

Министерство науки и высшего образования Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

С.А. Глебов, С.С. Гришунов

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ВЗАИМОДЕЙСТВИЯ С БД
Методические указания к выполнению домашней работы
по дисциплине «Базы данных»

Калуга – 2019

УДК 004.42
ББК 32.972.13
Г53

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 51.4/04 от « 20 » ноября 2019 г.


Зав. кафедрой ИУ4-КФ  к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол № 5 от «25» ноября 2019 г.

Председатель методической комиссии факультета ИУ-КФ  к.т.н., доцент М.Ю. Адкин

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол № 3 от « 3 » 12 2019 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана  д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ИУ6-КФ  А.Б. Лачихина

Авторы
к.ф.-м.н., доцент кафедры ИУ4-КФ
асс. кафедры ИУ4-КФ  С.А. Глебов
 С.С. Гришунов

Аннотация

Методические указания к выполнению домашней работы по курсу «Базы данных» содержат описание технологии ADO.Net и методов провайдера данных OLE DB для работы с базами данных Firebird.

Предназначены для студентов 3-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

© Калужский филиал МГТУ им. Н.Э. Баумана, 2019 г.
© С.А. Глебов, С.С. Гришунов, 2019 г.

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	3
ВВЕДЕНИЕ	4
ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ.....	5
КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ	6
ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ	58
КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ	58
ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ.....	58
ОСНОВНАЯ ЛИТЕРАТУРА.....	59
ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА	59

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения курса «Базы данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержат руководство по настройке взаимодействия между базами данных Firebird и клиентскими приложениями, руководство по установке и использованию OLE DB провайдера данных и задание на выполнение домашней работы.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Цель выполнения домашней работы: сформировать практические навыки разработки программного обеспечения, взаимодействующего с базой данных.

Основные задачи выполнения домашней работы:

- выполнить анализ исходных данных;
- изучить технологии доступа к данным и методы создания интерфейса приложения;
- реализовать приложение, взаимодействующее с базой данных, разработанной в ходе выполнения лабораторных работ.

Результатами работы являются:

- База данных
- Клиентское приложение для работы с базой данных
- Подготовленный отчет.

Требования к программному обеспечению:

- Microsoft Windows 7/8/10
- Microsoft Visual Studio Community 2017
- Firebird 2.5

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Для того чтобы осуществлять связь между базой данных и приложением на C# необходим посредник. Платформа .NET определяет ряд пространств имен, которые позволяют непосредственно взаимодействовать с локальными и удаленными базами данных. Вместе эти пространства имен известны как ADO.NET.

Основу интерфейса взаимодействия с базами данных в ADO.NET представляет ограниченный круг объектов: Connection, Command, DataReader, DataSet и DataAdapter. С помощью объекта Connection происходит установка подключения к источнику данных. Объект Command позволяет выполнять операции с данными из БД. Объект DataReader считывает полученные в результате запроса данные. Объект DataSet предназначен для хранения данных из БД и позволяет работать с ними независимо от БД. И объект DataAdapter является посредником между DataSet и источником данных. Главным образом, через эти объекты и будет идти работа с базой данных.

Однако, чтобы использовать один и тот же набор объектов для разных источников данных, необходим соответствующий провайдер данных. Через провайдер данных в ADO.NET и осуществляется взаимодействие с базой данных. Причем для каждого источника данных в ADO.NET может быть свой провайдер, который, собственно, и определяет конкретную реализацию вышеуказанных классов.

По умолчанию в ADO.NET имеются следующие встроенные провайдеры:

- Провайдер для MS SQL Server
- Провайдер для OLE DB (Предоставляет доступ к некоторым старым версиям MS SQL Server, а также к БД Access, DB2, MySQL и Oracle)
- Провайдер для ODBC (Провайдер для тех источников данных, для которых нет своих провайдеров)
- Провайдер для Oracle

- Провайдер EntityClient. Провайдер данных для технологии ORM Entity Framework
- Провайдер для сервера SQL Server Compact 4.0
- Кроме этих провайдеров, которые являются встроенными, существует также множество других, предназначенных для различных баз данных, например, для MySQL.

Основные пространства имен, которые используются в ADO.NET:

- `System.Data`: определяет классы, интерфейсы, делегаты, которые реализуют архитектуру ADO.NET
- `System.Data.Common`: содержит классы, общие для всех провайдеров ADO.NET
- `System.Data.Design`: определяет классы, которые используются для создания своих собственных наборов данных
- `System.Data.Odbc`: определяет функциональность провайдера данных для ODBC
- `System.Data.OleDb`: определяет функциональность провайдера данных для OLE DB
- `System.Data.Sql`: хранит классы, которые поддерживают специфичную для SQL Server функциональность
- `System.Data.OracleClient`: определяет функциональность провайдера для баз данных Oracle
- `System.Data.SqlClient`: определяет функциональность провайдера для баз данных MS SQL Server
- `System.Data.SqlServerCe`: определяет функциональность провайдера для SQL Server Compact 4.0
- `System.Data.SqlTypes`: содержит классы для типов данных MS SQL Server
- `Microsoft.SqlServer.Server`: хранит компоненты для взаимодействия SQL Server и среды CLR

Функционально классы ADO.NET можно разбить на два уровня: подключенный и отключенный. Каждый провайдер данных .NET реализует свои версии объектов Connection, Command, DataReader,

DataAdapter и ряда других, который составляют подключенный уровень. То есть с помощью них устанавливается подключение к БД и выполняется с ней взаимодействие. Как правило, реализации этих объектов для каждого конкретного провайдера в своем названии имеют префикс, который указывает на провайдер.

Другие классы, такие как DataSet, DataTable, DataRow, DataColumn и ряд других составляют отключенный уровень, так как после извлечения данных в DataSet мы можем работать с этими данными независимо от того, установлено ли подключение или нет. То есть после получения данных из БД приложение может быть отключено от источника данных.

OLE DB провайдер

OLE DB (Object Linking and Embedding, Database) — набор COM-интерфейсов, которые позволяют приложениям унифицировано работать с данными разных источников и хранилищ информации.

OLE DB отделяет хранилище данных от приложения, которое должно иметь доступ к нему через набор абстракций, состоящий из источника данных (DataSource), сессии (Session), команды (Command) и набора строк (Rowset). Это было сделано для предоставления унифицированного доступа к различным видам и источникам данных и изоляцию специфики взаимодействия с конкретным хранилищем. OLE DB концептуально разделена на потребителей (клиентов) и поставщиков (провайдеров). Потребителем является приложение, которому необходим доступ к данным, а поставщик реализует интерфейс доступа к данным и, следовательно, обеспечивает информацией потребителя.

Провайдеры OLE DB могут предоставлять доступ как к простым хранилищам данных, в виде текстовых файлов и электронных таблиц, так и к «настоящим» базам данных под управлением Oracle Database, Microsoft SQL Server, Sybase ASE, Firebird и Interbase. Также возможен доступ и к иерархическим хранилищам данных таких, как системы электронной почты.

Поскольку различные хранилища данных могут иметь разные возможности, поставщики OLE DB, как правило, не поддерживают все

интерфейсы, описанные в спецификации OLE DB. Доступные возможности поставщика данных определяются через запрос указателей на COM интерфейсы его объектов или через чтение информационных свойств источника данных (DataSource).

Для доступа к Firebird и InterBase в качестве OLE DB драйвера можно использовать IBProvider. Он поддерживает автоматическую настройку на работу с Firebird и InterBase всех версий, кроме того стандарт OLE DB позволяет использовать его в качестве Firebird .Net провайдера, а так же InterBase .Net провайдера.

Установка IBProvider-a

- Скачиваем MSI-инсталляторы IBProvider-a для 32-битной и 64-битной Windows.
- Устанавливаем оба пакета с использованием конфигурации по-умолчанию.

Подготовка проекта

Для всех примеров будет использоваться консольное приложение на C# для FW 4.5.1.

Добавление ссылок на сборки ADO.NET провайдера

В проект нужно будет добавить ссылки на сборки [ADO.NET](#) провайдера. Есть несколько способов это сделать.

Первый способ. Через UI Visual Studio.

1.Открываем меню со свойствами проекта и выбираем «Add->Reference»:

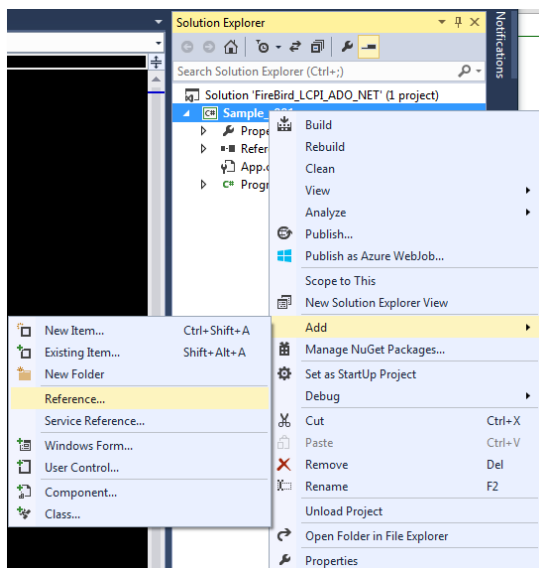


Рис. 1.

2. Находим и выбираем сборки:

- «LCPI ADO.NET Data Provider for OLE DB [NET 4.5.1]»
- «LCPI Instrumental Library for .NET 4.5.1»

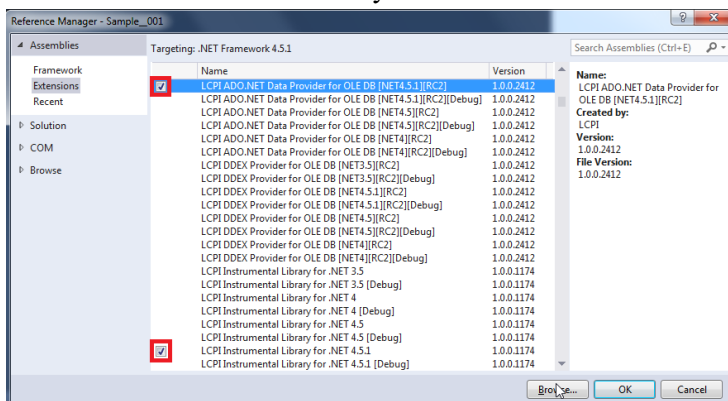


Рис. 2.

Второй способ. Через прямое редактирование csproj файла. Можно подключать сборки с учетом конфигурации проекта.

Третий способ. Ну и последний, пожалуй, самый простой способ – подключить к проекту NUGET-пакет «lcp1.data.oledb».

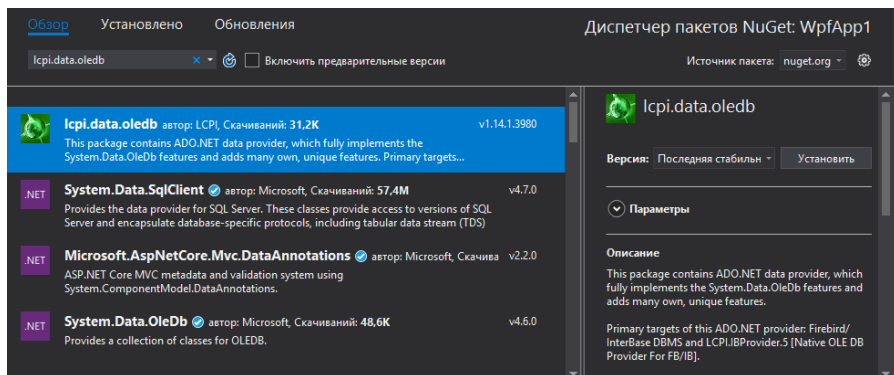


Рис. 3.

Пространства имен

Компоненты ADO.NET провайдера находятся в пространстве имен «lcp1.data.oledb». Добавим в начало cs-файлов следующую строку:

```
using lcp1.data.oledb;
```

Дополнительно, для удобства, нужно открыть доступ к пространству имен с общими для всех ADO.NET провайдеров конструкциями – «System.Data»:

```
using System.Data;
```

Работа с подключением к базе данных

Управление подключением осуществляется с помощью объекта класса OleDbConnection.

- Подключение базы данных
- Для подключения к базе данных нужно:
- Создать объект OleDbConnection.
- Указать строку подключения.
- Вызвать метод Open.

Строка подключения

```
static void Test_001()
{
    const string cn_str=
        "provider=LCPI.IBProvider.3;"

    +"location=localhost:d:\\database\\employee.fdb;"
    +"user id=SYSDBA;\n"
    +"password=masterkey;\n"
    +"dbclient_library=fbclient.dll";

    var cn=new OleDbConnection();

    cn.ConnectionString=cn_str;

    cn.Open();
} //Test_001
```

Строку подключения можно передавать прямо в конструктор класса OleDbConnection:

```
var cn=new OleDbConnection(cn_str);
```

Параметры приведенной строки подключения разделяются на две группы:

- Параметры ADO.NET провайдера: provider.
- Параметры OLE DB провайдера: location, user id, password, dbclient_library.

ADO.NET провайдер обрабатывает следующие параметры:

Таблица 1

Имя	Назначение
Provider	Идентификатор OLE DB провайдера.
File Name	Файл с параметрами подключения базы данных.

У [OLE DB](#) провайдера (IBProvider-a) настроек подключения гораздо больше. Вот основные:

Таблица 2

Имя	Назначение
Location	Расположение базы данных.
User ID	Имя пользователя.
Password	Пароль пользователя.
dbclient_library	Имя или файловый путь к серверному клиенту (gds32.dll, fbclient.dll, ibclient64.dll).
dbclient_library_64	Имя или файловый путь к 64-битному серверному клиенту. Учитывается только в 64-битных процессах.
ctype	Кодовая страница подключения.
ctype_none	Кодовая страница для текстовых данных без указания кодовой страницы (NONE).
auto_commit	Разрешение использовать автоматические транзакции.
nested_trans	Разрешение на создание «вложенных» транзакций.
named_param_prefix	Префикс именованных параметров. По умолчанию — «:».

Кроме того, есть свойства, которые обрабатываются как ADO.NET провайдером, так и OLE DB провайдером. Например:

Таблица 3

Имя	Назначение
OLE DB Services	Конфигурация взаимодействия с OLE DB провайдером. В частности – использование пула подключений.
Persist Security Info	Конфигурация доступа к значениям параметров аутентификации после подключения к базе данных.

Для упрощения процесса формирования строки подключения из кода программы, в ADO.NET провайдере реализован специальный компонент – OleDbConnectionStringBuilder.

Значения свойств можно указывать через «индексатор»:

```
static void Test_003()
{
    var cnsb=new OleDbConnectionStringBuilder();
    cnsb["provider"]="LCPI.IBProvider.3";
    cnsb["location"]="localhost:d:\\database\\employee.fdb";

    cnsb["user id"]="SYSDBA";
    cnsb["password"]="masterkey";
    cnsb["dbclient_library"]="fbclient.dll";
    var cn=new OleDbConnection(cnsb.ConnectionString)
    cn.Open();
}
```

Или через свойства класса OleDbConnectionStringBuilder:

```
static void Test_004()
{
    var cnsb=new OleDbConnectionStringBuilder();
    cnsb.Provider="LCPI.IBProvider.3";
    cnsb.Location="localhost:d:\\database\\employee.fdb";

    cnsb.UserID="SYSDBA";
    cnsb.Password="masterkey";
    cnsb.IBProvider.dbclient_library="fbclient.dll";
    var cn=new OleDbConnection(cnsb.ConnectionString);
    cn.Open();
}
```

По-умолчанию, после подключения к базе данных из строки подключения (OleDbConnection.ConnectionString) будет исключено свойство с паролем:

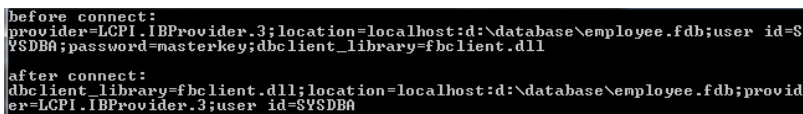
```

static void Test_005()
{
    const string cn_str=
        "provider=LCPI.IBProvider.3;"

+ "location=localhost:d:\\database\\employee.fdb;"
    + "user id=SYSDBA;"
    + "password=masterkey;"
    + "dbclient_library=fbclient.dll";

    var cn=new OleDbConnection(cn_str);
    Console.WriteLine("before connect:");
    Console.WriteLine(cn.ConnectionString);
    cn.Open();
    Console.WriteLine("");
    Console.WriteLine("after connect:");
    Console.WriteLine(cn.ConnectionString);
}

```



```

before connect:
provider=LCPI.IBProvider.3;location=localhost:d:\database\employee.fdb;user id=SYSDBA;password=masterkey;dbclient_library=fbclient.dll
after connect:
dbclient_library=fbclient.dll;location=localhost:d:\database\employee.fdb;provider=LCPI.IBProvider.3;user id=SYSDBA

```

Рис. 4.

Это поведение зависит от значения свойства «Persist Security Info»:

- False указывает исключать из строки подключения параметры, связанные с безопасностью. Это значение по-умолчанию.
- True оставляет в строке подключения значения свойств, относящихся к безопасности.

Использование файла с параметрами инициализации

Параметры инициализации подключения можно сохранить во UDL-файле и указать в строке подключения путь к этому файлу:

- Создайте пустой файл с расширением UDL — «test.udl».

- Выберите этот файл в проводнике Windows и нажмите Enter.
- Откроется диалог «Data Links» для настройки параметров подключения.
- Переключитесь на страницу «Поставщик данных» и выберите «LCPI OLE DB Provider for InterBase [v3]»:

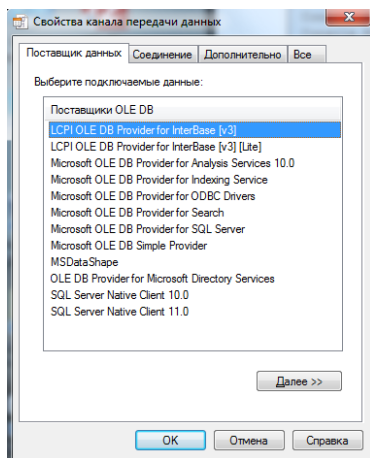


Рис. 5.

- Нажмите «Далее >>» или щелкните на вкладке «Соединение».
- Заполните поля с основными параметрами соединения:

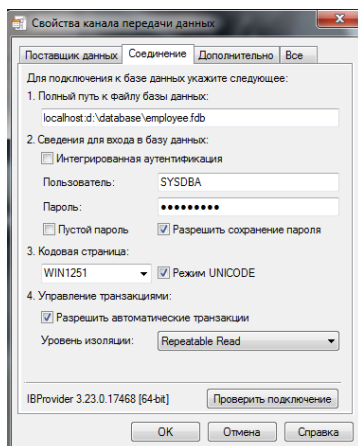


Рис. 6.

- На странице «Дополнительно» можно указать расширенные параметры соединения. Например, имя серверного клиента (fbclient.dll):

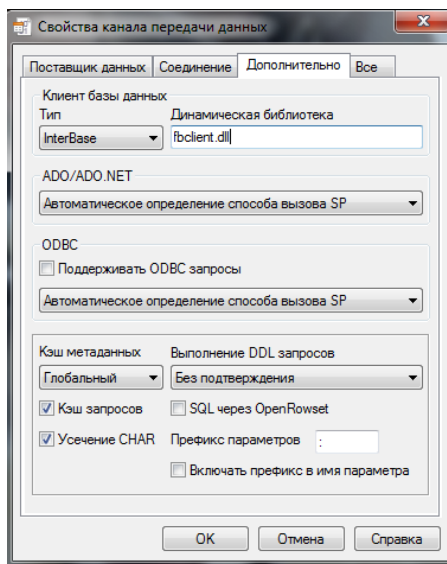


Рис. 7.

- Теперь можно вернуться на страницу «Соединение» и проверить параметры подключения, нажав кнопку «Проверить подключение». Если все было указано правильно, то появится диалог с информацией о типе и версии сервера базы данных:

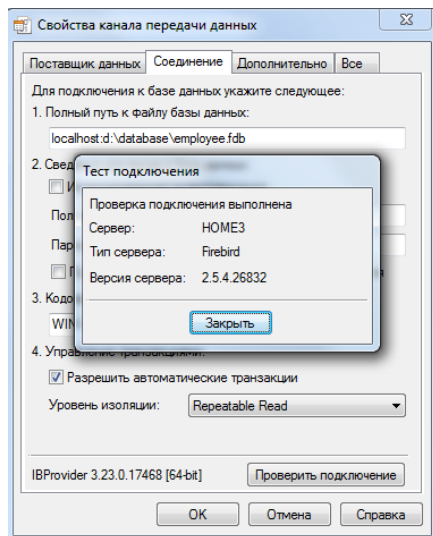


Рис. 8.

- Закрываем диалоги («Заккрыть», «ОК»). Разрешаем сохранение пароля в открытом виде:

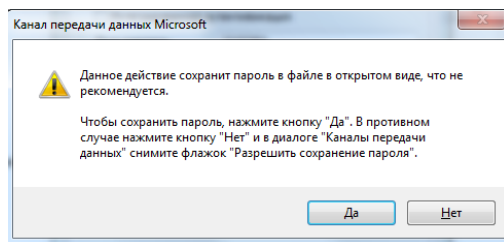


Рис. 9.

- Пишем следующий код:

```
static void Test_UDL()
{
    var cn=new OleDbConnection("file name=test.udl");
    cn.Open();
}
```

Здесь главное позаботиться о том, чтобы тестовый процесс нашел «test.udl». Лучше всего держать этот файл в одном каталоге с EXE программы.

Отключение от базы данных

Есть несколько способов завершения работы с подключением к базе данных.

Первый – с помощью метода `OleDbConnection.Close`. После вызова метода `Close` объект подключения остается работоспособным и с ним можно продолжать работать. Например, снова вызвать метод `Open`:

```
static void Test_006__close()
{
    var cn=new OleDbConnection("provider=LCPI.IBP
rovider.3;"
+"location=localhost:d:\\database\\employee.fdb;"
+"user id=SYSDBA;\n"

+"password=masterkey;\n"
+"dbclient_library=fbclient.dll");
    cn.Open();
    cn.Close();
    cn.Open();
    cn.Close();
}
```

Второй метод - это вызов метода `OleDbConnection.Dispose`. Он освобождает подключение и полностью деинициализирует объект подключения. Если вы попытаете продолжить с ним работать, то получите исключение `ObjectDisposedException`:

```
static void Test_006__dispose()
{
    var cn=new OleDbConnection(
        "provider=LCPI.IBProvider.3;"
```

```

+"location=localhost:d:\\database\\employee.fdb;"
+"user id=SYSDBA;\n"
+"password=masterkey;\n"
+"dbclient_library=fbclient.dll");

cn.Open();
cn.Dispose();

try
{
    cn.Open();
}
catch(ObjectDisposedException e)
{
    Console.WriteLine("ERROR: {0} -
        {1}", e.Source, e.Message);
}
}

```

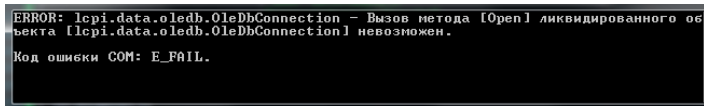


Рис. 10.

Во всех случаях, [ADO.NET](#) провайдер не иницирует фактическое отключение от базы данных. Он просто освобождает указатели на объекты OLE DB провайдера.

Реальное отключение произойдет только после освобождения последней ссылки на [OLE DB](#) подключение.

Если соединение с базой данных было создано через пул подключений (это режим по-умолчанию) или на COM-объект источника данных OLE DB остались внешние ссылки, то подключение останется активным.

Выполнение запросов к базе данных

Для работы с запросами в ADO.NET провайдере определен компонент `OleDbCommand`. Запросы выполняются в рамках открытого подключения и, как правило, активной транзакции. Существует пара способов создания объекта команды.

Первый способ – это явное конструирование объекта:

```
var cmd=new OleDbCommand();
```

`OleDbCommand` предоставляет дополнительные конструкторы, с помощью которых можно сразу указать текст запроса, подключение и транзакцию.

Второй способ – через вызов метода `OleDbConnection.CreateCommand`:

```
var cmd=cn.CreateCommand();
```

Полученный объект команды будет привязан к объекту подключения «cn».

Текст запроса указывается через свойство `CommandText` или передается в конструктор `OleDbCommand`.

Выполнение запроса выполняется через вызов одного из методов: `ExecuteScalar`, `ExecuteReader` или `ExecuteNonQuery`. Все зависит от типа запроса и способа получения результата. Формально, все запросы можно выполнять с помощью метода `ExecuteReader`.

ExecuteScalar

Метод `OleDbCommand.ExecuteScalar` возвращает единственное значение первой колонки первой строки. Остальные результаты игнорируются. Этот метод полезен для запросов, которые, к примеру, вычисляют количество записей в таблице — соответственно возвращают только одно значение:

```
static void Test_015__ExecuteScalar()  
{  
    const string c_sql=  
        "select count(*) from employee";  
    const string c_cn_str =  
        "provider=LCPI.IBProvider.3;\n"
```

```

+"location=localhost: "
+"d:\\database\\employee.fdb;\n"
+"user id=SYSDBA;\n"
+"password=masterkey;\n"
+"dbclient_library=fbclient.dll";

object n=null;

using(var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();

    using(var tr=cn.BeginTransaction())
    {
        using(var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            n=cmd.ExecuteScalar();
        }//using cmd

        tr.Commit();
    }//using tr
}//using cn

string s;

if(Object.ReferenceEquals(n,null))
    s="null";
else
    if(n.Equals(DBNull.Value))
        s="DBNull";
    else
        s=n.ToString();

Console.WriteLine("n: {0}",s);
} //n: 8

```

В случае пустого результирующего множества, то есть с нулевым количеством рядов, ExecuteScalar вернет null.

Если запрос вообще не возвращает множество (например, это update-запрос), то ExecuteScalar так же возвращает null.

ExecuteReader

OleDbCommand.ExecuteReader выполняет команду и возвращает объект OleDbDataReader, предназначенный для однонаправленного перебора записей результирующего множества.

Переход на следующую запись выполняется методом OleDbDataReader.Read. Он возвращает:

- true, если выбрана очередная запись результирующего множества.
- false, если записей больше не осталось.

Обратите внимание, что позиционирование на первую запись множества не осуществляется – нужно вызвать метод OleDbDataReader.Read.

После завершения работы с объектом OleDbDataReader, желательно вызывать его метод Close или Dispose.

```
static void Test_018()
{
    const string c_sql="select country from country";

    const string c_cn_str
        ="provider=LCPI.IBProvider.3;\n"

        +"location=localhost:d:\\database\\employee.fdb;\n"
        "
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";

    using (var cn=new OleDbConnection(c_cn_str))
```

```

{
    cn.Open();

    using(var tr=cn.BeginTransaction())
    {
        using(var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            using(var reader=cmd.ExecuteReader())
            {
                int n=0;

                while(reader.Read())
                {
                    ++n;

                    if(n>1)
                        Console.Write(", ");

                    Console.Write("{0}",reader.GetString(0));
                }//while

                Console.WriteLine("");
            }//using reader
        }//using cmd

        tr.Commit();
    }//using tr
} //using cn
}

```



Рис. 11.

Проверить доступность записей можно проверить с помощью свойства `OleDbDataReader.HasRows`:

```
static void Test_019__HasRows()
{
    const string c_cn_str
        ="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";

    using(var cn=new OleDbConnection(c_cn_str))
    {
        cn.Open();

        using(var tr=cn.BeginTransaction())
        {
            using(var cmd=new OleDbCommand("select country
                from country",cn,tr))
            {
                using(var reader=cmd.ExecuteReader())
                {
                    Console.WriteLine("1. HasRows:
                        {0}",reader.HasRows);
                }//using reader
            }//using cmd

            using(var cmd=new OleDbCommand("select country
                from country where 1=0",cn,tr))
            {
                using(var reader=cmd.ExecuteReader())
                {
                    Console.WriteLine("2. HasRows:
```

```

        {0}", reader.HasRows);
    } //using reader
} //using cmd

    tr.Commit();
} //using tr
} //using cn
}

```



```

1. HasRows: True
2. HasRows: False

```

Рис. 12.

С помощью `ExecuteReader` можно выполнять запросы, которые не возвращают результирующее множество. В этом случае все равно возвращается объект `OleDbDataReader`. Единственным осмысленным использованием этого объекта будет чтение значения свойства `OleDbDataReader.RecordsAffected`. В нем будет указано количество записей, затронутых запросом.

```

static void Test_020()
{
    const string c_sql="update COUNTRY
        set CURRENCY=upper(CURRENCY) where 1=0";

    const string c_cn_str
        ="provider=LCPI.IBProvider.3;\n"
        +"location=localhost"
        +":d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";
}

```

```

using (var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();

    using (var tr=cn.BeginTransaction())
    {
        using (var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            using (var reader=cmd.ExecuteReader())
            {
                Console.WriteLine("RecordsAffected:
                    {0}",reader.RecordsAffected);
            } //using reader
        } //using cmd

        tr.Rollback();
    } //using tr
} //using cn
}

```



Рис. 13.

ExecuteNonQuery

`OleDbCommand.ExecuteNonQuery` предназначен для выполнения запросов, которые не возвращают результирующее множество – insert, update, delete, DDL запросы. Метод возвращает количество рядов, затронутых командой.

```

static void Test_021()
{
    const string c_sql ="insert into COUNTRY (COUNTRY,
        CURRENCY) values ('Mars', 'Snickers')";

```

```

const string c_cn_str
    ="provider=LCPI.IBProvider.3;\n"
    +"location=localhost: "
    +"d:\\database\\employee.fdb;\n"
    +"user id=SYSDBA;\n"
    +"password=masterkey;\n"
    +"dbclient_library=fbclient.dll";

using (var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();

    using (var tr=cn.BeginTransaction())
    {
        using (var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            Console.WriteLine("RecordsAffected:
                {0}",cmd.ExecuteNonQuery());
        } //using cmd

        tr.Rollback();
    } //using tr
} //using cn
} //Test_021

```

Команды с параметрами

В реальной работе с базой данных, как правило, используются запросы с параметрами. Это позволяет однократно подготовить запрос и многократно выполнять его для разных наборов значений. [ADO.NET](#) провайдер в паре с [IBProvider](#)-ом предоставляют полную поддержку для параметризованных запросов:

- Можно использовать именованные и неименованные (позиционные) параметры.
- Можно самостоятельно формировать описания параметров и генерировать их автоматически.

- Поддерживаются IN, OUT и IN/OUT параметры.
- Предоставлена возможность конфигурирования типа OUT-значений параметров.
- Поддерживается использование параметров в скриптах (команда с несколькими SQL-запросами).

Для работы с параметрами на уровне класса OleDbCommand определено свойство Parameters. Это свойство возвращает объект класса OleDbParameterCollection, обслуживающий коллекцию параметров команды. Кроме того, по аналогии с классическим ADODB, на уровне того же класса OleDbCommand реализованы два нестандартных «индексатора» для доступа к параметрам по имени и по целочисленному индексу.

Неименованные параметры в тексте запроса обозначаются маркером “?”:

```
insert into COUNTRY (COUNTRY, CURRENCY) values( ?, ?);
```

В следующей паре примеров будут продемонстрированы прямолинейные способы выполнения этого запроса с самостоятельным формированием описаний параметров запроса:

```
static void Test_022a()
{
    const string c_sql ="insert into COUNTRY (COUNTRY,
        CURRENCY) values(?, ?)";

    const string c_cn_str
        ="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";
```

```

using(var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();

    using(var tr=cn.BeginTransaction())
    {
        using(var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            cmd.Parameters.Add(new OleDbParameter, null,
                OleDbType.VarWChar, 0,
                ParameterDirection.Input)).Value="Mars";

            cmd.Parameters.Add(new OleDbParameternull,
                OleDbType.VarWChar, 0,
                ParameterDirection.Input)).Value="Snickers";
            Console.WriteLine("RecordsAffected:
                {0}",cmd.ExecuteNonQuery());
        }//using cmd

        tr.Rollback();
    }//using tr
} //using cn
} //Test_022a

static void Test_022b()
{
    const string c_sql
        ="insert into COUNTRY (COUNTRY,
            CURRENCY) values(?, ?)";

    const string c_cn_str
        ="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"

```

```

+ "password=masterkey;\n"
+ "dbclient_library=fbclient.dll";

using (var cn = new OleDbConnection(c_cn_str))
{
    cn.Open();

    using (var tr = cn.BeginTransaction())
    {
        using (var cmd = new OleDbCommand(c_sql, cn, tr))
        {
            cmd.Parameters.Add(null, OleDbType.VarWChar,
                                0, ParameterDirection.Input).Value = "Mars";

            cmd.Parameters.Add(null, OleDbType.VarWChar,
                                0, ParameterDirection.Input).Value = "Snickers";

            Console.WriteLine("RecordsAffected:
                               {0}", cmd.ExecuteNonQuery());
        } // using cmd

        tr.Rollback();
    } // using tr
} // using cn
} // Test_022b

```

Этот код можно сократить, заставив ADO.NET провайдер самостоятельно определить тип параметров и (по-умолчанию) указывать IN-направление:

```

static void Test_023()
{
    const string c_sql
        = "insert into COUNTRY (COUNTRY,
                                CURRENCY) values(?, ?)";
}

```

```

const string c_cn_str="provider=LCPI.IBProvider.3;\n"
+"location=localhost: "
+"d:\\database\\employee.fdb;\n"
+"user id=SYSDBA;\n"
+"password=masterkey;\n"
+"dbclient_library=fbclient.dll";

using (var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();

    using (var tr=cn.BeginTransaction())
    {
        using (var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            cmd.Parameters.AddWithValue(null,"Mars");
            cmd.Parameters.AddWithValue(null,"Snickers");

            Console.WriteLine("RecordsAffected:
                {0}",cmd.ExecuteNonQuery());
        } //using cmd

        tr.Rollback();
    } //using tr
    } //using cn
} //Test_023

```

OLE DB провайдер может предоставить описания параметров. Для получения сгенерированных описаний параметров необходимо вызвать метод `OleDbParameterCollection.Refresh`:

```

static void Test_024()
{
    const string c_sql ="insert into COUNTRY (COUNTRY,
        CURRENCY) values(?, ?)";
}

```



```

const string c_cn_str="provider=LCPI.IBProvider.3;\n"
    +"location=localhost: "
    +"d:\\database\\employee.fdb;\n"
    +"user id=SYSDBA;\n"
    +"password=masterkey;\n"
    +"dbclient_library=fbclient.dll";

using (var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();

    using (var tr=cn.BeginTransaction())
    {
        using (var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            cmd.Parameters.Refresh();

            cmd.Parameters[0].Value="Mars";

            cmd.Parameters[1].Value="Snickers";

            Console.WriteLine("RecordsAffected:
                {0}",cmd.ExecuteNonQuery());
        } //using cmd

        tr.Rollback();
    } //using tr
} //using cn
} //Test_024

```

Ну и последний способ – это просто присвоить значения через «индексаторы» параметров команды. OleDbCommand самостоятельно запросит описания параметров у OLE DB провайдера:

```

static void Test_025()
{
    const string c_sql ="insert into COUNTRY (COUNTRY,
        CURRENCY) values(?, ?)";

    const string c_cn_str="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";

    using(var cn=new OleDbConnection(c_cn_str))
    {
        cn.Open();

        using(var tr=cn.BeginTransaction())
        {
            using(var cmd=new OleDbCommand(c_sql,cn,tr))
            {
                cmd[0].Value="Mars";

                cmd[1].Value="Snickers";

                Console.WriteLine("RecordsAffected:
                    {0}",cmd.ExecuteNonQuery());
            }//using cmd

            tr.Rollback();
        }//using tr
    }//using cn
} //Test_025

```

С неименованными (позиционными) параметрами есть одна серьезная проблема – их нужно определять строго в том порядке, в

каком они следуют в тексте команды. И, поскольку нет имени, не получится дважды использовать один и тот же параметр в запросе. Есть, конечно, небольшое преимущество – обращение к параметру по целочисленному индексу все-таки эффективнее обращения по его имени. Но в целом, гораздо надежнее давать параметрам имена.

Стандартный .NET провайдер для OLE DB из .NET Framework (System.Data.OleDb) реализует ограниченную поддержку именованных параметров.

Маркер именованного параметра представляет собой префикс и имя. По-умолчанию, в качестве префикса предлагается использовать двоеточие – ‘:’. Префикс можно поменять через свойство инициализации «named_param_prefix».

Как и неименованные параметры, именованные параметры можно определять самостоятельно или запрашивать у OLE DB провайдера. Перепишем последний пример с использованием именованных параметров. Для разнообразия, настроим IBProvider для использования префикса ‘@’ (в стиле MSSQL).

```
static void Test_026()
{
    const string c_sql = "insert into COUNTRY (COUNTRY,
        CURRENCY) values (@country, @currency)";

    const string c_cn_str
        = "provider=LCPI.IBProvider.3;\n"
        + "location=localhost: "
        + "d:\\database\\employee.fdb;\n"
        + "user id=SYSDBA;\n"
        + "password=masterkey;\n"
        + "dbclient_library=fbclient.dll;"
        + "named_param_prefix='@'";

    using (var cn = new OleDbConnection(c_cn_str))
    {
        cn.Open();
    }
}
```

```

using(var tr=cn.BeginTransaction())
{
    using(var cmd=new OleDbCommand(c_sql,cn,tr))
    {
        cmd["country"].Value="Mars";

        cmd["currency"].Value="Snickers";

        Console.WriteLine("RecordsAffected:
            {0}",cmd.ExecuteNonQuery());
    }//using cmd

    tr.Rollback();
} //using tr
} //using cn
} //Test_026

```

Если необходимо сохранить префикс параметра в его имени, то в строке подключения нужно указать «named_param_rules=1»:

```

static void Test_027()
{
    const string c_sql="insert into COUNTRY (COUNTRY,
        CURRENCY) values(@country, @currency)";

    const string c_cn_str
        ="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll;"
        +"named_param_prefix='@';"
        +"named_param_rules=1";
}

```

```

using (var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();

    using (var tr=cn.BeginTransaction())
    {
        using (var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            cmd["@country"].Value="Mars";

            cmd["@currency"].Value="Snickers";

            Console.WriteLine("RecordsAffected:
                {0}",cmd.ExecuteNonQuery());
        } //using cmd

        tr.Rollback();
    } //using tr
} //using cn
} //Test_027

```

В предыдущих примерах рассматривались запросы с IN-параметрами, которые передают свои значения серверу базы данных. Есть еще OUT-параметры, через которые получают результат работы запроса.

OUT-параметры присутствуют не только в запросах для выполнения хранимых процедур, но и в запросах с секцией «RETURNING». Например, запрос «INSERT ... RETURNING ...» позволяет перечитать и получить в OUT-параметрах значения колонок добавленной записи.

В «INSERT ... RETURNING ...» имена OUT-параметров назначаются сервером и, скорее всего, будут совпадать с именами перечитываемых колонок. Это не всегда удобно, поэтому IBProvider расширяет этот запрос до конструкции «INSERT ... RETURNING ...

INTO ...», предоставляя возможность явного указания имен OUT-параметров для возвращаемых значений.

Рассмотрим использование запроса «INSERT ... RETURNING ... INTO ...» на примере добавления новой записи в таблицу EMPLOYEE стандартной базы данных employee.fdb. У этой таблицы есть триггер «BEFORE INSERT», генерирующий значение первичного ключа. Вот это значение и будет получено через OUT-параметр:

```
static void Test_insert_returning_into()
{
    const string c_cn_str
        ="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";

    using(var cn=new OleDbConnection(c_cn_str))
    {
        cn.Open();

        using(var tr=cn.BeginTransaction())
        {
            const string c_sql="insert into EMPLOYEE
                (FIRST_NAME, LAST_NAME, HIRE_DATE, DEPT_NO,
                JOB_CODE, JOB_GRADE, JOB_COUNTRY, SALARY)\n"
            +"values (:FIRST_NAME, :LAST_NAME, :HIRE_DATE,
                :DEPT_NO, :JOB_CODE, :JOB_GRADE,
                :JOB_COUNTRY, :SALARY)\n"
            +"returning EMP_NO into :NEW_ID";

            using(var cmd=new OleDbCommand(c_sql,cn,tr))
            {
                cmd["first_name"].Value  ="Agent";
```

```

        cmd["last_name"].Value      ="Smith";
        cmd["hire_date"].Value      =DateTime.Now;
        cmd["dept_no"].Value        ="000";
        cmd["job_code"].Value        ="CEO";
        cmd["job_grade"].Value       =1;
        cmd["job_country"].Value     ="USA";
        cmd["salary"].Value         =200001;

        cmd.ExecuteNonQuery();

        Console.WriteLine("NEW_ID: {0}. Direction: {1}.",
            cmd["NEW_ID"].Value, cmd["NEW_ID"].Direction);
    } //using cmd

    tr.Rollback();
} //using tr
} //using cn
} //Test_insert_returning_into

```



Рис. 14.

Следующий пример практически полностью идентичен предыдущему. Отличие заключается в использовании IN/OUT параметра «EMP_NO». На входе указывается NULL-значение (DBNull.Value). А на выходе в этом параметре будет сгенерированное значение колонки «EMP_NO».

```

static void Test_in_out_param()
{
    const string c_cn_str="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"

```

```

+ "password=masterkey;\n"
+ "dbclient_library=fbclient.dll";

using (var cn = new OleDbConnection(c_cn_str))
{
    cn.Open();

    using (var tr = cn.BeginTransaction())
    {
        const string c_sql = "insert into EMPLOYEE
            (EMP_NO, FIRST_NAME, LAST_NAME, HIRE_DATE,
            DEPT_NO, JOB_CODE, JOB_GRADE, JOB_COUNTRY,
            SALARY) \n"
        + "values (:EMP_NO, :FIRST_NAME, :LAST_NAME,
            :HIRE_DATE, :DEPT_NO, :JOB_CODE, :JOB_GRADE,
            :JOB_COUNTRY, :SALARY) \n"
        + "returning EMP_NO into :EMP_NO";

        using (var cmd = new OleDbCommand(c_sql, cn, tr))
        {
            cmd["emp_no"].Value = DBNull.Value;
            cmd["first_name"].Value = "Agent";
            cmd["last_name"].Value = "Smith";
            cmd["hire_date"].Value = DateTime.Now;
            cmd["dept_no"].Value = "000";
            cmd["job_code"].Value = "CEO";
            cmd["job_grade"].Value = 1;
            cmd["job_country"].Value = "USA";
            cmd["salary"].Value = 200001;

            cmd.ExecuteNonQuery();

            Console.WriteLine("EMP_NO: {0}. Direction: {1}.",
                cmd["emp_no"].Value, cmd["emp_no"].Direction);
        } //using cmd
    }
}

```



```

        tr.Rollback();
    }//using tr
} //using cn
} //Test_in_out_param

```



```

EMP_NO: 157. Direction: InputOutput.

```

Рис. 15.

По умолчанию, тип возвращаемого значения определяется типом самого параметра. К примеру, текстовый блок будет возвращаться в виде строки (System.String). Однако это можно изменить и попросить ADO.NET провайдер вместо строки возвращать объект для потокового чтения данных (System.IO.TextReader).

```

static void Test_change_out_param_value_type()
{
    const string c_cn_str="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";

    using(var cn=new OleDbConnection(c_cn_str))
    {
        cn.Open();

        using(var tr=cn.BeginTransaction())
        {
            const string c_sql
                ="update PROJECT set "
                +"PROJ_DESC=UPPER(PROJ_DESC)\n"
                +"where PROJ_ID=:id\n"
                +"returning PROJ_DESC\n"

```

```

        +"into :desc";

        using(var cmd=new OleDbCommand(c_sql,cn,tr))
        {
            cmd["id"].Value="DGPII";

            cmd["desc"]
                .OutputBinding.Set(OleDbType.IUnknown,
                    typeof(System.IO.TextReader));
            cmd.ExecuteNonQuery();

            var v=cmd["desc"].Value;

            Console.WriteLine("type: {0}",v.GetType());

            Console.WriteLine("data: {0}",
                ((System.IO.TextReader)v).ReadToEnd());
        }//using cmd

        tr.Rollback();
    }
} //using cn
} //Test_change_out_param_value_type

```

```

type: lcp1.data.oledb.OleDbTextReader
data: DEVELOP SECOND GENERATION DIGITAL PIZZA MAKER
WITH FLASH-BAKE HEATING ELEMENT AND
DIGITAL INGREDIENT MEASURING SYSTEM.
=

```

Рис. 16.

В представленном примере, выполняется конфигурирование OUT-значения параметра «desc»:

```
cmd["desc"].OutputBinding.Set(OleDbType.IUnknown,
    typeof(System.IO.TextReader));
```

Первый аргумент метода Set указывает OLE DB тип значения, которое должен вернуть OLE DB провайдер. В данном случае запрашивается значение в виде COM-объекта.

Второй аргумент указывает OLE DB провайдеру, что возвращаемый COM-объект должен предоставить интерфейс для загрузки текста. ADO.NET провайдер, в свою очередь, использует этот аргумент для определения типа .NET объекта для обслуживания этого COM-объекта — `lcpi.data.oledb.OleDbTextReader`.

На самом деле, конечно, в `IBProvider` передается не «`typeof(System.IO.TextReader)`», а идентификатор COM-интерфейса. Конкретно в данном случае это будет `IID_IBP_SequentialStream_WideChar` – идентификатор нестандартного интерфейса для потоковой загрузки текстовых данных в виде двухбайтных UNICODE-символов.

К сожалению, в спецификации OLE DB отсутствует стандартный интерфейс для подобной задачи. Но в ней не запрещает для этих целей определять и использовать свои собственные интерфейсы.

Аналогичным способом можно получать OUT-значений бинарных блобов в виде потока байт:

```
cmd["binary_blob_out_param"].OutputBinding.Set(OleDbType.IUnknown,
typeof(System.IO.Stream));
```

В этом случае, у OLE DB провайдера будет запрошен COM-объект со стандартным интерфейсом `ISequentialStream`. А ADO.NET провайдер создаст объект класса `lcpi.data.oledb.OleDbStream`.

У `OleDbParameter.OutputBinding` (это свойство возвращает объект класса `OleDbValueBinding`) есть и другие варианты метода `Set`, некоторые из которых разрешают явно указывать идентификатор COM-интерфейса. Это позволяет достаточно гибко настраивать представления значений OUT-параметров под конкретные задачи.

Выполнение скриптов

[IBProvider](#) и [ADO.NET](#) провайдер поддерживают выполнение скриптов — команд с несколькими SQL-запросами. Запросы могут содержать именованные параметры (автоматическая генерация описаний параметров не поддерживается). Скрипт может управлять транзакцией и выполнять DDL-запросы.

Для перебора результатов выполнения скрипта нужно использовать метод `OleDbDataReader.NextResult`.

Рассмотрим выполнение скрипта, в котором:

- Создается таблица
- Добавляются записи в новую таблицу
- Делается выборка этих записей двумя разными запросами
- Удаляется таблица

Скрипт содержит IN-параметры (`data1`, `data2`, `data3`) и OUT-параметры (`id1`, `id2`, `id3`).

```
/*
    set transaction;

    SET AUTODDL ON; -- неявно коммитим DDL запросы

    create table TTT (id integer not null primary key,
                      data varchar(10));

    create generator GEN_ID_TTT;

    create trigger BI_TTT for TTT before insert as
        begin NEW.ID=GEN_ID(GEN_ID_TTT,1); end;

    insert into TTT (data) (:data1) returning ID into :id1;
    insert into TTT (data) (:data2) returning ID into :id2;
    insert into TTT (data) (:data3) returning ID into :id3;

    select ID,DATA from TTT order by ID;

    drop table TTT;

    drop generator GEN_ID_TTT;

    commit;
*/
```

```

static void Test__script()
{
    const string c_cn_str="provider=LCPI.IBProvider.3;\n"
        +"location=localhost: "
        +"d:\\database\\employee.fdb;\n"
        +"user id=SYSDBA;\n"
        +"password=masterkey;\n"
        +"dbclient_library=fbclient.dll";

    using (var cn=new OleDbConnection(c_cn_str))
    {
        cn.Open();

        var cmd=cn.CreateCommand();

        cmd.CommandText=
            "set transaction;\n"
            +"\n"
            +"SET AUTODDL ON; -- неявно коммитим DDL \n"
            +"\n"
            +"create table TTT (id integer
                not null primary key,\n"
            +"data varchar(10));\n"
            +"\n"
            +"create generator GEN_ID_TTT;\n"
            +"\n"
            +"create trigger BI_TTT for TTT before insert as
                begin NEW.ID=GEN_ID(GEN_ID_TTT,1); end;\n"
            +"\n"
            +"insert into TTT (data) values(:data1)
                returning ID into :id1;\n"
            +"insert into TTT (data) values(:data2)
                returning ID into :id2;\n"
            +"insert into TTT (data) values(:data3)
                returning ID into :id3;\n"

```

```

+"\n"
+"select ID,DATA from TTT order by ID asc;\n"
+"select ID,DATA from TTT order by ID desc;\n"
+"\n"
+"drop table TTT;\n"
+"\n"
+"drop generator GEN_ID_TTT;"
+"\n"
+"commit;";

```

```

//----- IN-parameters
cmd.Parameters.AddWithValue("data1","QWERTY");
cmd.Parameters.AddWithValue("data2","ASDFGH");
cmd.Parameters.AddWithValue("data3","ZXCVCBN");

```

```

//----- OUT-parameters
cmd.Parameters.Add("id1",OleDbType.Variant, 0,
    ParameterDirection.Output);
cmd.Parameters.Add("id2",OleDbType.Variant, 0,
    ParameterDirection.Output);
cmd.Parameters.Add("id3",OleDbType.Variant, 0,
    ParameterDirection.Output);

```

```

//-----
var reader=cmd.ExecuteReader();

```

```

//-----
Console.WriteLine("id1: {0}",cmd["id1"].Value);
Console.WriteLine("id2: {0}",cmd["id2"].Value);
Console.WriteLine("id3: {0}",cmd["id3"].Value);

```

```

//-----
for(int n=0;;)
{
    ++n;
}

```

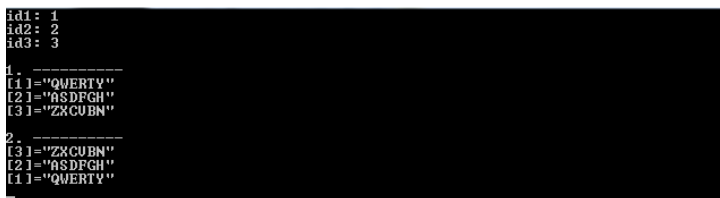
```

Console.WriteLine("");
Console.WriteLine("{0}. -----", n);

while (reader.Read())
{
    Console.WriteLine("[{0}]=\"{1}\"",
        reader["ID"], reader["DATA"]);
}

if (!reader.NextResult())
    break;
} //for n
} //using cn
} //Test__script

```



```

id1: 1
id2: 2
id3: 3

1. -----
[1]="QWERTY"
[2]="ASDFGH"
[3]="ZXCVBN"

2. -----
[3]="ZXCVBN"
[2]="ASDFGH"
[1]="QWERTY"

```

Рис. 17.

Отмена выполнения запроса

Отмена выполнения запроса выполняется с помощью метода `OleDbCommand.Cancel`. Вызов `OleDbCommand.Cancel` должен выполняться в отдельном потоке, потому что методы выполнения команды (`ExecuteScalar`, `ExecuteReader`, `ExecuteNonQuery`) блокируют текущий поток до завершения операции.

В следующем примере в отдельном потоке выполняется запрос с вызовом хранимой процедуры, а основной поток отменяет выполнение этого запроса.

```

/*

create procedure SP_EXEC_DUMMY_COUNTER(n integer)
as
  declare variable i integer;
begin
  i=0;

  while (i<n) do
  begin
    i=i+1;
  end
end;

*/

//-----
-----

class ThreadWorker
{
  private readonly OleDbCommand m_cmd;

  public Exception m_exc=null;

//-----
-----

  public ThreadWorker(OleDbCommand cmd)
  {
    m_cmd=cmd;
  } //ThreadWorker

//-----
-----

```



```

public void ExecuteNonQuery()
{
    Console.WriteLine("Enter to
        ThreadWorker.ExecuteNonQuery");

    try
    {
        m_cmd.ExecuteNonQuery();
    }
    catch(Exception e)
    {
        Console.WriteLine("Catch exception in
            ThreadWorker.ExecuteNonQuery");

        m_exc=e;
    } //catch

    Console.WriteLine("Exit from
        ThreadWorker.ExecuteNonQuery");
} //ExecuteNonQuery
}; //class ThreadWorker

//-----
static void Test__cmd_cancel()
{
    const string c_sql_create_sp="create procedure
        SP_EXEC_DUMMY_COUNTER(n integer)\n"
        +"as\n"
        +" declare variable i integer;\n"
        +"begin\n"
        +" i=0;\n"
        +" \n"
        +" while(i<n)do\n"
        +" begin\n"
        +" i=i+1;\n"

```

```
+ " end\n"
+"end;";
```

```
const string c_cn_str="provider=LCPI.IBProvider.3;\n"
+"location=localhost: "
+"d:\\database\\employee.fdb;\n"
+"user id=SYSDBA;\n"
+"password=masterkey;\n"
+"dbclient_library=fbclient.dll";
```

```
using(var cn=new OleDbConnection(c_cn_str))
{
    cn.Open();
```

```
using(var tr=cn.BeginTransaction())
{
    if(cn.GetOleDbSchemaTable(
        OleDbSchemaGuid.Procedures,
        new object[]{null,
            null,
            "SP_EXEC_DUMMY_COUNTER"}).Rows.Count==0)
    {
        using(var cmd=
            new OleDbCommand(c_sql_create_sp,cn,tr))
        {
            cmd.ExecuteNonQuery();
        }
```

```
        tr.CommitRetaining();
    } //if
```

```
using(var cmd=new OleDbCommand(null,cn,tr))
{
    cmd.CommandText="execute procedure
        SP_EXEC_DUMMY_COUNTER(100000000)";
```

```

var threadWorker=new ThreadWorker(cmd);
var thread=
    new System.Threading
        .Thread(threadWorker.ExecuteNonQuery);

try
{
    thread.Start();

    while(thread.IsAlive)
    {
        System.Threading.Thread.Sleep(2000);

        Console.WriteLine("Cancel");

        cmd.Cancel();
    }//while

    Console.WriteLine("threadWorker was
        stopped");

    Console.WriteLine("");

    if(Object.ReferenceEquals(
        threadWorker.m_exc,null))
    {
        Console.WriteLine("No exception");
    }
    else
    {
        Console.WriteLine("Thread exception:
            {0}\n\n{1}", threadWorker.m_exc.Source,
            threadWorker.m_exc.Message);
    }//else

```

```

    }
    finally
    {
        thread.Join();
    } //finally
} //using cmd
} //using tr
} //using cn
} //Test__cmd_cancel

```

```

Enter to ThreadWorker.ExecuteNonQuery
Cancel
Catch exception in ThreadWorker.ExecuteNonQuery
Exit from ThreadWorker.ExecuteNonQuery
Cancel
threadWorker was stopped

Thread exception: Multiple Sources
1. [LCPI.IBProvider.3] Ошибка выполнения SQL выражения.
operation was cancelled
Код ошибки COM: DB_E_CANCELED. SQL State: "HY008". Код ошибки DBMS: 335544794.
2. [lcpi.data.oledb.OleDbCommand] Выполнение запроса завершилось с ошибками. Порядковый номер запроса: 1.
Код ошибки COM: DB_E_CANCELED.

```

Рис. 18.

Поддержка типов данных

[IBProvider](#) предоставляет поддержку для всех типов данных Firebird/InterBase. А [ADO.NET](#) провайдер, используя возможности .NET Framework, максимально упрощает работу с ними на уровне прикладного кода.

Типы данных можно разделить на следующие группы:

- Целочисленные типы данных;
- Вещественные типы данных;
- Типы данных NUMERIC и DECIMAL;
- Строковые типы данных;
- Бинарные типы данных;
- Булевский тип;
- Типы данных даты и времени;
- Блобы;
- Массивы;

Целочисленные типы данных

К этой группе типов данных относятся типы SMALLINT, INTEGER, BIGINT.

Таблица 4

Тип данных FB/IB	Размер в байтах	Тип данных OLE DB	System.Data.DbType	lcpi.data.oledb.OleDbType	Тип данных .NET Framework
SMALLINT	2	DBTYPE_INT2	DbType.Int16	OleDbType.Smallint	System.Int16
INTEGER	4	DBTYPE_INT4	DbType.Int32	OleDbType.Integer	System.Int32
BIGINT	8	DBTYPE_INT8	DbType.Int64	OleDbType.BigInt	System.Int64

Вещественные типы данных

К этой группе типов данных относятся типы FLOAT, DOUBLE PRECISION.

Таблица 5

Тип данных FB/IB	Размер в байтах	Тип данных OLE DB	System.Data.DbType	lcpi.data.oledb.OleDbType	Тип данных .NET Framework
FLOAT	4	DBTYPE_FLOAT	DbType.Single	OleDbType.Single	System.Single
DOUBLE PRECISION	8	DBTYPE_DOUBLE	DbType.Double	OleDbType.Double	System.Double

Типы данных NUMERIC и DECIMAL

Типы данных NUMERIC и DECIMAL, в общем случае, не различаются и обрабатываются единообразно.

Таблица 5

Тип данных х FB/IB	Макс. Точно сть	Тип данных OLE DB	System.Data. DbType	lcpi.data.oledb.O leDbType	Тип данных .NET Framework
DECIMAL	18	DBTYPE_NUMERIC	DbType.Decimal	OleDbType.Numeric	System.Decimal
NUMERIC	18	DBTYPE_NUMERIC	DbType.Decimal	OleDbType.Numeric	System.Decimal

Строковые типы данных

К этой группе относятся типы данных CHAR, VARCHAR с кодовой страницей отличной от OCTETS.

С этими типами данных есть определенные сложности, потому что их представление и обработка зависит от настроек подключения и кодовой страницы самих данных.

По-умолчанию IBProvider работает в UNICODE-режиме (unicode_mode=true). Это означает что IBProvider предлагает обмениваться текстовыми данными с использованием двухбайтных UNICODE-символов.

В этом случае, для строковых данных с кодовой страницей отличной от NONE будут действовать следующие правила:

Таблица 6

Тип данных FB/IB	Тип данных OLE DB	System.Data.Db Type	lcpi.data.oledb.OleD bType	Тип данных .NET Framewo rk
CHAR	DBTYPE_W STR	DbType.String	OleDbType.WChar	System.Str ing
VARCH AR	DBTYPE_W STR	DbType.String	OleDbType.VarWCha r	System.Str ing

Если строковые данные имеют кодовую страницу NONE, то представление будет другим:

Таблица 7

Тип данных FB/IB	Тип данных OLE DB	System.Data.Db Type	lcpi.data.oledb.OleDb Type	Тип данных .NET Framewor k
CHAR	DBTYPE_S TR	DbType.AnsiStrin g	OleDbType.Char	System.Str ing
VARCH AR	DBTYPE_S TR	DbType.AnsiStrin g	OleDbType.VarChar	System.Str ing

По-умолчанию, преобразование таких «NONE-данных» в UNICODE и обратно осуществляется с использованием кодовой страницы ASCII.

Это можно поменять, указав в строке подключения параметр «`type_none`» с именем символьного набора отличным от NONE.

Максимальное количество символов в CHAR/VARCHAR колонках зависит от кодовой страницы. Для однобайтных кодировок эта величина составляет 32767 и 32765 для CHAR и VARCHAR соответственно.

Типы данных даты и времени

Таблица 8

Тип данных х FB/IB	Настройка	Тип данных OLE DB	System.Data. DbType	lcpi.data.oledb. OleDbDbType	Тип данных .NET Framework
TIMESTAMP	dbtimestamp_rules=0	DBTYPE_DBTIMESTAMP	DbType.DateTime2	OleDbDbType.DBTimeStamp	System.DateTime
DATE	dbdate_rules=0	DBTYPE_DBDATE	DbType.Date	OleDbDbType.DBDate	System.DateTime
TIME	dbtime_rules=0	DBTYPE_DBTIME	DbType.Time	OleDbDbType.DBTime	System.TimeSpan
TIME	dbtime_rules=1	DBTYPE_DBTIME2	DbType.Time	OleDbDbType.DBTime2	System.TimeSpan

В первоначальной версии OLE DB, тип для представления времени (DBTYPE_DBTIME) не предполагал хранение долей секунды. Это было исправлено в MSSQL 2008 – у него появился тип DBTYPE_DBTIME2. На текущий момент, по умолчанию для типа TIME используется старый формат представления времени (DBTYPE_TIME). При работе с IBProvider-ом через рассматриваемый ADO.NET провайдер рекомендуется указать в строке подключения «dbtime_rules=1».

Блобы

Блобы делятся на две группы – текстовые и бинарные.

К текстовым относятся blobs с подтипом (SUB_TYPE) TEXT (id: 1) и кодовой страницей отличной от OCTETS.

К бинарным относятся blobs с подтипом отличным от TEXT и blobs с подтипом TEXT и кодовой страницей OCTETS.

В приведенной ниже таблице предполагается что у текстового блага определена нормальная (не NONE) кодовая страница и подключение работает в режиме UNICODE (unicode_mode=true).

Таблица 9

Тип данных FB/IB	Тип данных OLE DB	System.Data.Db Type	lcpi.data.oledb.OleDb Type	Тип данных .NET Framework
BLOB (binary)	DBTYPE_BINARY	DbType.Binary	OleDbType.LongVar Binary	System.Byte[]
BLOB (text)	DBTYPE_WSTR	DbType.String	OleDbType.LongVar WChar	System.String

ADO.NET провайдер поддерживает все способы обработки блобов, включая потоковую обработку с использованием System.IO.Stream и System.IO.TextReader.

Методы OleDbDataReader для получения значений текстовых блобов:

- string GetString(int ordinal)
- TextReader GetTextReader(int ordinal)
- object GetValue(int ordinal)
- long GetChars(int ordinal, long dataOffset, char[] buffer, int bufferOffset, int length)

Методы OleDbDataReader для получения значений бинарных блобов:

- byte[] GetBytes(int ordinal)
- Stream GetStream(int ordinal)
- object GetValue(int ordinal)
- long GetBytes(int ordinal, long dataOffset, byte[] buffer, int bufferOffset, int length)

Методы для блочной загрузки содержимого блобов «long GetChars(...)» и «long GetBytes(...)», в свете наличия методов GetTextReader и GetStream, являются анахронизмом ранних версий ADO.NET.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать клиентское приложение для вывода, добавления, удаления и изменения данных в базе данных, разработанной в лабораторных работах.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Дайте определение технологии ADO.Net.
2. Перечислите основные пространства имен ADO.Net.
3. Дайте определение OLE DB провайдера.
4. Перечислите основные настройки подключения OLE DB провайдера.
5. Приведите пример использования компонента OleDbCommand.
6. Опишите работу со скриптами в ADO.Net.
7. Приведите пример параметризованного запроса.
8. Для чего используется метод ExecuteReader.
9. Опишите работу метода ExecuteScalar.

ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ

На выполнение домашней работы отводится 11 академических часов: 9 час на выполнение и сдачу домашней работы и 2 час на подготовку отчета).

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), модель базы данных, ход выполнения работы, результаты выполнения работы, выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Карпова, Т.С. Базы данных: модели, разработка, реализация [Электронный ресурс]: учебное пособие / Т.С. Карпова. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 241 с. : ил. ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=429003>
2. Советов, Б. Я. Базы данных [Электронный ресурс] : учебник для среднего профессионального образования / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2019. — 420 с. — (Профессиональное образование). —URL: <https://biblio-online.ru/bcode/438438>
3. Нестеров, С. А. Базы данных [Электронный ресурс]: учебник и практикум для среднего профессионального образования / С. А. Нестеров. — Москва : Издательство Юрайт, 2019. — 230 с. — (Профессиональное образование). — URL: <https://biblio-online.ru/bcode/445770>
4. Маркин, А.В. Построение запросов и программирование на SQL [Электронный ресурс]: учебное пособие / А.В. Маркин. - 3-е изд., перераб. и доп. - Москва : Диалог-МИФИ, 2014. - 384 с. : ил. - Библиогр.: с. 364-366 - URL: <http://biblioclub.ru/index.php?page=book&id=89077>

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1. Разработка приложений на C# с использованием СУБД PostgreSQL [Электронный ресурс]: учебное пособие / И.А. Васюткина, Г.В. Трошина, М.И. Бычков, С.А. Менжулин ; Министерство образования и науки Российской Федерации, Новосибирский государственный технический университет. - Новосибирск : НГТУ, 2015. - 143 с. : схем., табл., ил. - Библиогр. в кн. - URL: <http://biblioclub.ru/index.php?page=book&id=438432>.
2. Гуцин, А.Н. Базы данных[Электронный ресурс] : учебник / А.Н. Гуцин. - Москва : Директ-Медиа, 2014. - 266 с. : ил.,табл., схем. - ISBN 978-5-4458-5147-9 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=222149>

3. Щелоков, С.А. Базы данных [Электронный ресурс] : учебное пособие / С.А. Щелоков ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Оренбургский государственный университет», Кафедра программного обеспечения вычислительной техники и автоматизированных систем. - Оренбург : Оренбургский государственный университет, 2014. - 298 с. : ил. - Библиогр. в кн. - URL: <http://biblioclub.ru/index.php?page=book&id=260752>

Электронные ресурсы:

1. Научная электронная библиотека <http://eLIBRARY.RU>
2. Электронно-библиотечная система <http://e.lanbook.com>
3. Электронно-библиотечная система «Университетская библиотека онлайн» <http://biblioclub.ru>
4. Электронно-библиотечная система IPRBook
<http://www.iprbookshop.ru>