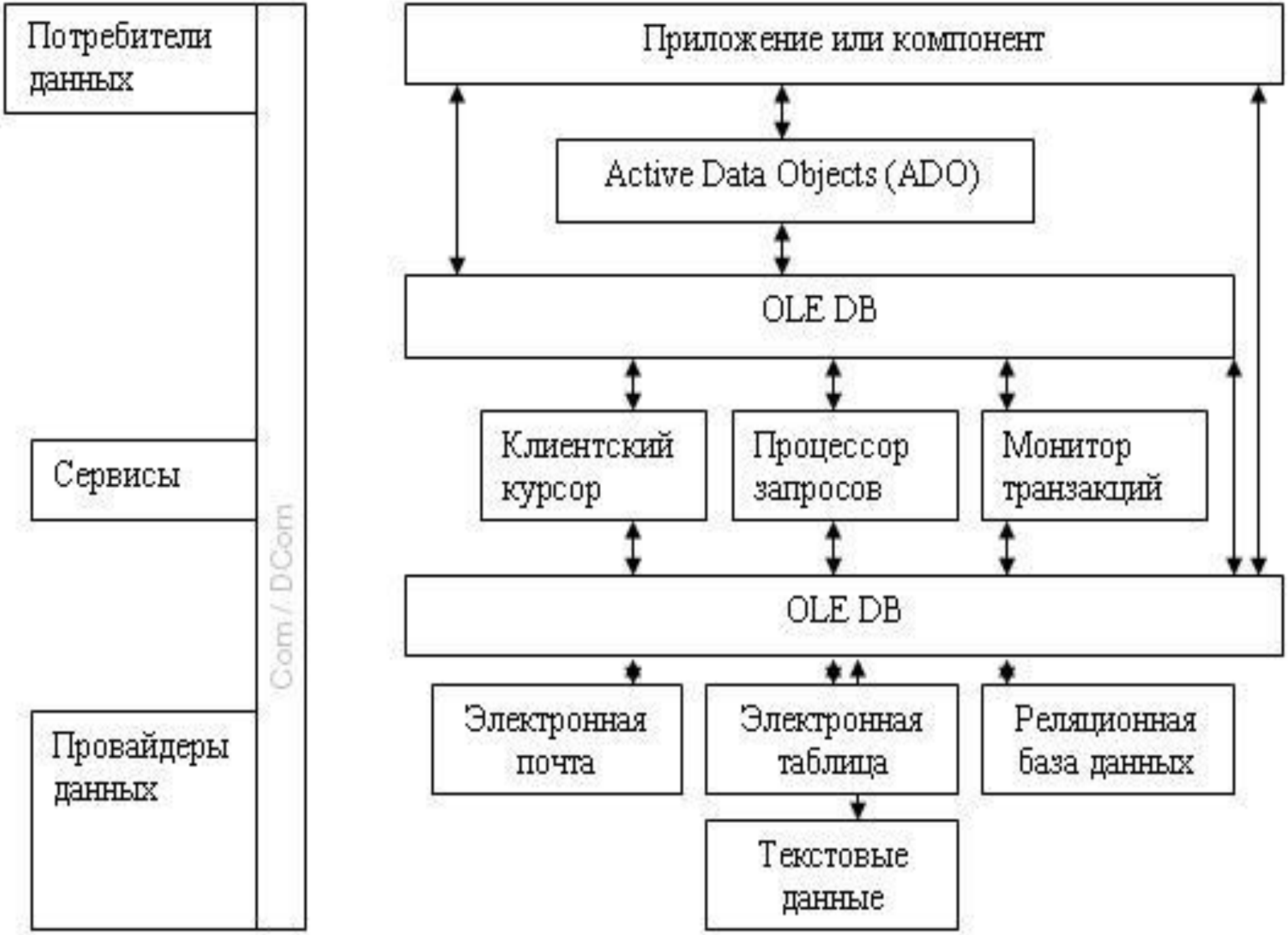


OLE DB

- В середине 1990-х годов, с развитием и распространением технологии *COM* (*Component Object Model*), компания *Microsoft* объявила о постепенном переходе от *ODBC* к использованию новой технологии *OLE DB.*
- **OLE DB** – представляет собой программный интерфейс для доступа к различным источникам данных: реляционные и нереляционные данные, текстовая и графическая информация и т.д.)
- Любой программный компонент, применяющий **OLE DB**, является потребителем

OLE DB

- **Потребители** могут обращаться к данным посредством ADO или применять OLE DB непосредственно, используя OLE-DB-провайдер.
- **Провайдер** – это часть ПО, в которой реализованы интерфейсы OLE DB.
- Существуют два типа провайдеров: **провайдеры данных** и **сервисные компоненты.**


Потребители данных
Сервисы
Провайдеры данных
COM / DCOM
Приложение или компонент
Active Data Objects (ADO)
OLE DB
Клиентский курсор
Процессор запросов
Монитор транзакций
OLE DB
Электронная почта
Электронная таблица
Реляционная база данных
Текстовые данные

OLE DB

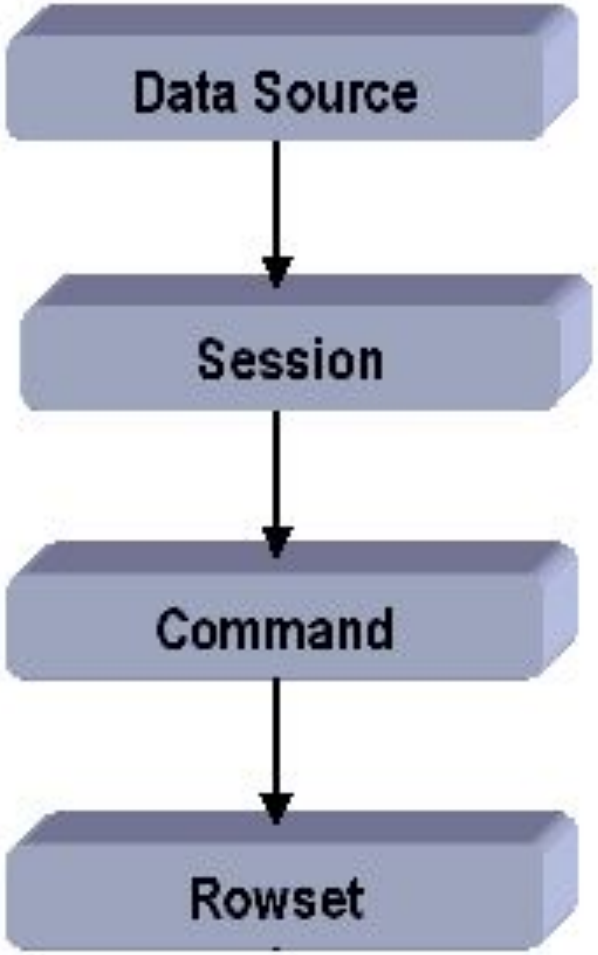
- **Провайдеры данных** – это компоненты ПО, манипулирующие данными.
- Они располагаются между потребителями данных и БД.
- Примеры:
  - Microsoft Jet 4.0 OLE DB Provider
  - Microsoft OLE DB Provider for ODBC Drivers
  - Microsoft OLE DB Provider for SQL Server
  - Microsoft OLE DB Provider for Oracle
  - и др

OLE DB

- **Сервисный компонент** (провайдер сервисов) – реализует расширенную функциональность, не поддерживаемую обычными провайдерами данных.
- Сервисный компонент может обращаться к хранилищу данных непосредственно или с помощью соответствующего провайдера данных – в этом случае провайдер сервисов является одновременно и поставщиком и потребителем.
- Примеры:
  - Microsoft Cursor Service for OLE DB
  - Microsoft Data Snapping Service for OLE DB

OLE DB

- Каждый OLE DB-провайдер должен содержать реализацию объектов:



OLE DB

Объект DataSource – предоставляет данные из источника данных потребителю; инкапсулирует информацию, связанную с соединением (включая имя пользователя и пароль).

Объект Session - предоставляет контекст для транзакций, может генерировать наборы данных (rowsets) из источников данных, а также команды для запросов к источнику данных.

Объект Command – используется для выполнения команд, представляющих собой строки, передаваемые от потребителя данных объекту Data Source для выполнения.

Объект Rowset (набор данных) позволяет OLE DB-провайдеру данных представлять данные из источников данных в табличном формате, также используется для

ADO

ADO

- *OLE DB* является интерфейсом системного уровня, который используется системными программистами.
- **ADO** – представляет собой высокоуровневый программный интерфейс для доступа к *OLE* DB-интерфейсам.
- **ADO** позволяет манипулировать данными с помощью любых OLE DB-провайдеров.
- **ADO** содержит набор объектов, используемых для соединения с источником данных, для чтения, добавления, удаления и модификации данных.
- Технология ADO является наиболее эффективной и универсальной.

ADO

- Для работы с *ADO* в *Delphi* предусмотрены компоненты, расположенные на странице *ADO* (в последних версиях на странице *dbGo*). Они инкапсулируют такие объекты *ADO* как ***Connection***, ***Command*** и ***RecordSet***.

ADO

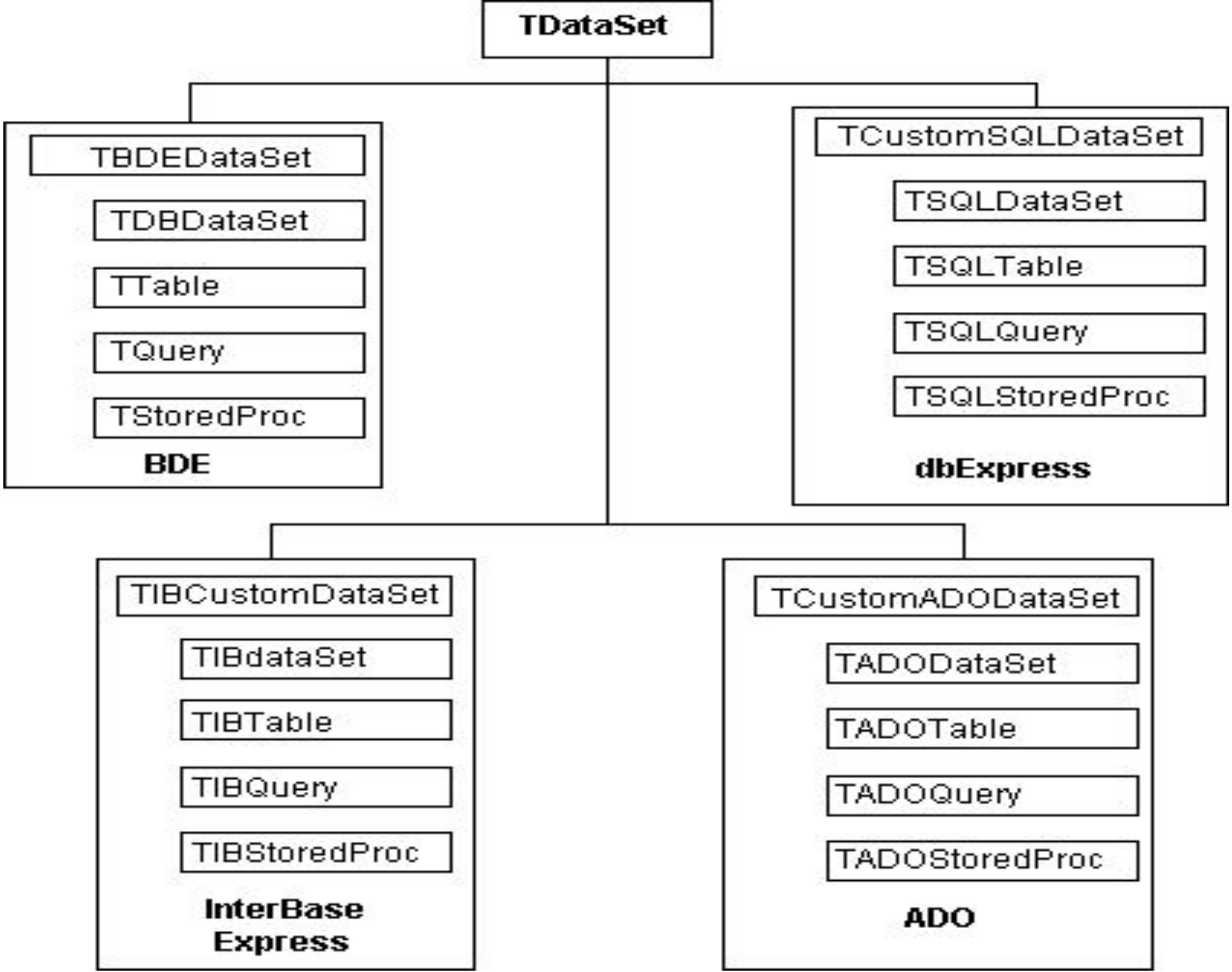
- Для связи с набором данных используются компоненты *ADO:*
  - ***ADOConnection*** – используется для соединения с различными источниками ADO,
  - ***ADODataSet*** – используется для выборки данных из одной или нескольких таблиц и доступа к ним посредством ADO,
  - ***ADOTable*** – используется для работы с одной таблицей в базе данных

ADO

- ***ADOQuery*** – служит для определения SQL-операторов, позволяющих осуществить доступ к одной или нескольким таблицам в БД,
- ***ADOStoredProc*** – предназначена для исполнения хранимых процедур БД,
- ***ADOCommand*** – обеспечивает выполнение команд, не возвращающих результаты

ADO

- Для **обмена** информацией между набором данных и компонентами визуализации и управления данными используется компонент ***DataSourse***.
- Для **визуализации и управления данными** применяются компоненты ***DBGrid***, ***DBText***, ***DBNavigator*** и др.

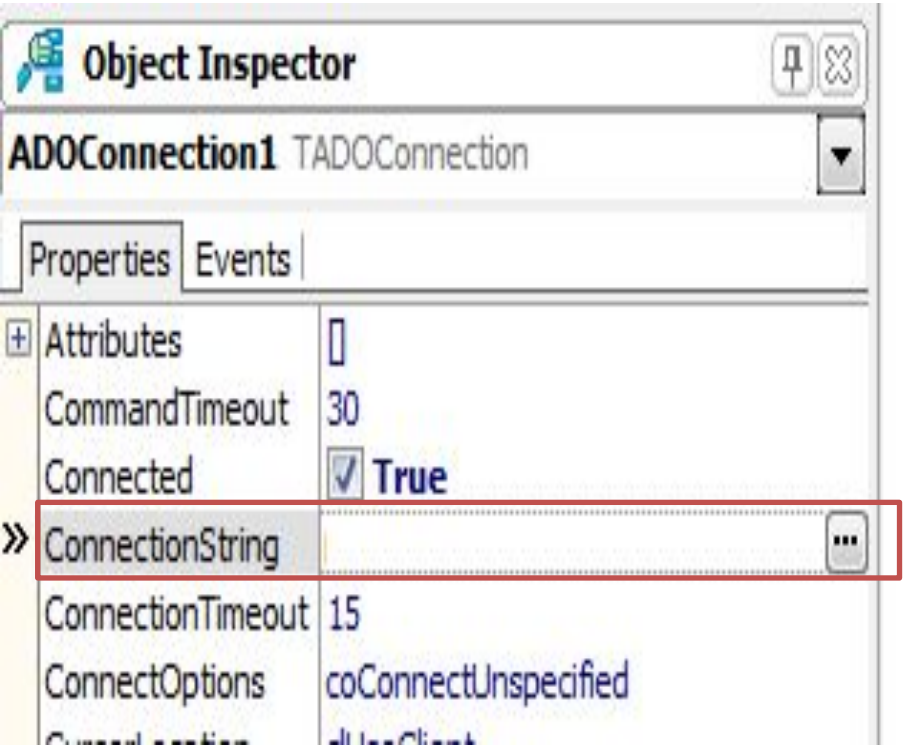

TDataSet
TBDEDataSet
TDBDataSet
TTable
TQuery
TStoredProc
BDE
TCustomSQLDataSet
TSQLDataSet
TSQLTable
TSQLQuery
TSQLStoredProc
dbExpress
TIBCustomDataSet
TIBdataSet
TIBTable
TIBQuery
TIBStoredProc
InterBase Express
TCustomADODataSet
TADODataSet
TADOTable
TADOQuery
TADOStoredProc
ADO

Компонент ADOConnection

Компонент ADOConnection

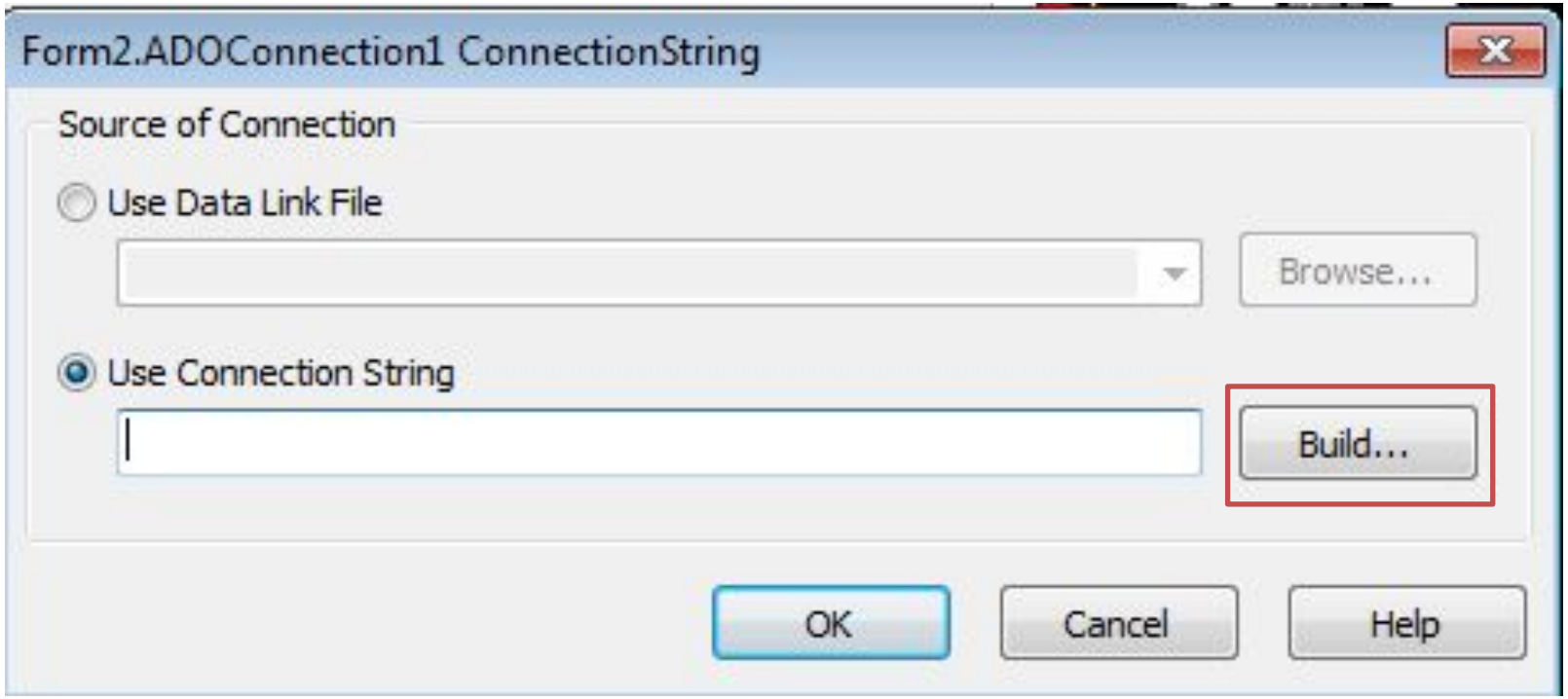
- Данный компонент связывается с набором данных *ADO*.
- Компонент *ADOConnection* позволяет настроить процедуру аутентификации, контролировать транзакции, напрямую выполнять команды, адресованные БД, кроме того, он позволяет сократить количество подключений, существующих в рамках приложения.
- Для использования данного компонента необходимо разместить его на форме и настроить его свойство **ConnectionString**.

Компонент ADOConnection

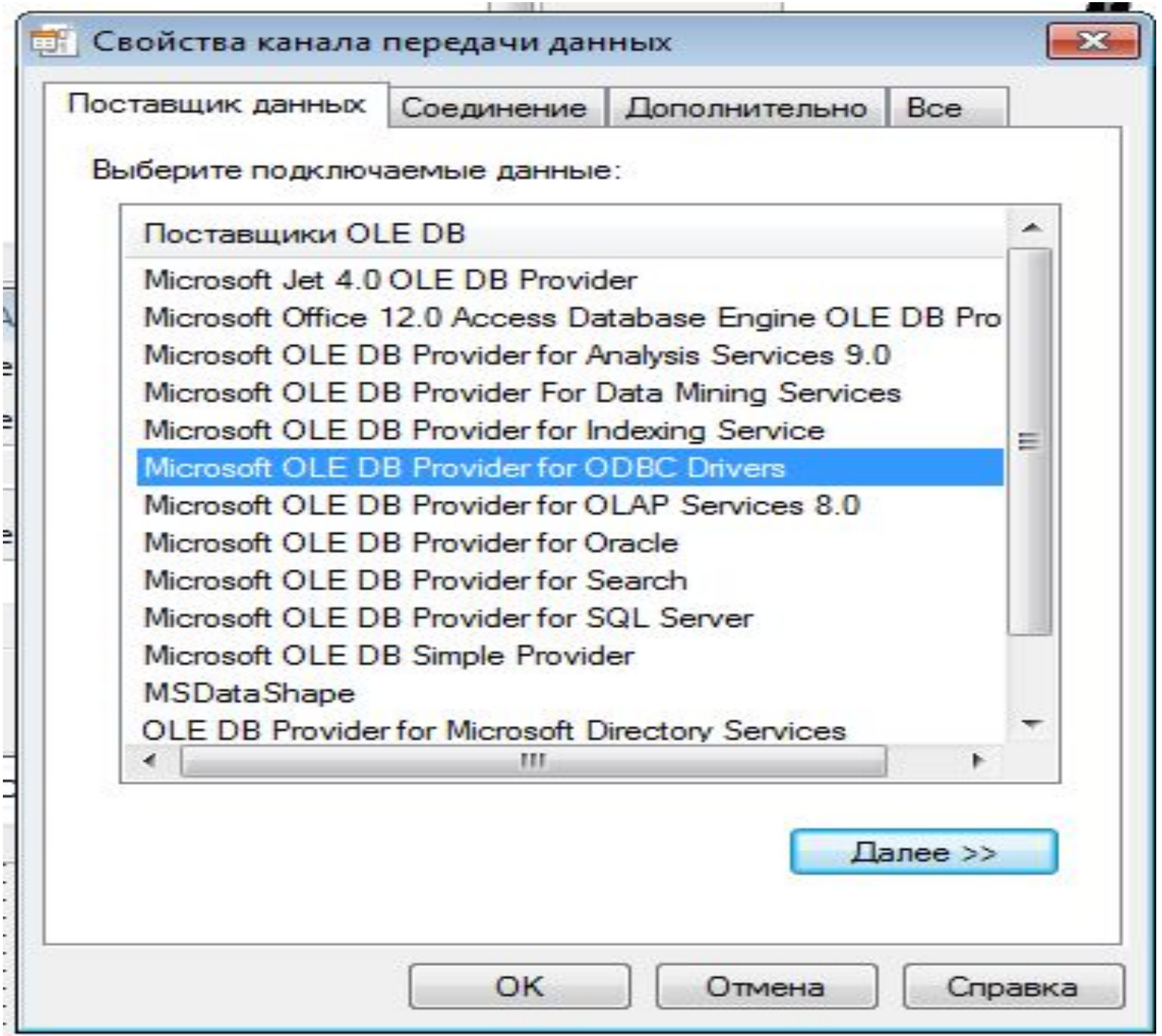


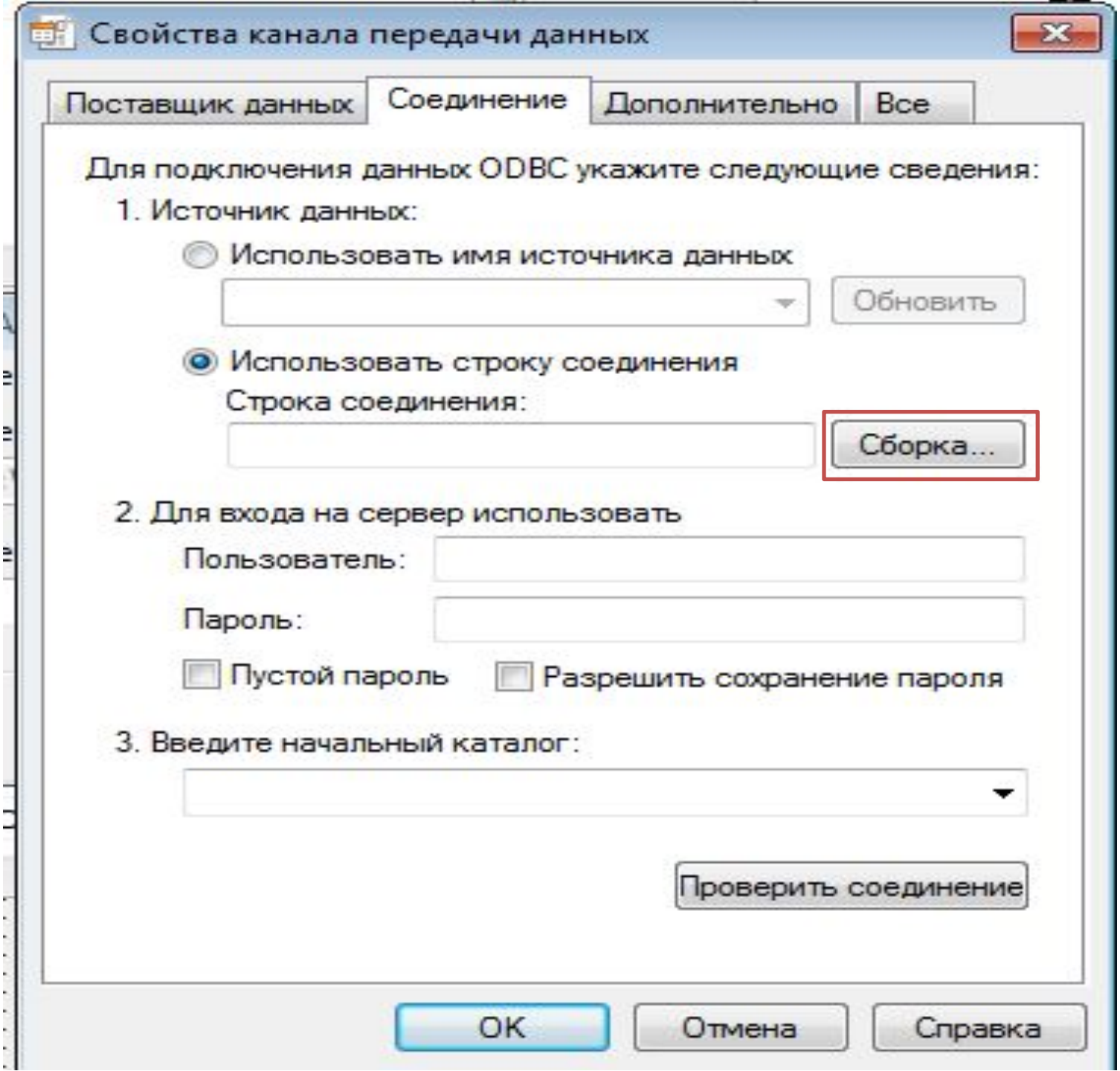
- свойство **ConnectionString** представляет собой строку, содержащую в себе несколько параметров соединения, которые разделяются друг от друга точками с запятой:
- ***Provider*** – имя провайдера, используемое для соединения,
- ***File name*** – имя файла, содержащего информацию о соединении,
- ***Remote provider*** – имя провайдера, используемое со стороны клиента,
- ***Remote Server*** – путь и имя сервера.

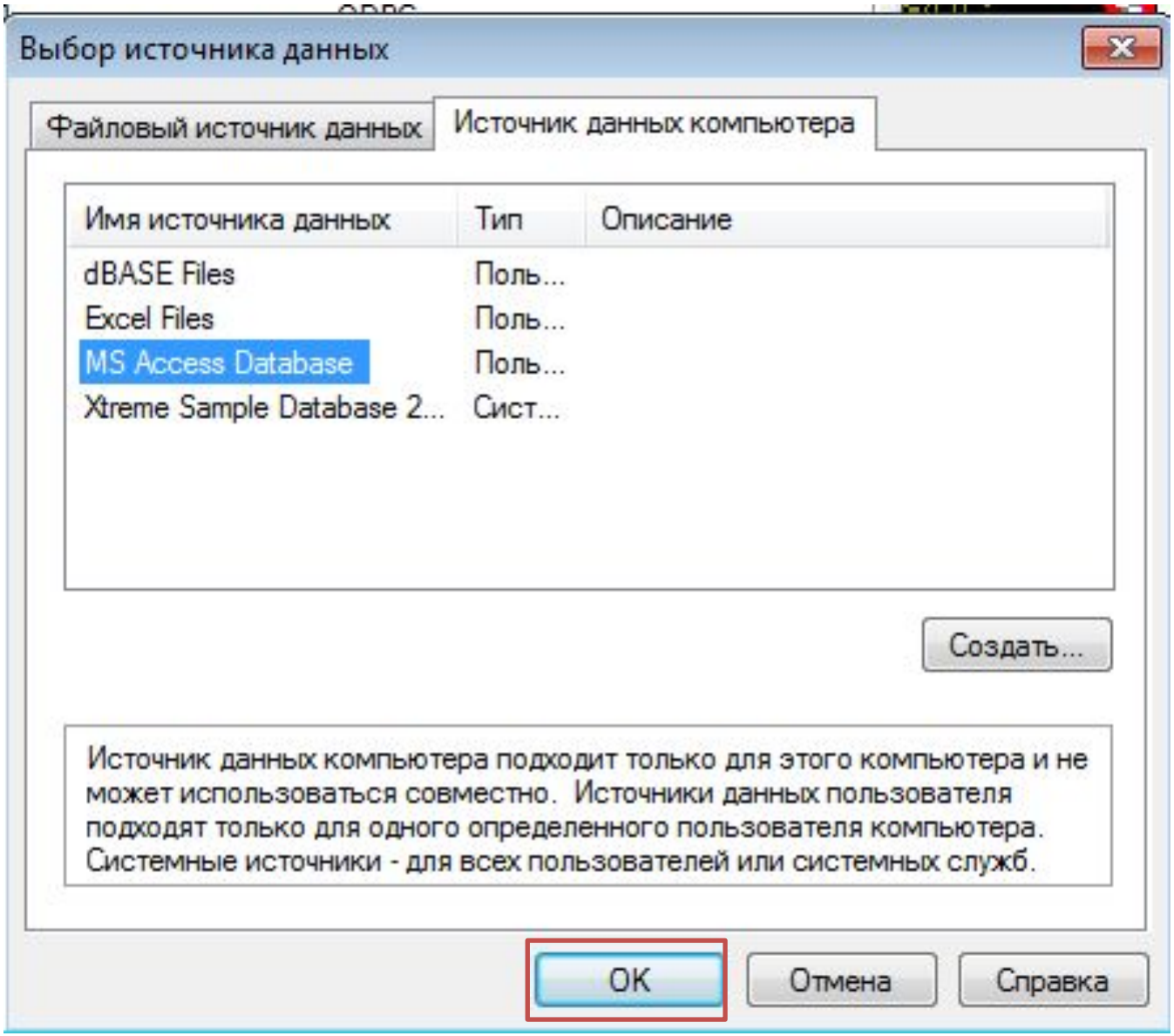
Компонент ADOConnection

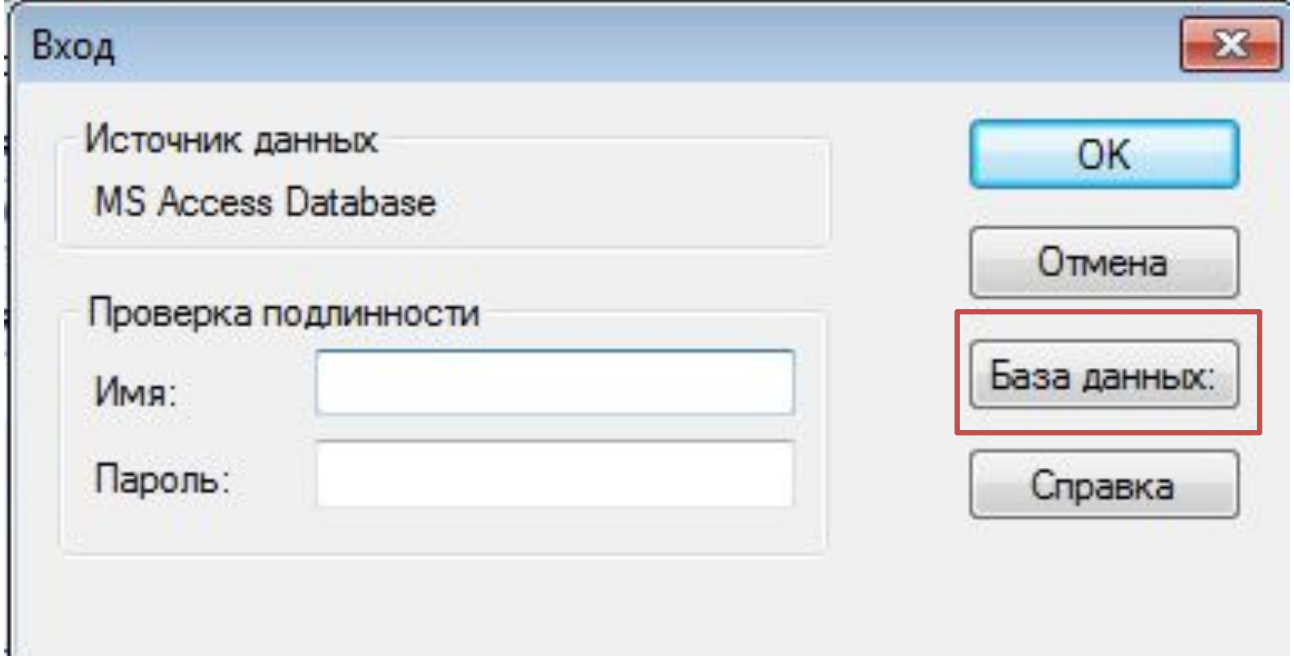


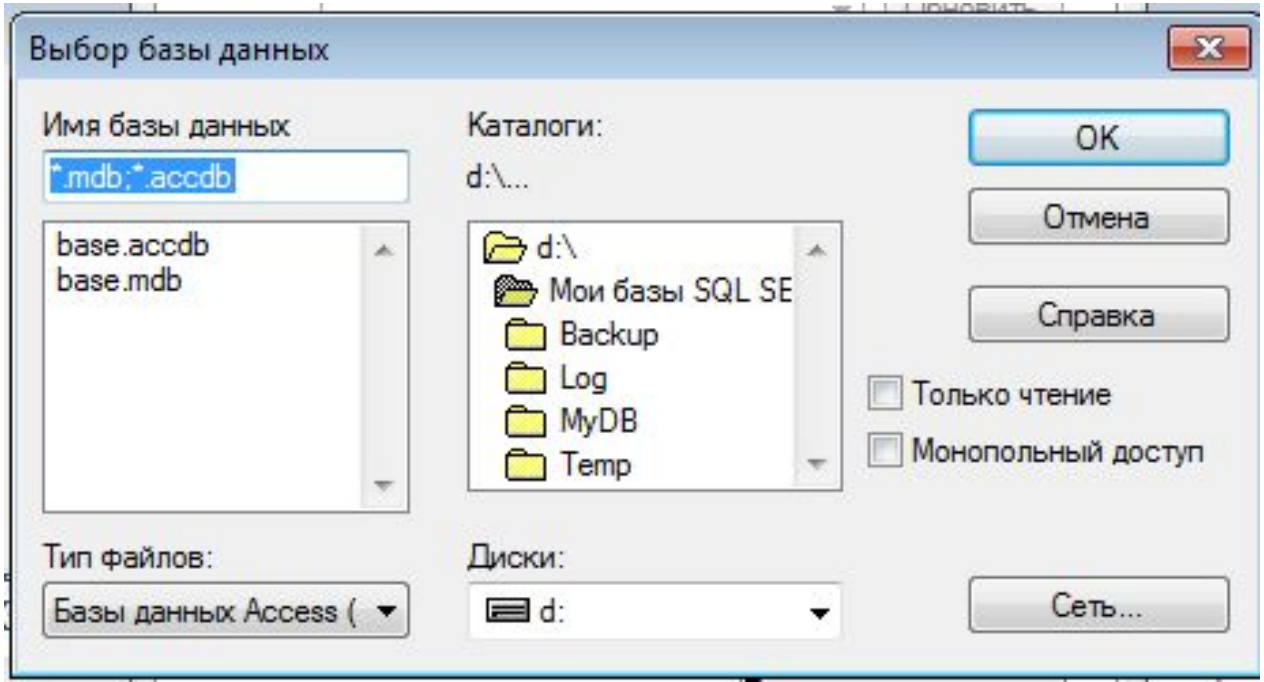
- При выборе первой опции необходимо указать *Data Link File* –файл связи с данными, содержащий информацию о подключаемой БД.
- Файл может иметь любое расширение, но обычно это файл с расширением *.*udl*.

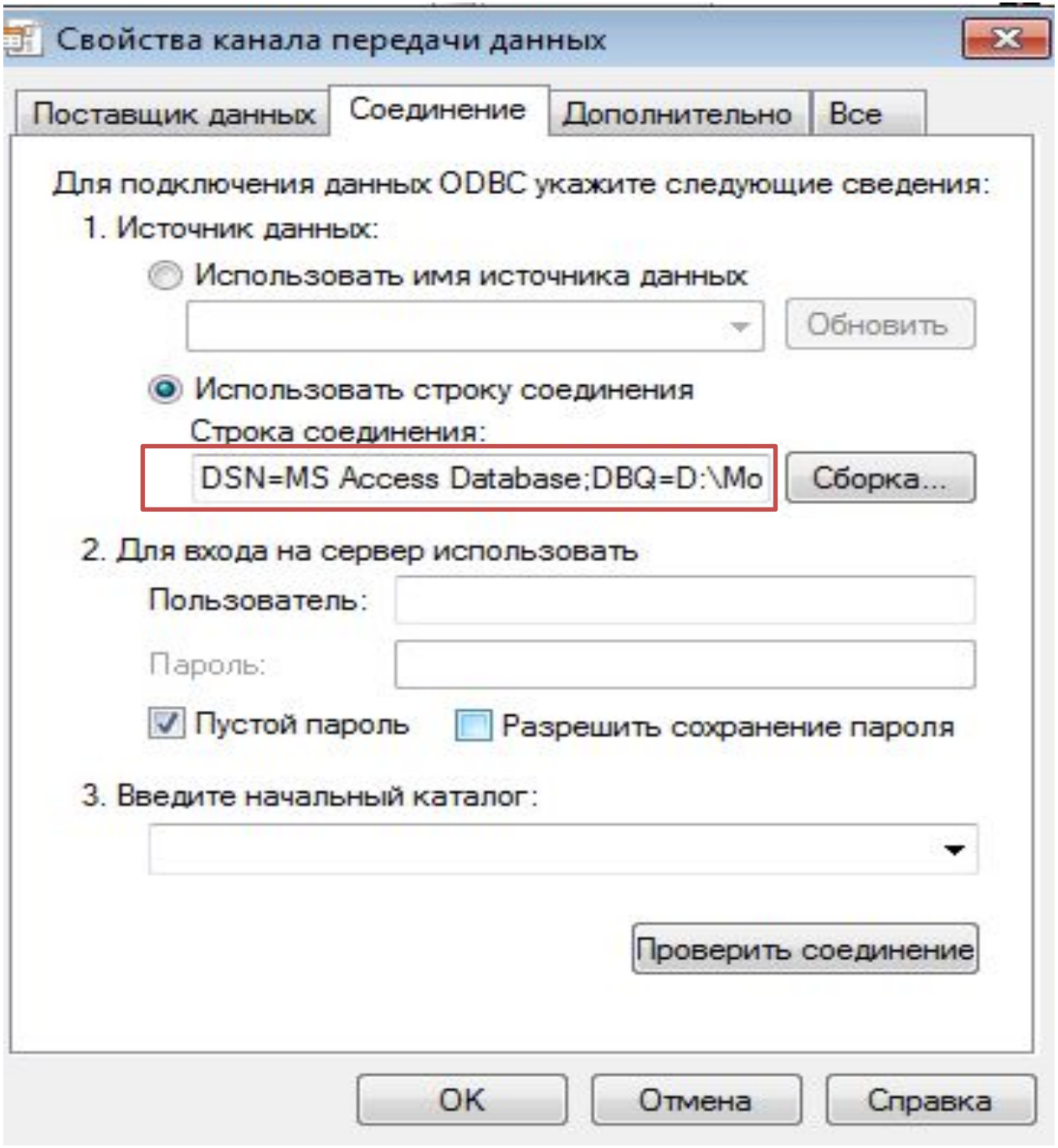

Свойства канала передачи данных
Поставщик данных
Соединение
Дополнительно
Все
Выберите подключаемые данные:
Поставщики OLE DB
Microsoft Jet 4.0 OLE DB Provider
Microsoft Office 12.0 Access Database Engine OLE DB Pro
Microsoft OLE DB Provider for Analysis Services 9.0
Microsoft OLE DB Provider For Data Mining Services
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for Search
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services
Далее >>
OK
Отмена
Справка


Свойства канала передачи данных
Поставщик данных
Соединение
Дополнительно
Все
Для подключения данных ODBC укажите следующие сведения:
1. Источник данных:
Использовать имя источника данных
Обновить
Использовать строку соединения
Строка соединения:
Сборка...
2. Для входа на сервер использовать
Пользователь:
Пароль:
Пустой пароль
Разрешить сохранение пароля
3. Введите начальный каталог:
Проверить соединение
OK
Отмена
Справка


Выбор источника данных
Файловый источник данных
Источник данных компьютера
Имя источника данных
Тип
Описание
dBASE Files
Поль...
Excel Files
Поль...
MS Access Database
Поль...
Xtreme Sample Database 2...
Сист...
Создать...
Источник данных компьютера подходит только для этого компьютера и не может использоваться совместно. Источники данных пользователя подходят только для одного определенного пользователя компьютера. Системные источники - для всех пользователей или системных служб.
OK
Отмена
Справка


Вход
Источник данных
MS Access Database
Проверка подлинности
Имя:
Пароль:
OK
Отмена
База данных:
Справка


Выбор базы данных
Имя базы данных
.mdb;.accdb
base.accdb
base.mdb
Каталоги:
d:\...
d:\
Мои базы SQL SE
Backup
Log
MyDB
Temp
Тип файлов:
Базы данных Access (
Диски:
d:
OK
Отмена
Справка
Только чтение
Монопольный доступ
Сеть...


Свойства канала передачи данных
Поставщик данных
Соединение
Дополнительно
Все
Для подключения данных ODBC укажите следующие сведения:
1. Источник данных:
Использовать имя источника данных
Обновить
Использовать строку соединения
Строка соединения:
DSN=MS Access Database;DBQ=D:\Mo
Сборка...
2. Для входа на сервер использовать
Пользователь:
Пароль:
Пустой пароль
Разрешить сохранение пароля
3. Введите начальный каталог:
Проверить соединение
OK
Отмена
Справка

Компонент ADOConnection

Строка соединения - **ConnectionString**

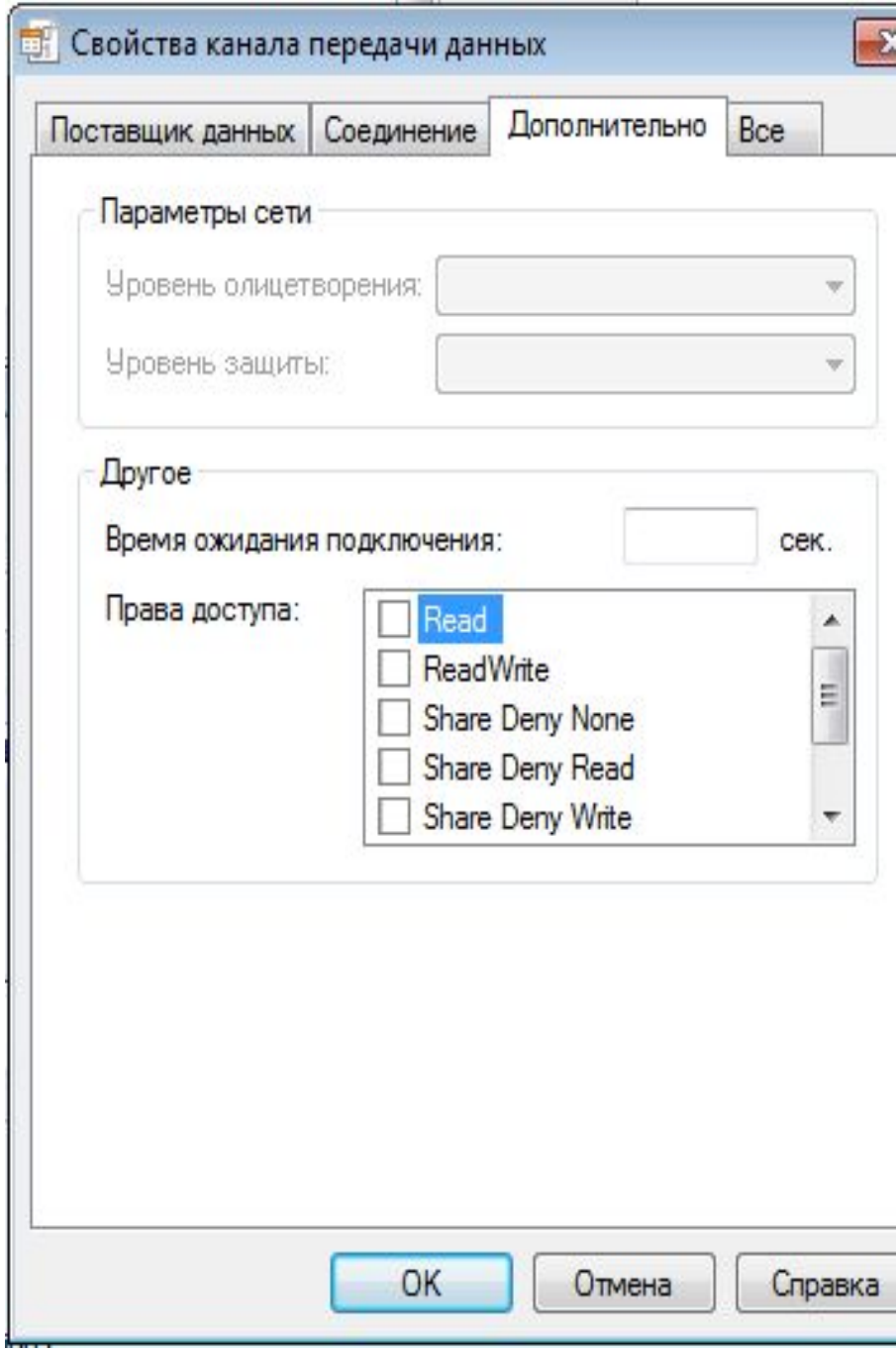
Provider=MSDASQL.1;

Persist Security Info=False;

Extended Properties=

"**DSN**=MS Access Database;**DBQ**=D:\Мои базы SQL SERVER\base.accdb;
DefaultDir=D:\Мои базы SQL SERVER;
DriverId=25; **FIL**=MS Access;
MaxBufferSize=2048; **PageTimeout**=5;
UID=admin;"

Read—только чтение;
ReadWrite — чтение и запись;
Share Deny None — режим совместной работы невозможен;
Share Deny Read — нельзя совместно использовать данные, открытые в режиме чтения;
Share Deny Write — нельзя совместно использовать данные, открытые в режиме записи;
Share Exclusive — нельзя совместно использовать данные, открытые в режиме чтения и/или записи;



Компонент ADOConnection

Соединение с БД

ADOConnection1.Open;

или

ADOConnection1.Connected := true;

Метод *Open*, содержащего два необязательных параметра: *UserID*– идентификатор пользователя, *Password* – пароль пользователя.

Закрытие соединения с БД

ADOConnection1.Close;

или

ADOConnection1.Connected := false;

Компонент ADOConnection

Свойства:

- ***LoginPromt*** *:Boolean* – определяет необходимость ввода пароля и идентификатора пользователя . Если *LoginPromt=false* – пароль можно не задавать ни в строке соединения, ни при вызове метода *Open*.
- свойство ***KeepConnection*** – используется для проверки соединения компонента *ADOConnection* с базой данных.

Компонент ADOConnection

- Соединение с базой данных может работать в **синхронном** или **асинхронном** режиме - свойство *ConnectOptions*:
  - *ConnectOptions=coConnectUnspecified* в синхронном режиме,
  - *ConnectOptions=coAsyncConnect* в асинхронном режиме.
- **Синхронное соединение** всегда ожидает результат последнего запроса, **асинхронное соединение** может выполнять новые запросы, не дожидаясь ответа от предыдущих

Работа с БД

- Для создания приложения БД через механизм ADO потребуются 3 группы компонентов:
  - Компоненты – наборы данных (**вкладка ADO / dbGo**)
  - Компоненты доступа к данным (**вкладка Data Access**)
  - Компоненты управления данными (**вкладка Data Controls**)

КОМПОНЕНТЫ – НАБОРЫ ДАННЫХ

Компоненты ADOTable и ADOQuery

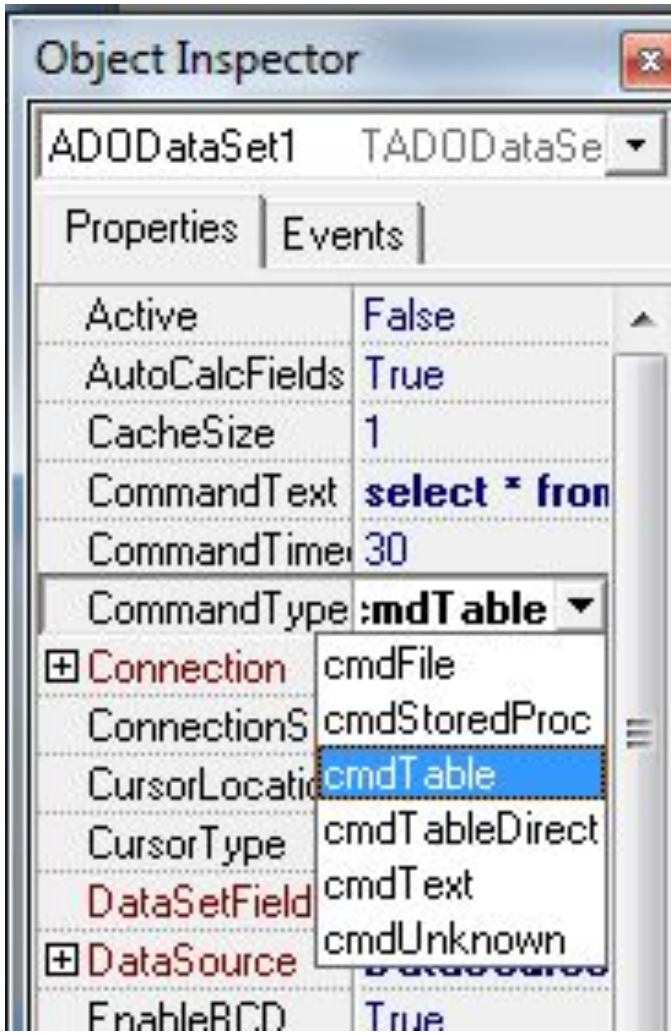
- **Компонент *ADOTable:***
  - обеспечивает доступ к одной таблице базы данных.
  - основное свойство **TableName** с помощью которого задается имя таблицы.
- **Компонент *ADOQuery:***
  - используется для выполнения запросов на языке *SQL*, позволяет получать данные из одной или нескольких таблиц БД .
  - основное свойство **SQL**, которое содержит текст запроса

Компонент ADODataset

- **Компонент *ADODataSet*** используется для выборки данных из одной или нескольких таблиц и доступа к ним посредством *ADO*.
- С помощью этого компонента можно:
  - получить все данные из таблицы,
  - установить фильтры для того, чтобы выбрать ту информацию, которая отвечает некоторым условиям,
  - выполнять *SQL*-запросы,
  - запускать системные и определенные пользователем хранимые процедуры,
  - а также сохранять наборы данных в файле и загружать их.

Компонент ADODataset

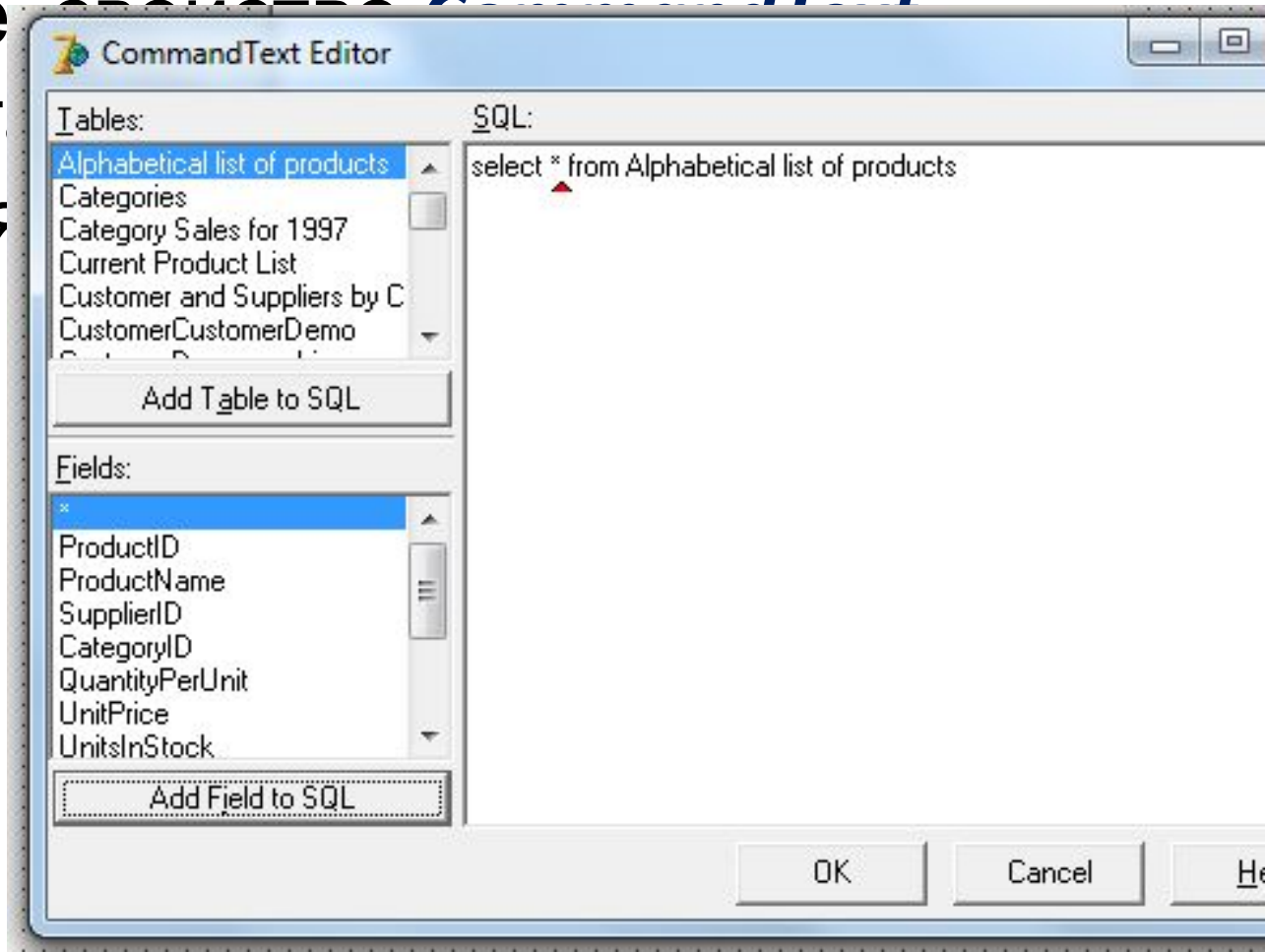
- Способ работы компонента определяется свойством ***CommandType***.
- В зависимости от выбранного типа выставляется соответствующее значение свойства ***CommandText***



Например, если выбран тип **cmdTable**, то в свойстве ***CommandText*** указывается

Компонент ADODataset

- Для задания запроса, используется тип **cmdText**.
- В этом случае свойство *CommandText* является сост
  редактируется
  посредством
  редактора
  CommandText
  Editor



КОМПОНЕНТЫ ДОСТУПА К ДАННЫМ

Компонент **DataSource**

Вкладка Data Access

- Компонент **DataSource** представляет собой «мост» между визуальными компонентами интерфейса БД и компонентами TDataSet.
- Необходим для работы практически любого из компонентов интерфейса БД.
- Позволяет устанавливать некоторые параметры НД, устанавливать состояние НД, отслеживать изменения НД

КОМПОНЕНТЫ УПРАВЛЕНИЯ ДАННЫМИ (DATA CONTROLS)

Компоненты управления данными

Компонент	Описание
DBGrid	Отображает информацию БД в виде таблицы. Позволяет редактировать выбранное поле активной записи.
DBNavigator 	Перемещает курсор по записям компонента TDataSet (обычно это таблица БД или результат выполнения запроса к РБД). Дает возможность переводить выбранную запись в режим редактирования, добавления новой записи и т.д, а также позволяет сохранять изменения набора данных
DBText	Выводит одно поле источника данных

Компоненты управления данными

Компонент	Описание
DBEdit	Отображает и дает возможность редактировать одно поле
DBMemo	Отображает и позволяет редактировать поля типа MEMO с помощью простого многострочного окна редактирования, оснащенного линейкой прокрутки текста
DBImage	Позволяет отображать и редактировать графические рисунки или BLOB-поля
DBListBox	Отображает список возможных значений, которые принимает

Компоненты управления данными

Компонент	Описание
DBComboBox	Отображает выпадающий список значений выбранного поля
DBCheckBox	Отображает и устанавливает булевское значение поля
DBRadioGroup	Позволяет установить для поля одно из нескольких значений

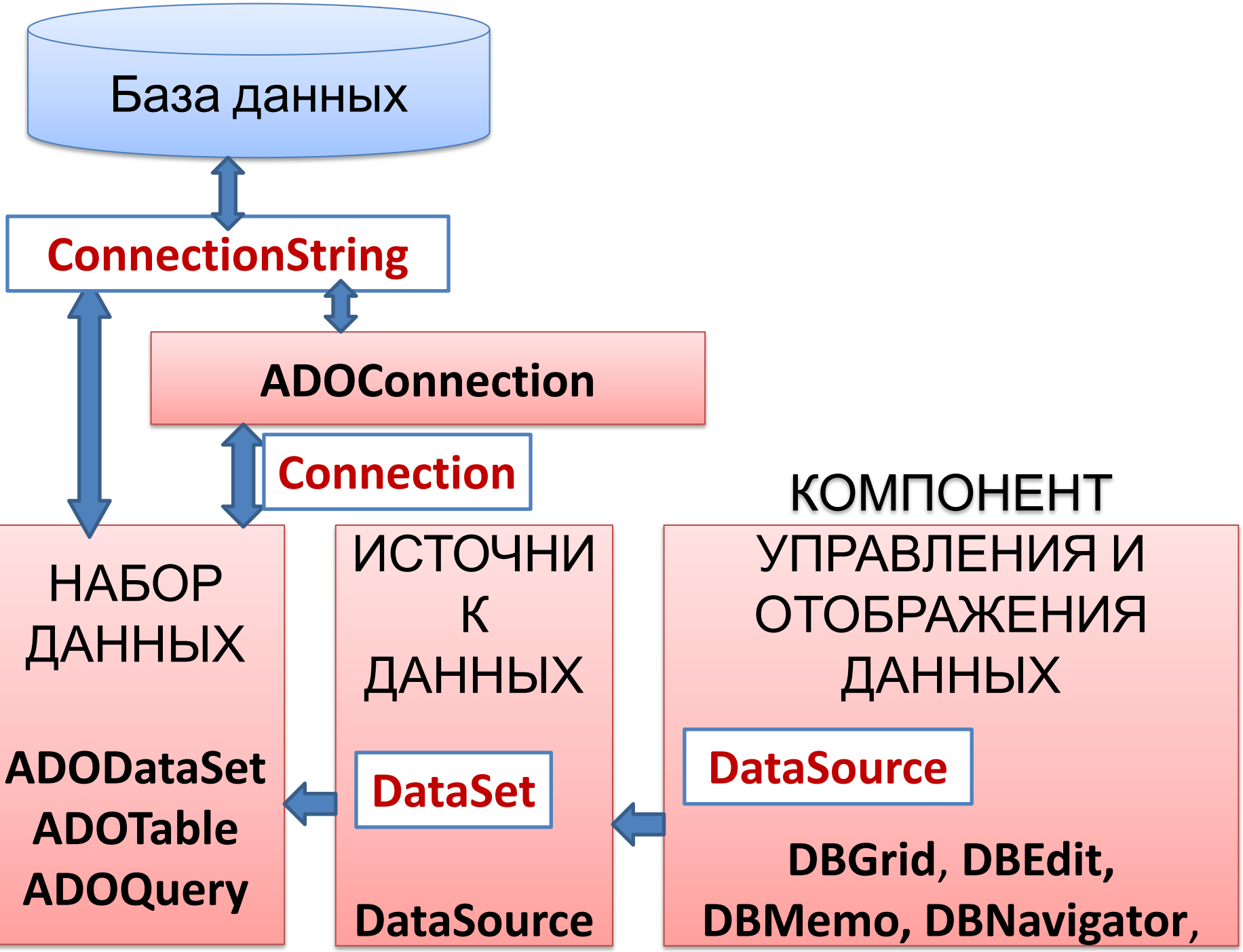
Компоненты управления данными

Компонент	Описание
DBLookupListBox	Позволяет отображать дополнительную информацию из другого источника данных, который связан с данным значением поля. Например, при перемещении по таблице «Студенты» позволяет видеть их оценки, размещенные в другой таблице
DBLookupComboBox	Представляет собой комбинацию компонентов TDBLookupListBox и TDBComboBox.

Компоненты управления данными

Компонент	Описание
DBRichEdit	Позволяет отображать и редактировать MEMO-поля с использованием различных шрифтов
DBCtrlGrid	Позволяет выводить записи из источника данных, располагая каждую запись на собственной панели
DBChart	Позволяет отображать данные из компонента TDataSet в виде графиков различного формата

ОТОБРАЖЕНИЕ ДАННЫХ ИЗ НАБОРОВ


База данных
ConnectionString
ADOConnection
Connection
НАБОР ДАННЫХ
ADODataSet
ADOTable
ADOQuery
ИСТОЧНИК ДАННЫХ
DataSet
DataSource
КОМПОНЕНТ УПРАВЛЕНИЯ И ОТОБРАЖЕНИЯ ДАННЫХ
DataSource
DBGrid, DBEdit,
DBMemo, DBNavigator,

Компоненты - наборы

Свойства

- **Active**:boolean; – свойство определяет открыт ли набор
- **CurrentRecord** – номер текущей записи
- **DataSource** – свойство, указывающее родительскую таблицу (для таблиц, связанных отношением родитель-потомок)
- **BOF** – определяет, находится ли курсор в первой записи набора
- **EOF** – определяет, достигнут ли конец набора

Компоненты - наборы

- **State** – текущее состояние набора данных. Возможные состояния наборов:
  - **dsInactive** – набор закрыт
  - **dsBrowse** – режим просмотра данных
  - **dsEdit** – режим редактирования активной записи
  - **dsInsert** – режим добавления данных
  - **dsSetKey** – просмотр ограниченного множества записей или поиск записи
  - **dsCalcFields**, **dsFilter** – выполняются обработчики *onCalcFields* или *onFilterRecord*, соответственно

Компоненты - наборы

- Отслеживать состояние набора данных удобно через событие компонента **DataSource** - **OnStateChange**

```
procedure TForm1.DataSource1StateChange
                                    (Sender:TObject);
var s:string;
begin
   case ADOTable1.State of
     dsInactive : s := 'Inactive';
     dsEdit     : s := 'Edit';
     dsBrowse   : s := 'Browse';
     dsInsert   : s := 'Insert';
   else
      s := 'other';
   end;
  Label1.Caption := s;   end;
```

В результате в тексте метке выводится информация о текущем состоянии набора данных

Компоненты - наборы

Свойства

- **Modified** – свойство определяет, была ли изменена активная запись
- **RecordCount** – общее число записей в наборе

Методы

- **Open** – открыть набор данных
- **Close** – закрыть набор данных

Открытие / закрытие наборов

Открытие наборов данных

– ADOTable1.**Active** := true;

или

– ADOTable1.**Open**;

Закрытие наборов данных

– ADOTable1.**Active** := false;

или

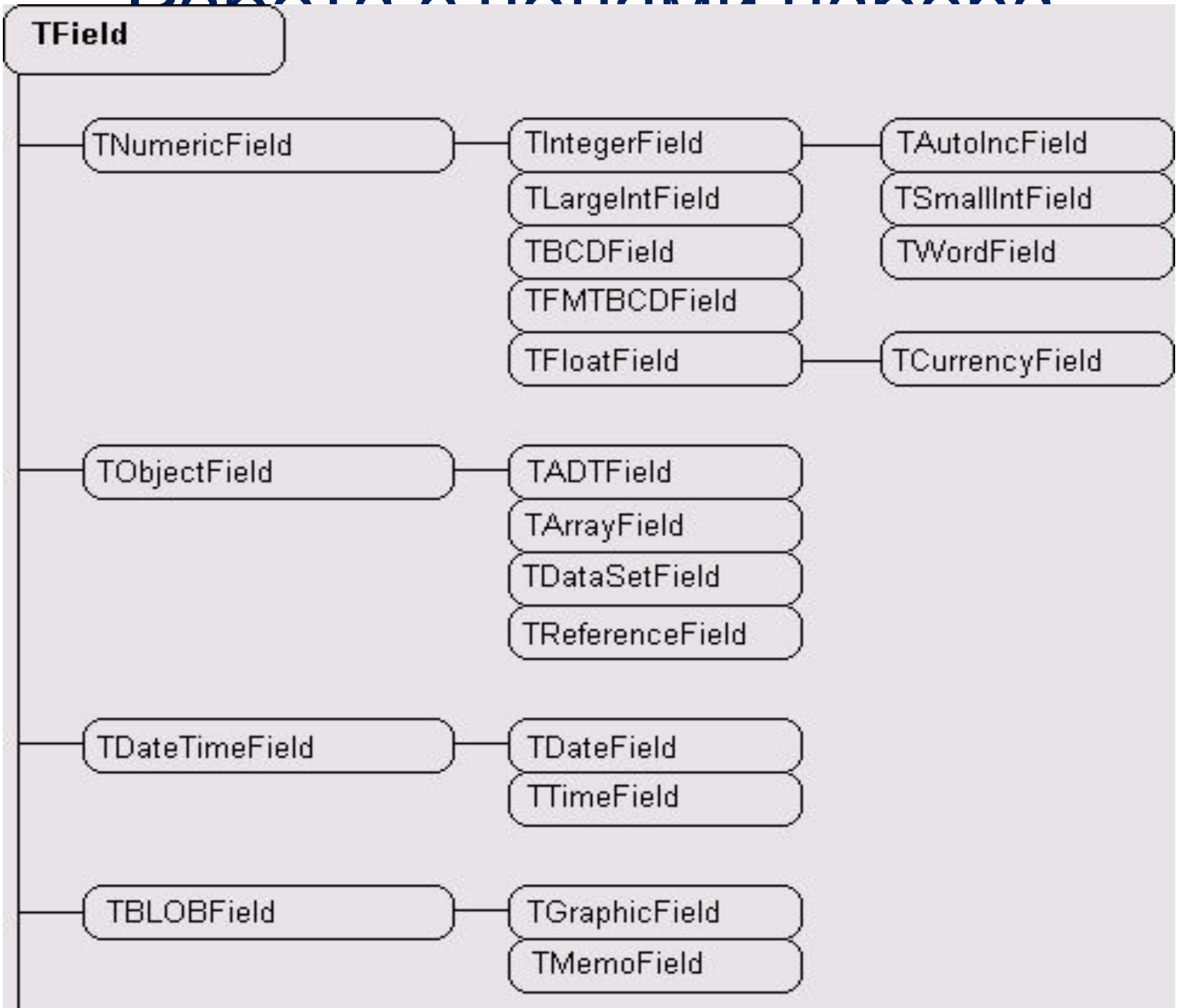
– ADOTable1.**Close**;

ПОЛЯ НАБОРОВ ДАННЫХ

Работа с полями набора данных

- Любой НД содержит как минимум одно поле.
- В Delphi каждому полю набора данных приложения соответствует собственный объект.
- Основой объектов полей является **класс TField**, который инкапсулирует основные свойства абстрактного поля, не зависящего от типа данных.
- От класса **TField** порождены другие классы, обеспечивающие функционирование реальных объектов полей, зависящих от

Работа с полями набора



Работа с полями набора данных

Свойства класса TField

- **Fields** – массив полей набора данных (нумерация полей с 0).
- **FieldCount** – количество полей набора данных
- **FieldName** – имя поля
- **Value** – значение поля
- **FieldByName** (имя_поля):**TField**– доступ к полю по имени
- **FieldValues** [имя_поля]:**Variant**– текущее

Работа с полями набора данных

Пример: Записать в список имена полей таблицы

```
var
S: String;
begin
   ComboBox1.Items.BeginUpdate;
   ComboBox1.Items.Clear;
   for i:=0 to ADOTable1.FieldCount-1 do
      begin
        S :=ADOTable1.Fields[i].FieldName;
        ComboBox1.Items.Add(s);
      end;
    ComboBox1.Items.EndUpdate;
end;
```

Работа с полями набора данных

Свойства приведения типов полей

- **AsString** – значение поля как строка
- **AsInteger** – значение поля как целое число
- **AsDateTime** – значение поля как дата/время
- **AsBoolean** – значение поля **true** или **false**
- **AsSingle** – значение поля как веществ. число
- **AsFloat** –как веществ.число двойной точности
- и др.

Работа с полями набора данных

Массив **Fields** предоставляет доступ к полям по номеру.

При этом поля нумеруются в порядке следования при их объявлении

Пример: Считать значение полей активной записи

```
var
  S: String; x, y:integer;
  ...
    S :=ADOTable1.Fields[0].asString;
    X :=ADOTable1.Fields[1].asInteger;
    y := ADOTable1.Fields[2].Value;
     ...
```

Работа с полями набора данных

Для доступа к полям по имени, используются свойства **FieldByName** или **FieldValues** (обращение по умолчанию)

Пример: Считать значения полей активной записи в переменные

```
var
  S: String; x,x1:integer;
  ...
S:=ADOTable1.FieldByName('CustName').asString;

X := ADOTable1.FieldValues['CustNo'];
    ...
X1 := ADOTable1['CustNo'];
```

Работа с полями набора данных

- Имена объектов поля формируются путем объединения имени набора данных и имени поля.
- Например, если в таблице ADOTable1 имеется поле CustName, то объект класса TField данного поля получит имя **ADOTable1CustNo**

Пример: Считать значения поля CustNo:

Способы:

```
S:=ADOTable1.FieldByName('CustNo').asInteger;
X := ADOTable1.FieldValues['CustNo'];
X1 := ADOTable1['CustNo'];

X2 := ADOTable1CustNo.asInteger;
```

Работа с полями набора данных

Свойства класса TField

- **DisplayLabel** – имя поля при выводе
- **DisplayWidth** – ширина поля при выводе
- **isNull** – определяет содержит ли поле NULL-значение в текущей записи
- **Required** – является ли поле обязательным к заполнению
- **CanModify** – определяет, может ли поле изменяться
- и др

Работа с полями набора данных

Пример : Для поля **CustNo** задать название «ИД заказчика», а ширину поля вывода - 30:

```
ADOTable1.FieldByName('CustNo').DisplayLabel :=
«ИД заказчика» ;

ADOTable1CustNo.DisplayWidth = 30;
```

Работа с полями набора данных

Пример 2: Если значение поля **CustName** не заполнено, выдать соответствующее сообщение:

```
If ADOTable1.FieldByName('CustName').isNull then
    ShowMessage('Поле не заполнено!');
```

```
If ADOTable1CustName.isNull then
    ShowMessage('Поле не заполнено!');
```

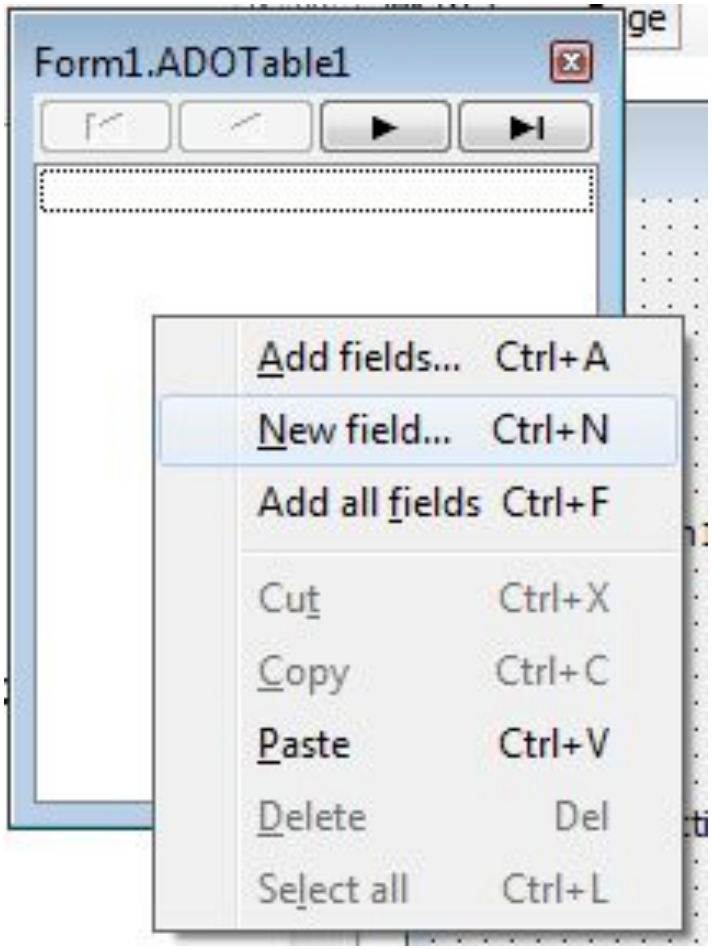
РЕДАКТИРОВАНИЕ СПИСКА ПОЛЕЙ

Работа с полями набора данных

- В Delphi предусмотрено два способа создания объектов полей: статические и динамические поля.
  - **Динамические** объекты полей автоматически создаются при открытии набора данных в соответствии со структурой связанной таблицы БД.
  - **Статические** объекты полей создаются программистом на этапе разработки при помощи специализированного Редактора полей, их свойства доступны в Инспекторе объектов.

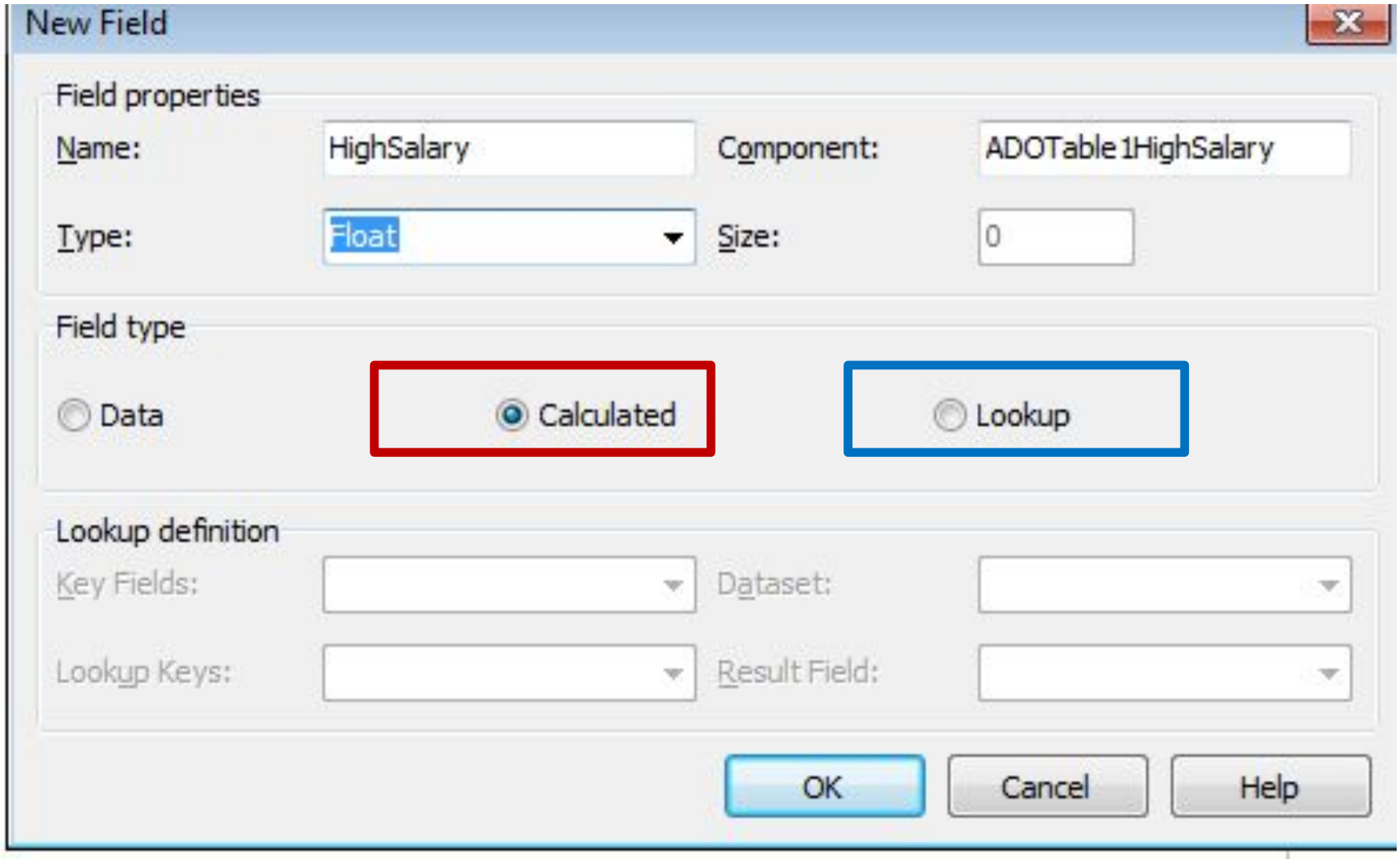
Работа с полями набора данных

- Для редактирования состава полей используется специальный **редактор полей**, который открывается двойным щелчком по объекту-набору данных или через контекстное меню набора данных – Fields Editor



Работа с полями набора данных

- Кроме обычных полей данных, также можно создать вычисляемые поля и поля выбора



Работа с полями набора данных

- **Вычисляемые поля** – это поля, которые не существуют в реальных таблицах, а вычисляются по значениям других полей
- Для задания вычисляемого поля – используется Field type – **Calculated**
- Выражение для вычисления нужно задать в обработчике события **onCalcFields** набора данных.
- Данное событие возникает при открытии НД, а также при перемещении курсора от одной записи к другой.
- Если свойство **AutoCalcFields = true**, то событие **onCalcFields** возникает при каждой модификации невычисляемых полей этого набора данных.

Вычисляемые поля

Пример: Для всех сотрудников выдать информацию по текущей зарплате, а также значение зарплаты, увеличенной на 20%

Требуется:

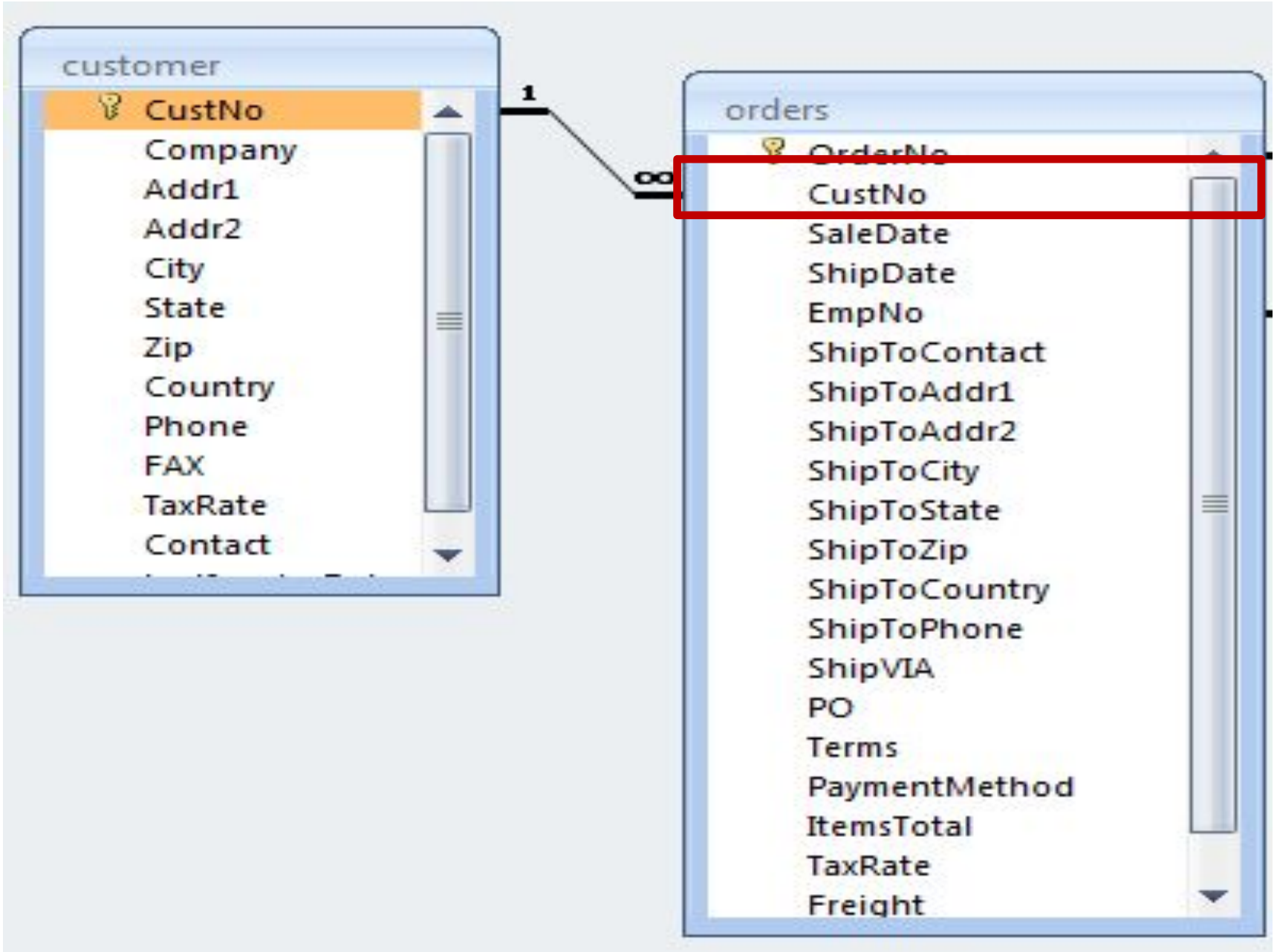
1. В список полей добавить новое вычисляемое поле «HighSalary»
2. Создать обработчик события onCalcFields таблицы

```
Procedure
  TForm1.ADOTable1CalcFields(DataSet:TDataSet);
Begin
   ADOTable1HighSalary.asFloat :=
                        1.2*ADOTable1Salary.asFloat;
End;
```

Поля выбора

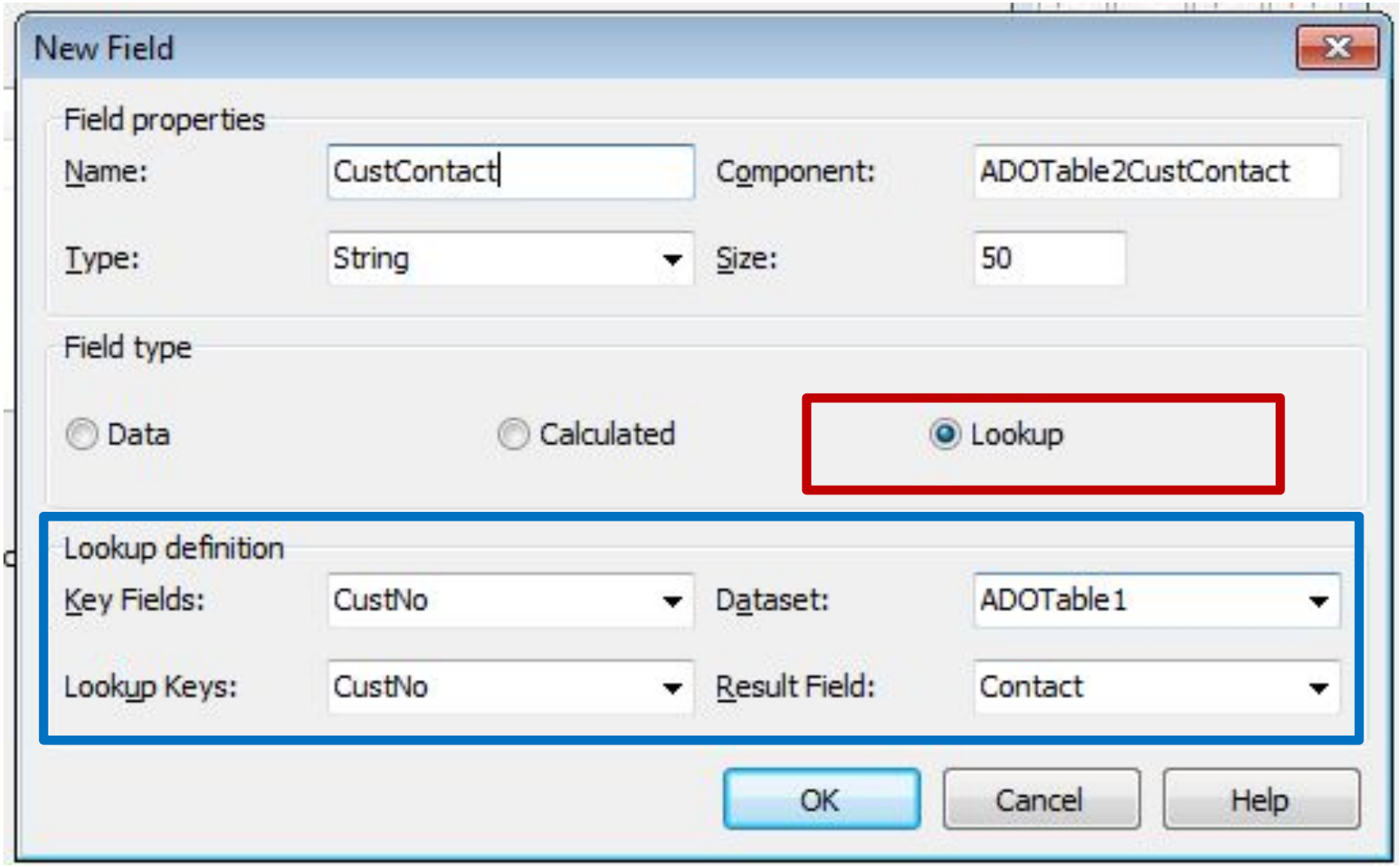
- **Поля выбора (lookup-поля)** – это поля, которые автоматически получают значения, выбранные из списка, формируемого на основе значений определенного поля из другого набора.
- Поля выбора позволяют отображать данные из таблицы, связанной отношением «многие-к-одному» с исходной таблицей.
- Для создания таких полей необходимо обязательно указать поле, через которое

Поля выбора


customer
CustNo
Company
Addr1
Addr2
City
State
Zip
Country
Phone
FAX
TaxRate
Contact
1
∞
orders
OrderNo
CustNo
SaleDate
ShipDate
EmpNo
ShipToContact
ShipToAddr1
ShipToAddr2
ShipToCity
ShipToState
ShipToZip
ShipToCountry
ShipToPhone
ShipVIA
PO
Terms
PaymentMethod
ItemsTotal
TaxRate
Freight

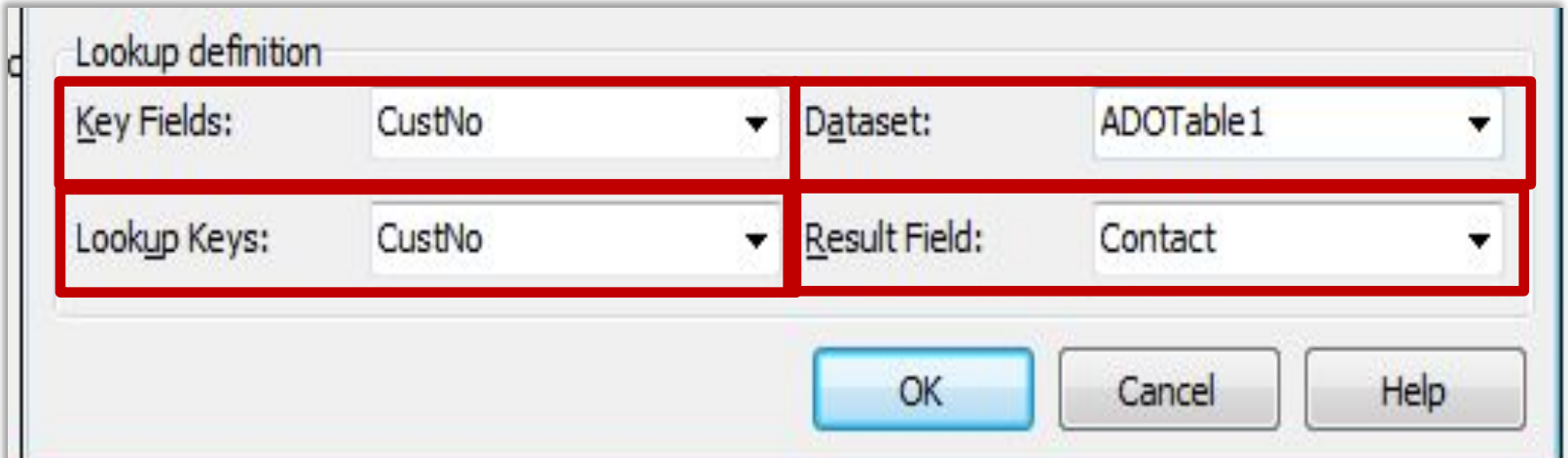
Поля выбора

- Для определения поля выбора необходимо в редакторе полей создать новое поле типа **Lookup**.



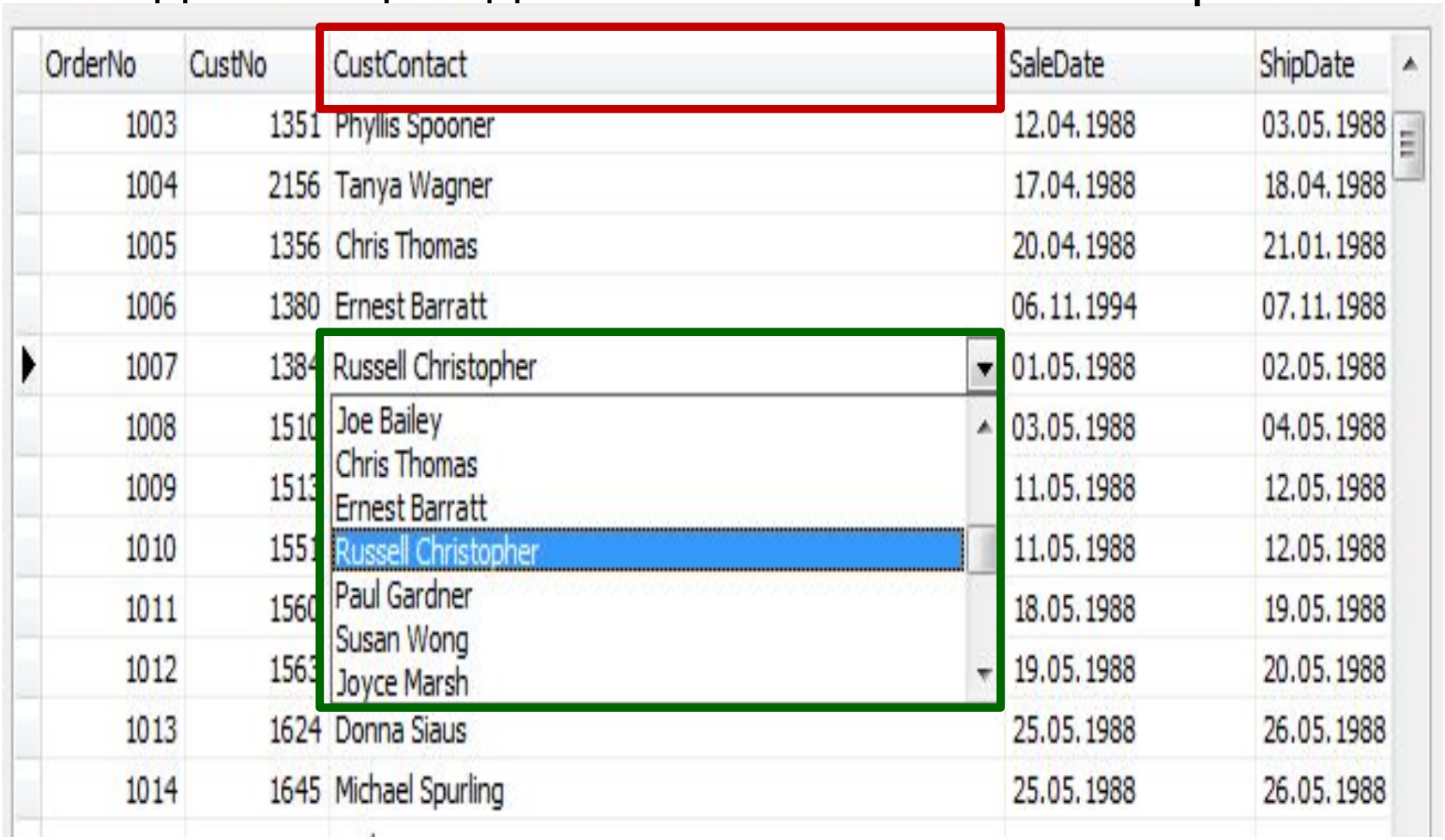
Поля выбора

- Далее в группе параметров **Lookup definition** следует настроить параметры для связи таблиц:
  - **Key Fields** – поле в исходной таблице (НД-1), используемое в качестве внешнего ключа
  - **Dataset** – набор данных (НД-2), из которого поле выбора должно получать значения
  - **Lookup Keys** – поле для связи, определенное в НД-2
  - **Result Field** – поле результата, значение которого нужно выводить в исходной таблице (НД-1)



Поля выбора

- Вид таблицы с добавленным полем выбора



НАВИГАЦИЯ ПО НАБОРУ ДАННЫХ

Навигация по набору данных

- **First** – перейти к первой записи набора
- **Last** – перейти к последней записи набора
- **Next** – перейти к следующей записи
- **Prev** – перейти к предыдущей записи
- **MoveBy**(Distance:integer) – перейти на указанное в параметре **Distance** число записей вперед или назад

Навигация по набору данных

Пример: Найти сумму значений 4-го столбца таблицы

```
ADOTable1.First;
s := 0;
While not ADOTable1.EOF do
    begin
       x := ADOTable1.Fields[3].AsInteger;
       s := s + x;
      ADOTable1.Next;
   end;
```

Навигация по набору данных

Пример: Найти сумму значений 4-го столбца таблицы

```
ADOTable1.Last;
s := 0;
While not ADOTable1.BOF do
    begin
      x := ADOTable1.Fields[3].AsInteger;
      s := s + x;
     ADOTable1.Prev;
   end;
```

Навигация по набору данных

Пример: Найти сумму значений 4-го столбца таблицы

```
ADOTable1.Last;
s := 0;
While not ADOTable1.BOF do
    begin
       x := ADOTable1.Fields[3].AsInteger;
       s := s + x;
      ADOTable1.Move(-1);
   end;
```

МОДИФИКАЦИЯ ДАННЫХ

Модификация данных

- **Append** – добавление новой записи в конец набора
- **Delete** – удаление текущей записи набора
- **Edit** – перевод текущей записи в режим редактирования
- **Cancel** – отмена изменений, сделанных в текущей записи
- **Post** – сохранение изменений, сделанных в текущей записи
- **SetFields**(**const** Values: **array of const**) – используется для изменения значений всех или некоторых полей текущей записи

Модификация данных

- **Refresh** – обновление результирующего набора путем повторного извлечения данных из набора
- **Insert** – вставляет новую запись в текущую позицию
- **InsertRecord** – вставляет новую запись в текущую позицию со значениями, указанными в параметрах
- **AppendRecord** – вставляет новую запись в конец набора со значениями, указанными в параметрах

Модификация данных

- Перед модификацией данных, нужно вызвать метод **Edit**, **Append** или **Insert**, для перевода набора данных в состояние редактирования или добавления.
- Для **сохранения** измененных значений записи, требуется вызвать метод **Post** или **Next**, для **отмены** изменений – метод **Cancel**
- Непосредственное изменение значения текущего поля активной записи выполняется одним из следующих способов:

1. ADOTable1.Fields[1].AsString := 'Иванов';
2. ADOTable1.FieldByName('CustName').asString := 'Иванов';
3. ADOTable1.FieldValues['CustName'] := 'Иванов';
4. ADOTable1['CustName'] := 'Иванов';

Добавление записи

Пример: В таблицу добавить запись со следующими значениями: CustNo = 1345, CustName = ‘Петров’, Price = 120.56;

```
With ADOTable1 do
Begin
    Append;
    FieldValues[‘CustNo’] := 1345;
    FieldValues[‘Price’] := 120.56;
    FieldByName(‘CustName’).asString :=‘Петров’;
    Post;
end;
```

Редактирование записи

Пример: В выбранной записи таблицы увеличить значение поля Price на 10%

```
var x:single;
...
With ADOTable1 do
Begin
    Edit;
    x := FieldValues['Price'];
    FieldValues['Price'] := 1.1*x;
    Post;
end;
```

Редактирование записи

Пример: Для всех записей таблицы увеличить значение поля Price на 10%

```
var x:single;
...
With ADOTable1 do begin
    First;
    While not EOF do
        Begin
            Edit;
            x := FieldValues['Price'];
            FieldValues['Price'] := 1.1*x;
            Next;
        end;
end;
```

Для сохранения результатов используется метод **Next**

Редактирование записи

Пример: В текущей записи изменить значения полей таблицы: CustNo = 1345, CustName = ‘Петров’, Price = 120.56;

(Предположим, в таблице имеются следующие поля: CustNo, City, CustName, Price)

```
With ADOTable1 do
Begin
    Edit;
    SetFields([1345, nil, ‘Петров’, 120.56]);
    Post;
end;
```

Константа **nil** используется для сохранения предыдущего значения поля

Поиск данных

Фильтрация данных

Свойства

- **Filter**:string; – свойство для задания фильтра. Фильтр определяет условие, которому должны удовлетворять доступные записи
- **Filtered**: boolean – определяет, используется ли фильтр, заданный свойством Filter

Событие

- **onFilterRecord** – возникает, когда свойство Filtered устанавливается в True. В параметре Accept события можно указывать дополнительные (к Filter) условия

Фильтрация данных

Пример: Найти всех сотрудников с номерами > 1200, имена которых начинаются от буквы ‘A’ до ‘D’

```
ADOTable1.Filtered :=False;
ADOTable1.Filter :=
      'CustNo>1200 and Contact<’#29'E’#29;
ADOTable1.Filtered :=True;
```

#29 – символ «’». Используется для задания условия на строковые значения

```
ADOTable1.Filter :=
      'CustNo>1200 and Contact<’’’+‘E’+’’’;
```

Поиск данных

- Поиск записей может осуществляться следующими методами:
  - **Found, FindFirst**, **FindPrev**, **FindNext**, **FindLast** – логические функции, возвращающие **True**, если найдена запись, удовлетворяющая заданному фильтру (при этом значение свойства **Filtered** может быть как **True** так и **False**).
  - **Locate**
  - **Lookup**

Поиск данных

Пример: Определить, есть ли в таблице сотрудники, номера которых > 2000

```
ADOTable1.Filtered :=False;
ADOTable1.Filter :='CustNo>2000';
if ADOTable1.FindFirst
    then ShowMessage('Есть такие сотрудники')
    else ShowMessage('Нет таких сотрудников');
```

Поиск данных

Пример: Определить минимальную зарплату (поле Price) сотрудников с номерами > 2000

```
ADOTable1.Filtered :=False;
ADOTable1.Filter := 'CustNo>2000';
If ADOTable1.FindFirst then
    min_price := ADOTable1.FieldValues['Price'];
While ADOTable1.Found do
   begin
      x := ADOTable1.FieldValues['Price'];
      if x < min_price then min_price := x;
      ADOTable1.FindNext;
   end;
```

Поиск данных

- Метод **Locate**

```
Function Locate(const KeyFields;
                const KeyValues: Variant;
                Options:TLocateOptions):Boolean;
```

KeyFields – поле/поля, по которым ведется поиск в виде строки, разделитель «;»

KeyValues – вариантный массив критериев поиска

Options – множество режимов поиска:

- [] – пустое множество означает полное совпадение с критерием
- [**loCaseInsensitive**] – без учета регистра
- [**loPartialKey**] – частичное совпадение с критерием поиска

Поиск данных – метод Locate

- Метод **Locate** производит поиск по любым полям.
- Первая запись, удовлетворяющая критериям поиска становится активной.

Пример – поиск по одному полю:

Найти сотрудника по следующему критерию:

CustName=‘Петров’

```
if ADOTable1.Locate('CustName', 'Петров',[])
  then ShowMessage ('Запись найдена')
  else  ShowMessage('Запись не найдена');
```

Поиск данных – метод Locate

Пример – поиск по нескольким полям:

Найти запись по следующим критериям:

CustName=‘Петров’, City = ‘Москва’, Price = 1235

Установить режим частичного совпадения.

```
if ADOTable1.Locate('CustName;City;Price',
VarArrayOf('Петров', 'Москва', 1235),
[loPartialKey])
  then ShowMessage ('Запись найдена')
  else  ShowMessage('Запись не найдена');
```

Поиск данных

- Метод **Lookup** находит запись, удовлетворяющую критериям поиска, но не делает её текущей, а возвращает значения некоторых полей этой записи.

```
Function Lookup(const KeyFields;
                const KeyValues: Variant;
                const ResultFields:String):Variant;
```

KeyFields – поле/поля, по которым ведется поиск в виде строки, разделитель «;»

KeyValues – вариантный массив критериев поиска

Поиск данных

- Метод **Lookup**

ResultFields – список полей, значения которых возвращает метод Lookup, если запись, удовлетворяющая критериям поиска найдена:
  - Если указано одно поле, то результат – либо значение этого поля, либо NULL
  - Если указано несколько полей, то результат – вариантный массив, число элементов которого не превышает количество результирующих полей
- Для проверки типа результата (переменная, или вариантный массив) используется логическая функция **VarIsArray**

Поиск данных – метод Lookup

Пример – поиск по одному полю:

Определить зарплату сотрудника по следующему критерию: CustName= ‘Петров’

```
Var Res:Variant;
        s:string;
  ...
Res := ADOTable1.Lookup('CustName', 'Петров',
'Price');
 case VarType(Res) of
  varEmpty : s:= 'Нет данных о зарплате';
  varNull : s:= 'Запись не найдена';
 else s := Res;
 end;
 ShowMessage(s);
```

Поиск данных – метод Lookup

Пример – поиск по нескольким полям:

Определить зарплату и город сотрудника по следующему критерию: CustName= ‘Петров’,

```
Var Res:Variant; s:string;x:integer;
 ...
  Res := ADOTable1.Lookup('CustName', 'Петров',
                                        'Price;City');
if VarIsArray(Res) then
 begin
    x:= Res[0];
    if Res[1]<>Null then s:= Res[1];
 end else
 case VarType(Res) of
 varEmpty : s:= 'Нет данных о зарплате';
 varNull : s:= 'Запись не найдена';
else s := Res;
 end;
 ShowMessage(s);
```

СВЯЗАННЫЕ КУРСОРЫ

Связанные курсоры

Связанные курсоры позволяют программистам определить отношение «один ко многим»

Для этих целей используются следующие свойства наборов данных:

- **MasterSource**:TDataSource;
- **MasterFields**:WideString;

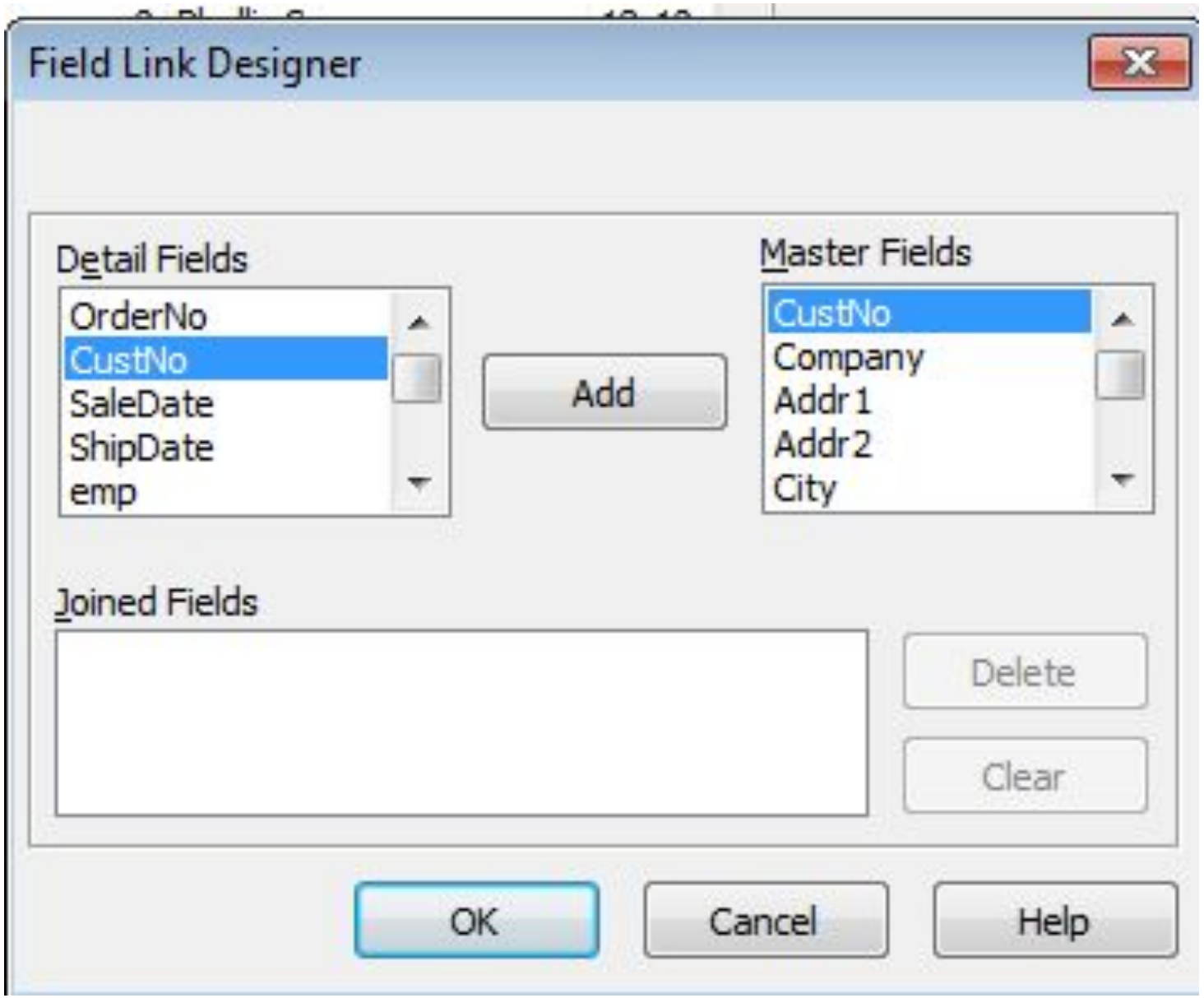
Свойство **MasterSource** – определяет источник данных родительской таблицы

Свойство **MasterFields** – определяет поле/поля для связи родительской таблицы с дочерней

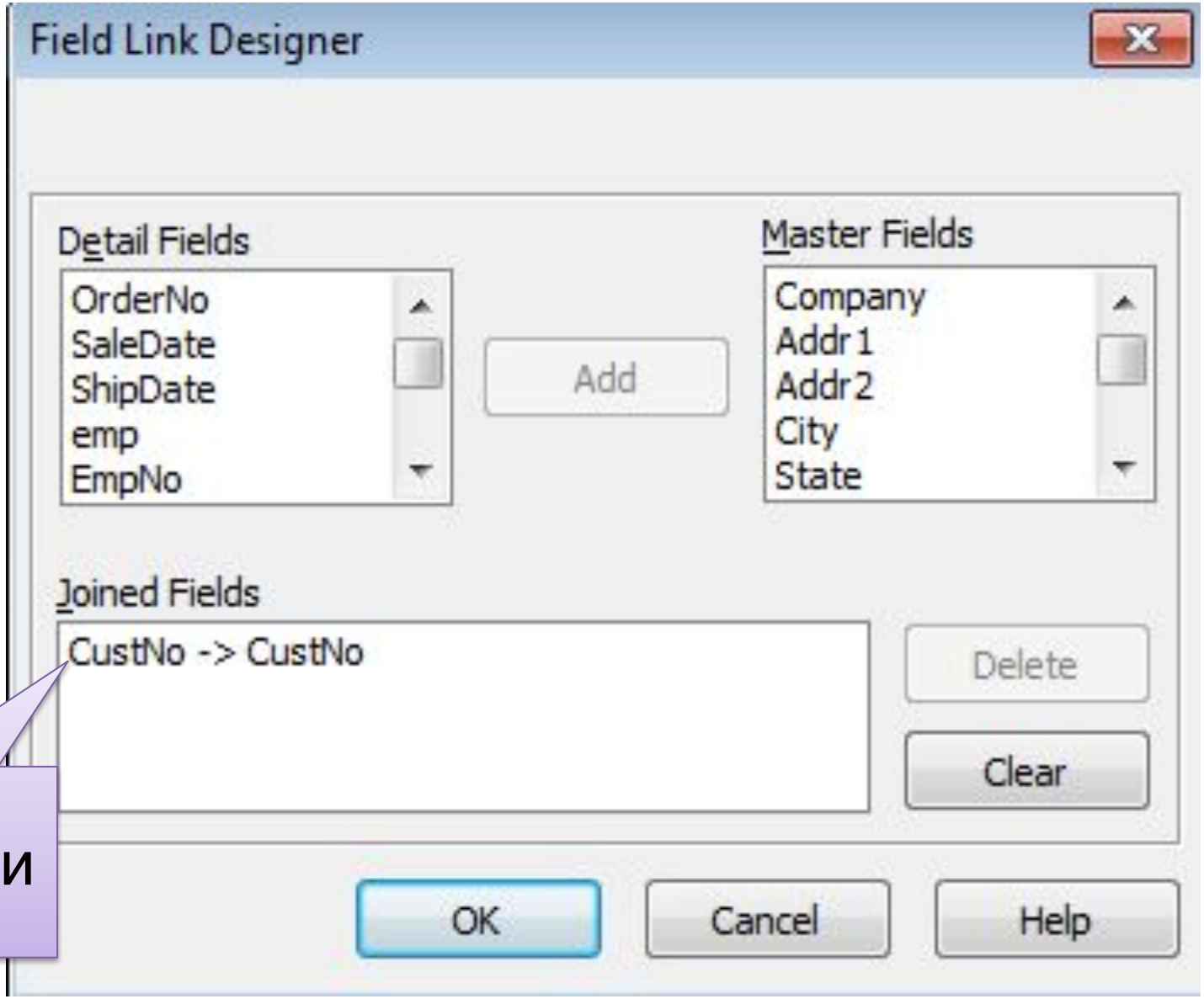
Связанные курсоры

- Для создания связи нужно:
  - в свойстве поля **MasterSource** дочерней таблицы установить DataSource, связанный с родительской таблицей.
  - Открыть свойство **MasterFields**
  - В появившемся окне выбрать связанные поля и нажать на кнопку **ADD**.
- В результате при перемещении по родительской таблицы, в дочерней таблице будут отображаться только связанные записи

Связанные курсоры



Связанные курсоры



Условие соединения