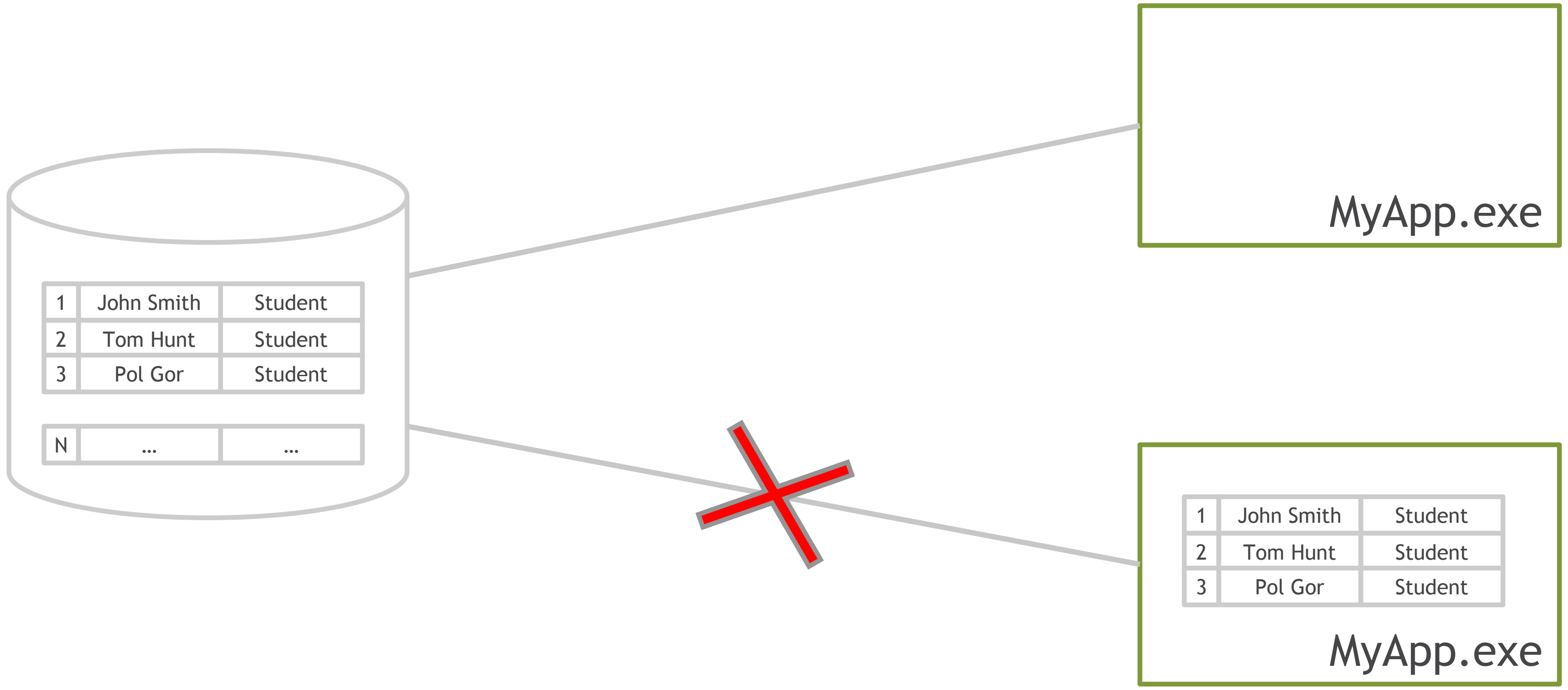
ADO.NET

Agenda

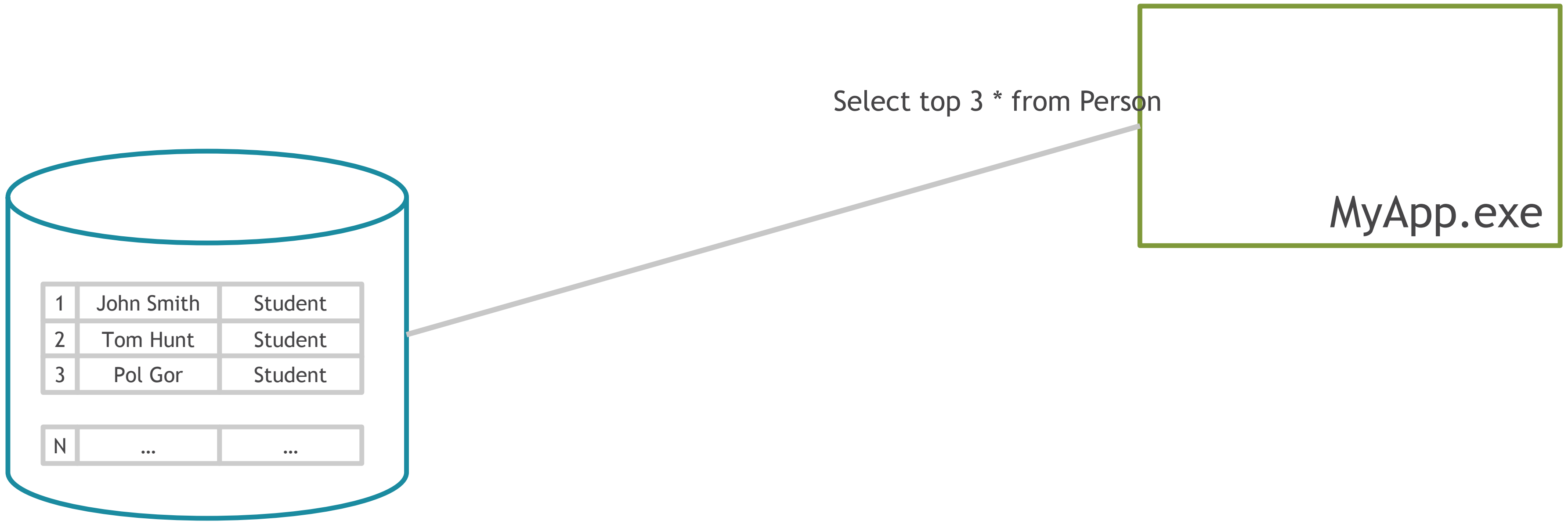
- ADO.Net Basic
- Connected model

ADO.NET BASIC

Connected and Disconnected models

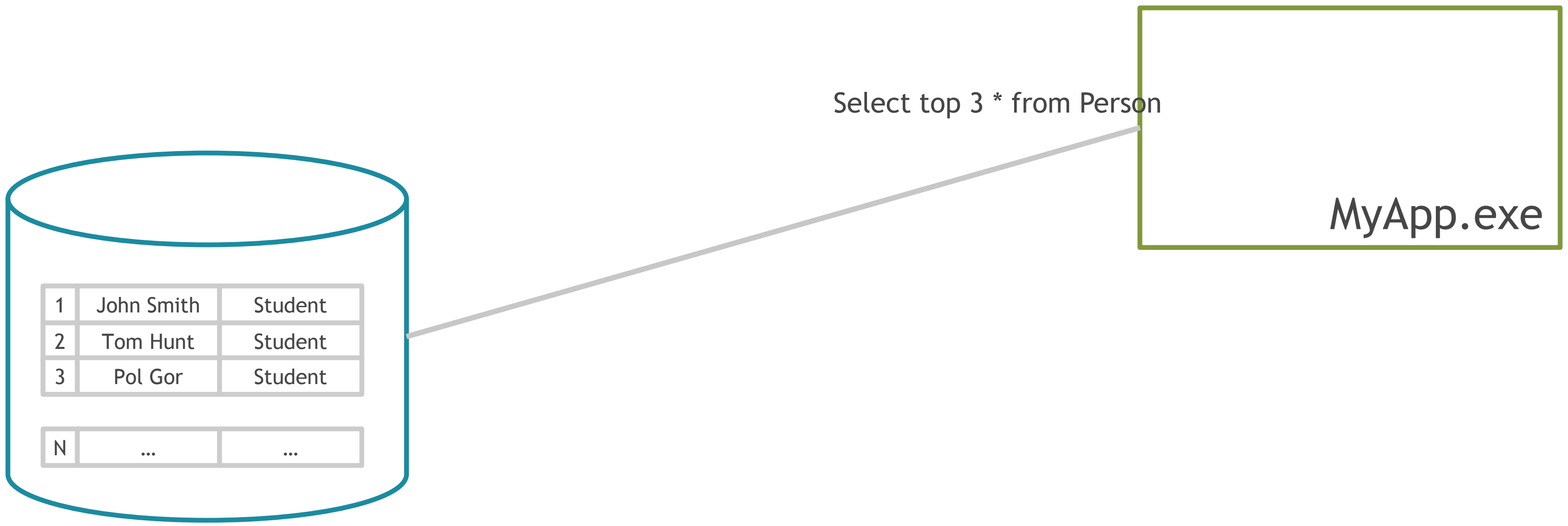


Connected model

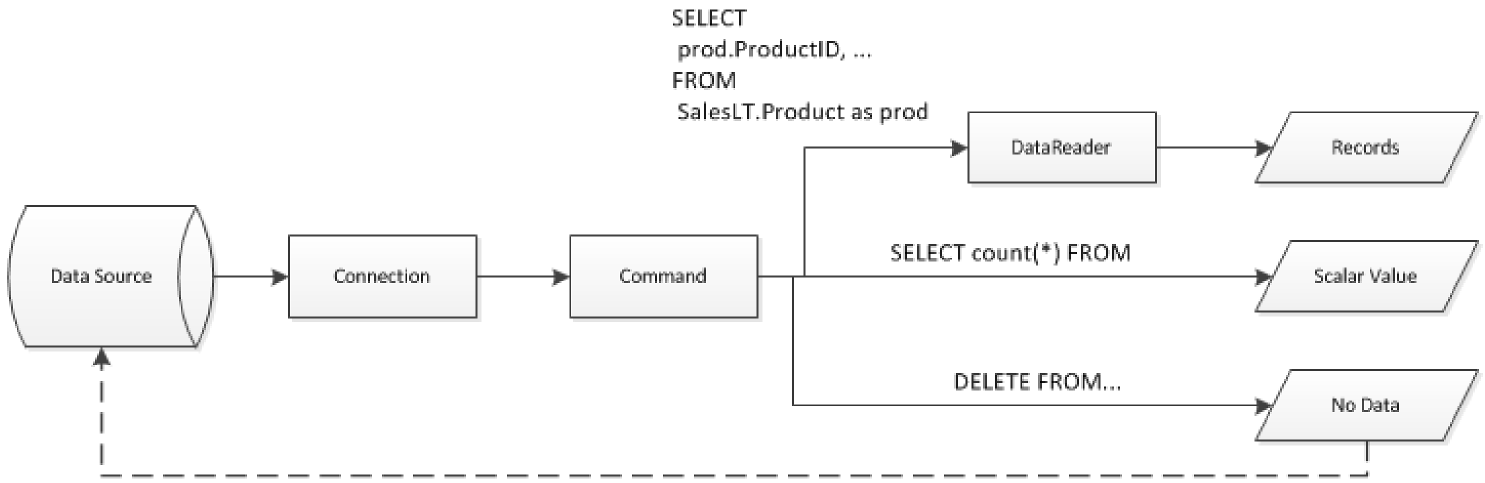


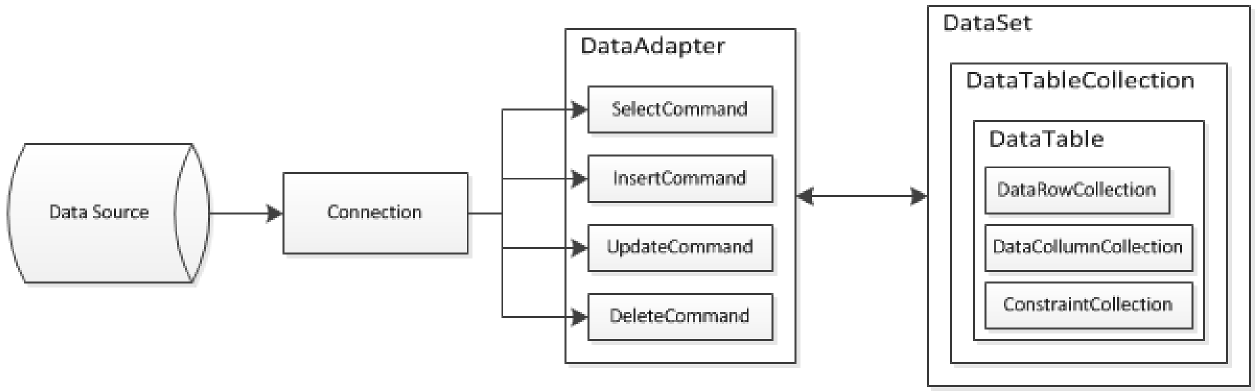
1	John Smith	Student
2	Tom Hunt	Student
3	Pol Gor	Student
N	...	...

Disconnected model



Common ADO.Net components



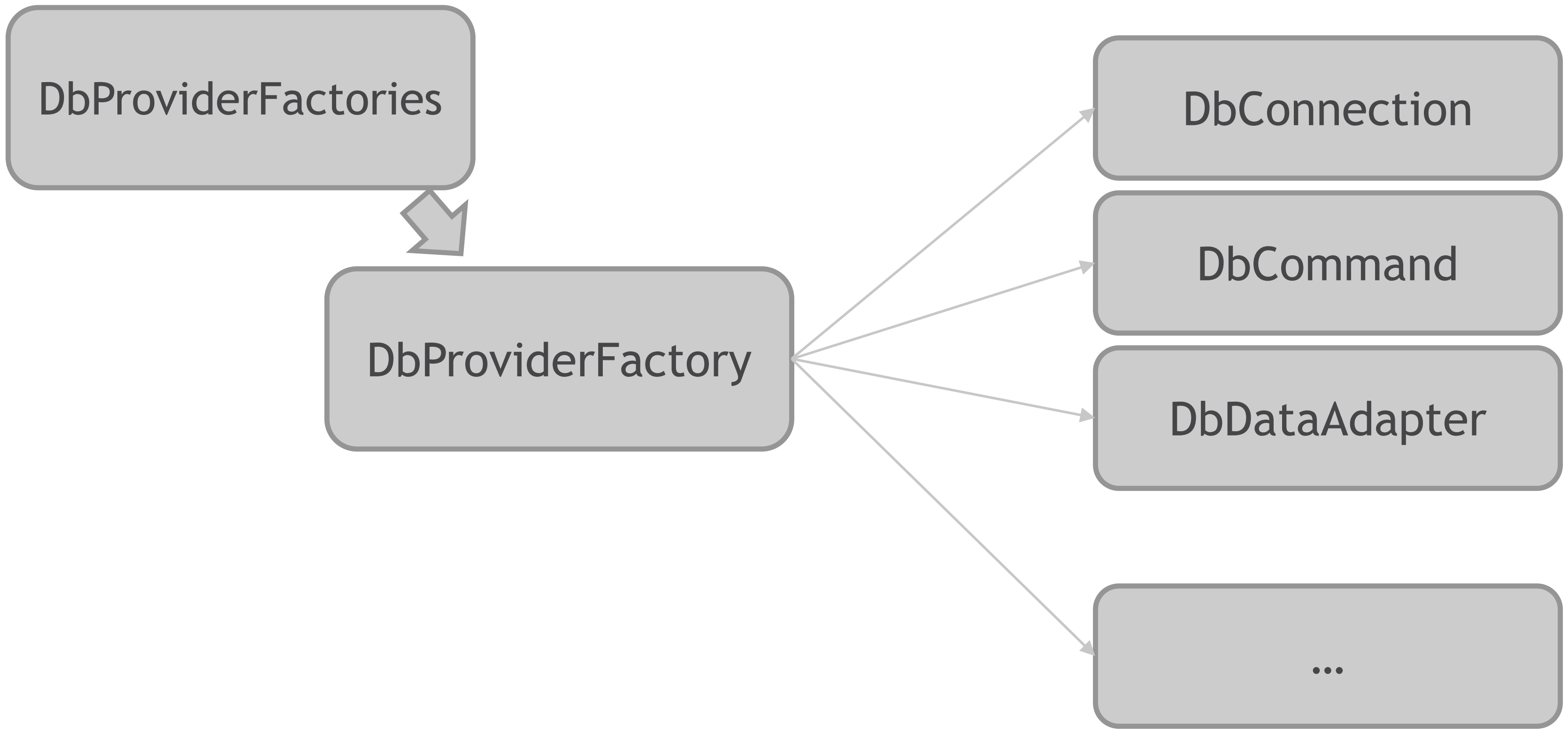


- Connection
- Command
- Transaction
- DataReader
- DataAdapter
- DataSet

ADO.Net provider model

Interfaces (System.Data)	Abstract classes (System.Data.Common)	Concrete providers
IDbConnection	DbConnection	**System.Data.EntityClient**.EntityConnection **System.Data.Odbc**.OdbcConnection **System.Data.OleDb**.OleDbConnection **System.Data.OracleClient**.OracleConnection **System.Data.SqlClient**.SqlConnection
IDbCommand	DbCommand	System.Data.EntityClient.EntityCommand System.Data.Odbc.OdbcCommand System.Data.OleDb.OleDbCommand System.Data.OracleClient.OracleCommand System.Data.SqlClient.SqlCommand
IDataReader	DbDataReader	System.Data.EntityClient.EntityDataReader …
IDbDataAdapter	DbDataAdapter	…
IDbTransaction	DbTransaction	…
…		

Provider invariant programming



Writing Provider-Independent Code in ADO.NET

Get current providers. Register custom

```
// Get drivers list as DataTable
DataTable drivers = DbProviderFactories.GetFactoryClasses();

foreach (DataRow driver in drivers.Rows)
{
    Console.WriteLine("{0} | {1} | {2} | {3}",
        driver["Name"],
        driver["Description"],
        driver["InvariantName"],
        driver["AssemblyQualifiedName"]);
}
```

Return standard and custom providers

```
<configSections>
  <section name="system.data" type="System.Data.Common.DbProviderFactoriesConfigurationHandler, System.Data" />
</configSections>
<system.data>
  <DbProviderFactories>
    <add name="Microsoft SQL Server Compact Data Provider 4.0"
         invariant="System.Data.SqlServerCe.4.0"
         description=".NET Framework Data Provider for Microsoft SQL Server Compact"
         type="System.Data.SqlServerCe.SqlCeProviderFactory, System.Data.SqlServerCe …"/>
  </DbProviderFactories>
</system.data>
```

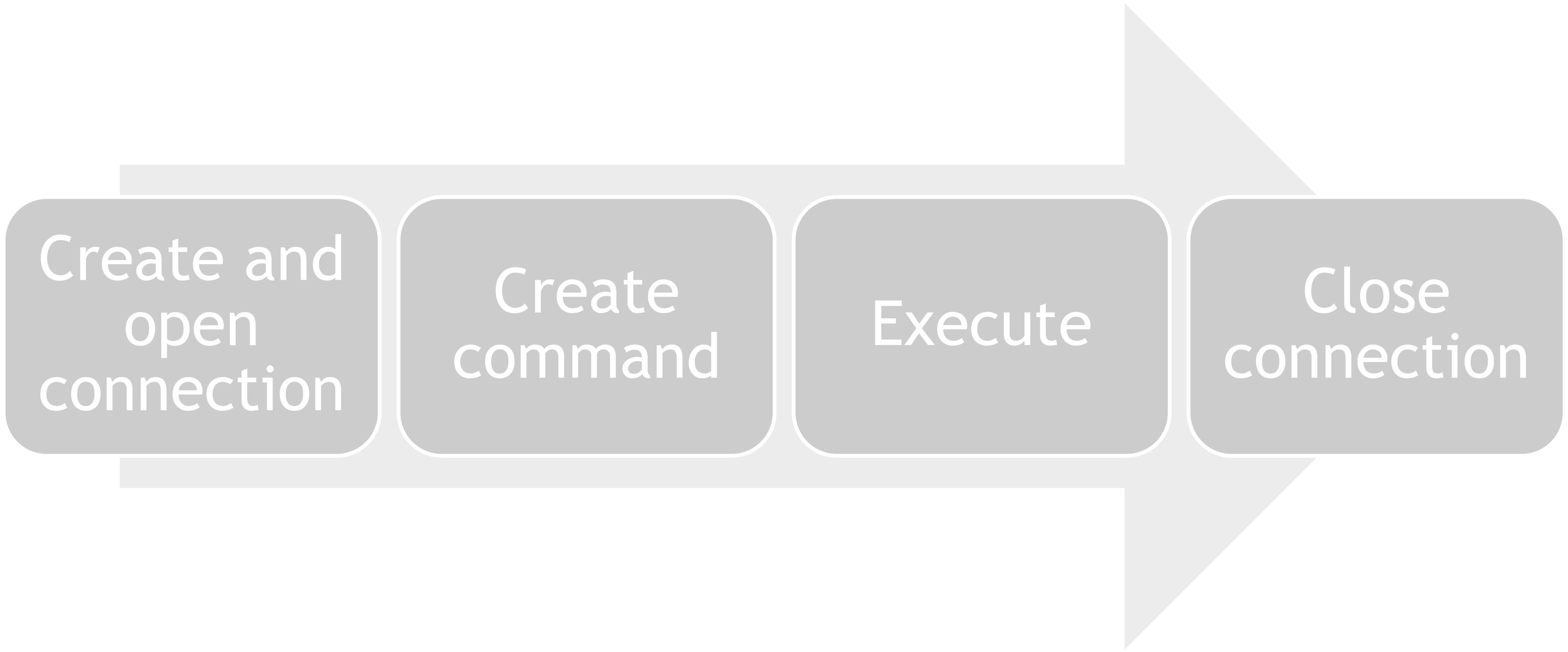
Work with Provider Factories

```
// Use invarinat name
DbProviderFactory providerFactory = DbProviderFactories.GetFactory("System.Data.SqlClient");

IDbConnection connection = providerFactory.CreateConnection();
IDbCommand command = providerFactory.CreateCommand();
IDbDataAdapter dataAdapter = providerFactory.CreateDataAdapter();
```

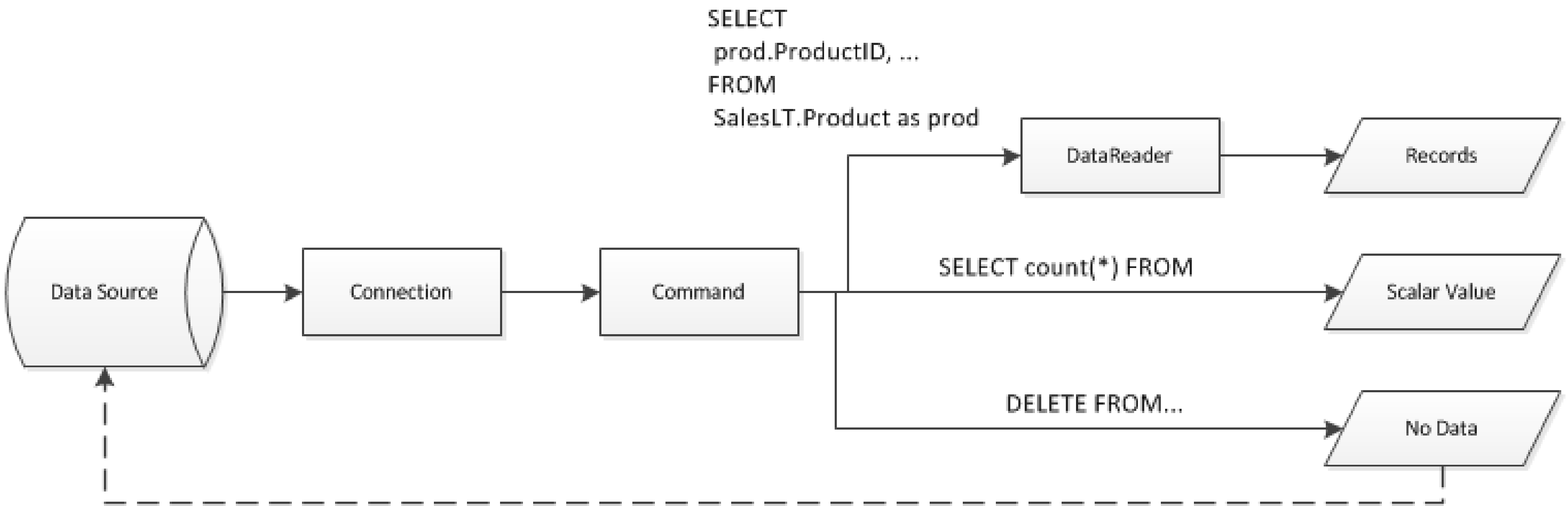
```
// Some object can create only through other
connection.ConnectionString = "Server=(local);Database=Northwind;Integrated Security=True";
connection.Open();
IDbTransaction transaction = connection.BeginTransaction();
```

CONNECTED MODEL

Base ADO.Net usage flow



Connected model components



- Connection
- Command
- DataReader

CONNECTED MODEL. CONNECTION

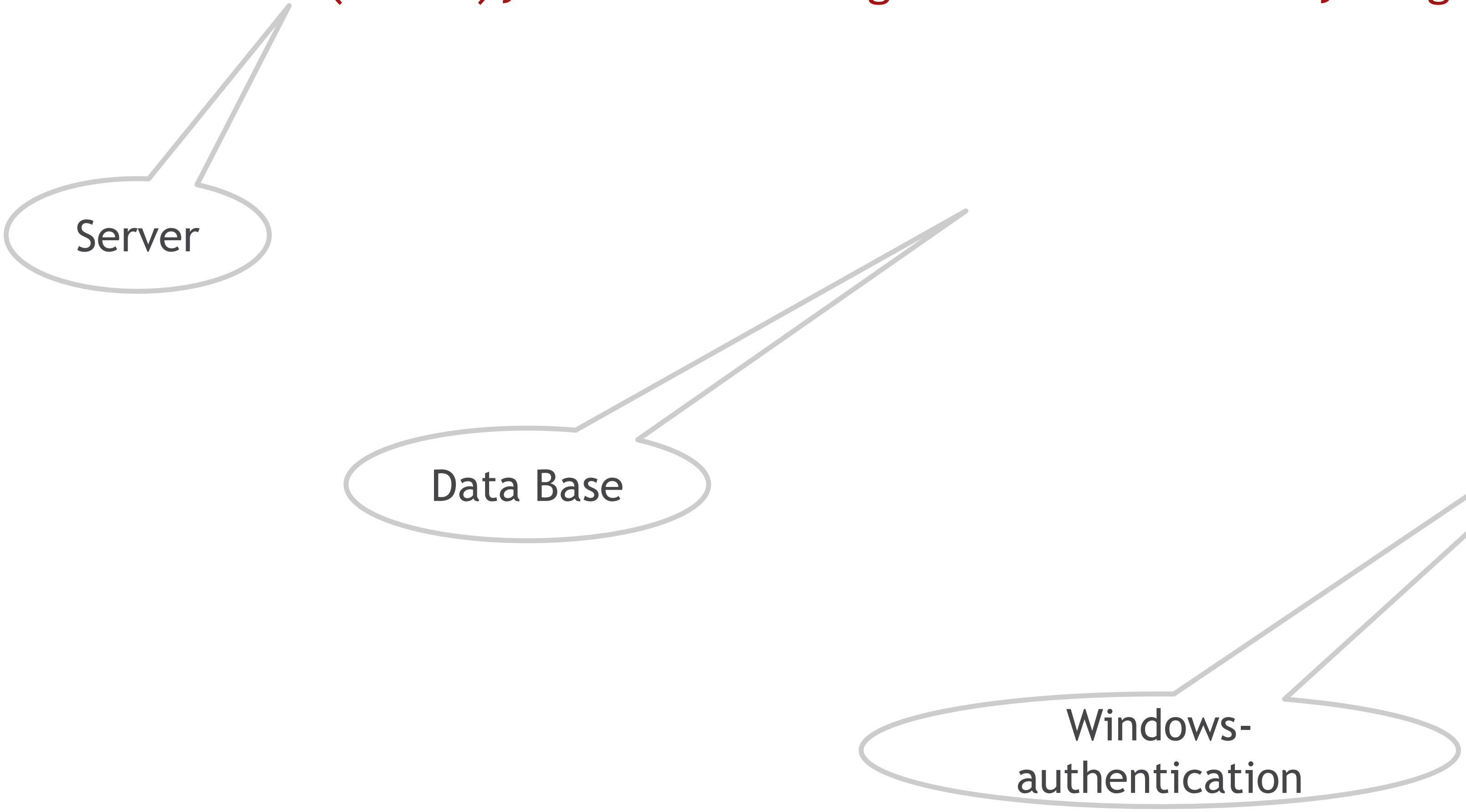
Create connection

```
var conn = new SqlConnection(
    "Data Source=(local);Initial Catalog=AdventureWorksLT;Integrated Security=True");

conn.Open();

// ...

conn.Close();
```

```
using (var conn = new SqlConnection(
    "Data Source=(local);Initial Catalog=AdventureWorksLT;Integrated Security=True"))
{
    conn.Open();

    // ...
}
```

Connection strings

```
Data Source=(local);Initial Catalog=AdventureWorksLT;Integrated Security=True
```



Connection strings

- Common structure
  param1=value; param2=value; …

- Every provider have own features

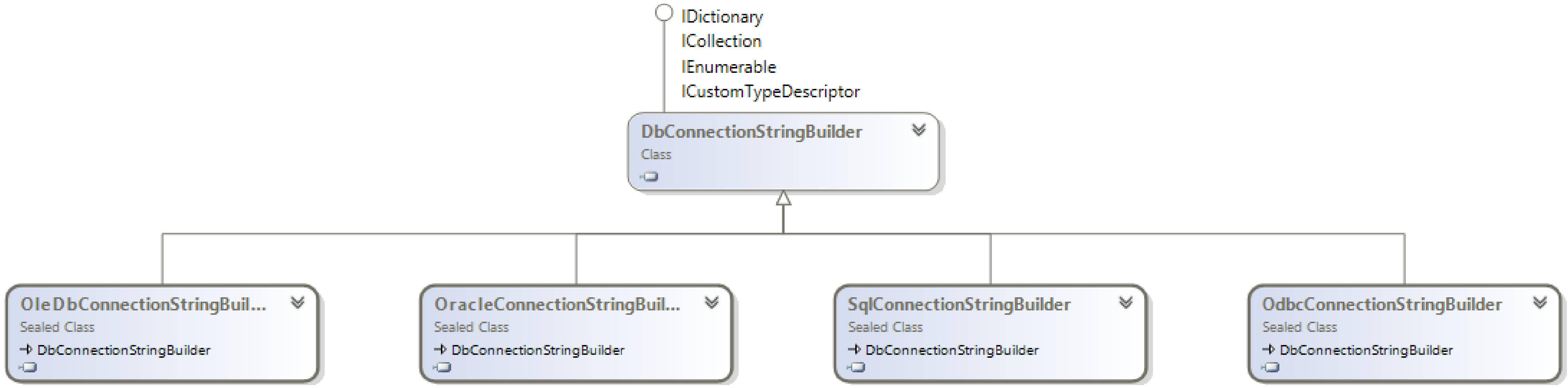
SQL Client	"Persist Security Info=False;Integrated Security=true;Initial Catalog=Northwind;server=(local)"
OleDb (MS Access)	"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\bin\LocalAccess40.mdb"
ODBC (Excel)	"Driver={Microsoft Excel Driver (*.xls)};DBQ=c:\bin\book1.xls"

Connection Strings (ADO.NET)

http://www.connectionstrings.com

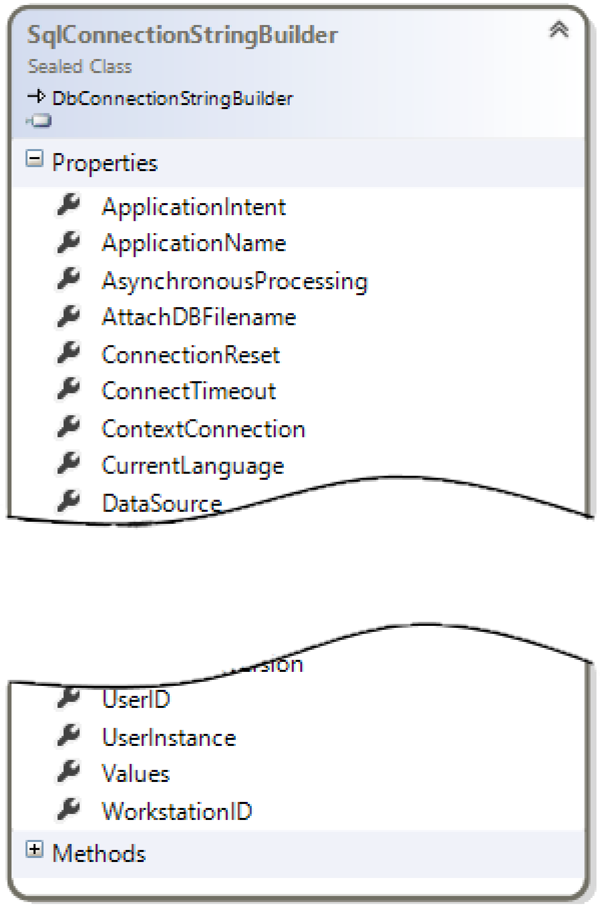
Connection string builder

Simplification of creation and decrease error number


IDictionary
ICollection
IEnumerable
ICustomTypeDescriptor
DbConnectionStringBuilder
Class
OleDbConnectionStringBuil...
Sealed Class
DbConnectionStringBuilder
OracleConnectionStringBuil...
Sealed Class
DbConnectionStringBuilder
SqlConnectionStringBuilder
Sealed Class
DbConnectionStringBuilder
OdbcConnectionStringBuilder
Sealed Class
DbConnectionStringBuilder

Connection string builder sample

```
var connectionStringBuilder = new SqlConnectionStringBuilder
{
    DataSource = "(local)",
    InitialCatalog = "Northwind",
    IntegratedSecurity = true
};

using (var connection =
    new SqlConnection(connectionStringBuilder.ConnectionString))
{
    connection.Open();
}
```


SqlConnectionStringBuilder
Sealed Class
→ DbConnectionStringBuilder

⊟ Properties
- ApplicationIntent
- ApplicationName
- AsynchronousProcessing
- AttachDBFilename
- ConnectionReset
- ConnectTimeout
- ContextConnection
- CurrentLanguage
- DataSource
- UserID
- UserInstance
- Values
- WorkstationID

⊞ Methods

Common connection parameters (SqlClient)

Parameter	Samples
Data Source / Server	(local) np:(local), tcp:(local), lpc:(local) W406811-DB11\PrimaryInstance
Initial Catalog / Database	Northwind
Integrated Security / Trusted_Connection	True
User ID / UID	mihail_romanov
Password	S12SFweqb3wl
AttachDBFilename / Initial File Name	|DataDirectory|\data\YourDB.mdf
Connect Timeout / Timeout	30

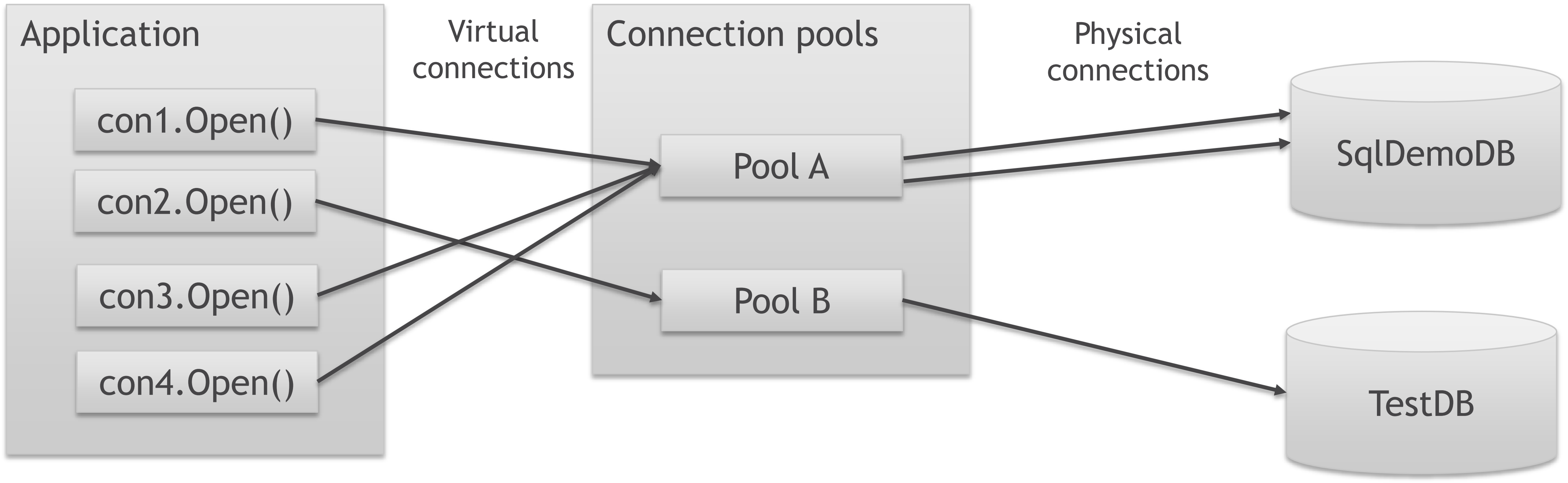
Connection parameters

Connection string + app.config + Provider Factories sample

```xml
<configuration>
  <connectionStrings>
    <add name="NorthwindConection"
         providerName="System.Data.SqlClient"
         connectionString="Data Source=(local);Initial Catalog=Northwind;Integrated Security=True"/>
  </connectionStrings>
</configuration>
```

```csharp
var connectionStringItem = ConfigurationManager.ConnectionStrings["NorthwindConection"];
var connectionString = connectionStringItem.ConnectionString;
var providerName = connectionStringItem.ProviderName;

var factory = DbProviderFactories.GetFactory(providerName);

using (var connection =factory.CreateConnection())
{
    connection.ConnectionString = connectionString;
    connection.Open();
}
```

Connection pools



~~Data Source=(local);Initial Catalog=SqlDemoDB;Integrated Security=True~~

Data Source=(local);Initial Catalog=**TestDB**;Integrated Security=True

Data Source=(local);Initial Catalog=**SqlDemoDB**;Integrated Security=True

Data Source=(local);Initial Catalog=**SqlDemoDB**;Integrated Security=True

Connection Pooling

Connection issues and best practices

- Keep connections to the data source in use for a minimal amount of time
- Always explicitly close your Connection or DataReader objects when you are finished using them

Best Practice

CONNECTED MODEL. COMMAND

Create a command

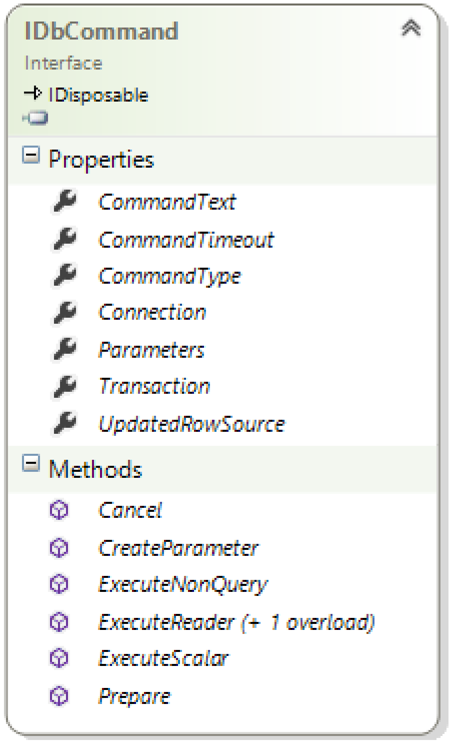
Command should be associated with Connection

```
using (IDbConnection connection = new SqlConnection(ConnectionString))
{
    connection.Open();

    var command = connection.CreateCommand();
}
```

```
using (IDbConnection connection = new SqlConnection(ConnectionString))
{
    connection.Open();

    IDbCommand command = new SqlCommand();
    command.Connection = connection;
}
```

Common Command properties

```
using (IDbConnection connection = new SqlConnection(ConnectionString))
{
    connection.Open();

    var command = connection.CreateCommand();

    command.CommandText = "select count(*) from Northwind.Customers";
    command.CommandType = CommandType.Text;

    var customersCount = command.ExecuteScalar();
    Console.WriteLine(customersCount);
}
```



IDbCommand
Interface
IDisposable

Properties
- CommandText
- CommandTimeout
- CommandType
- Connection
- Parameters
- Transaction
- UpdatedRowSource

Methods
- Cancel
- CreateParameter
- ExecuteNonQuery
- ExecuteReader (+ 1 overload)
- ExecuteScalar
- Prepare

Command types

Command Type	Samples / Comments

Text (default)	`command1.CommandText = "SELECT * FROM Northwind.Products";`
	`command1.CommandType = CommandType.Text;`
	
	`command2.CommandText = "exec sp_helpdb";`
	`command2.CommandType = CommandType.Text;`

StoredProcedure	`command3.CommandText = "sp_helpdb";`
	`command3.CommandType = CommandType.StoredProcedure;`

TableDirect	**Support only in .NET Framework Data Provider for OLE DB**
	
	`command.CommandText = "Northwind.Customers";`
	`command.CommandType = CommandType.TableDirect;`

Command results

Result Type	Samples

Row set	`command.CommandText =`
	`    "SELECT CompanyName FROM Northwind.Customers";`
	
	`SqlDataReader reader = command.ExecuteReader();`

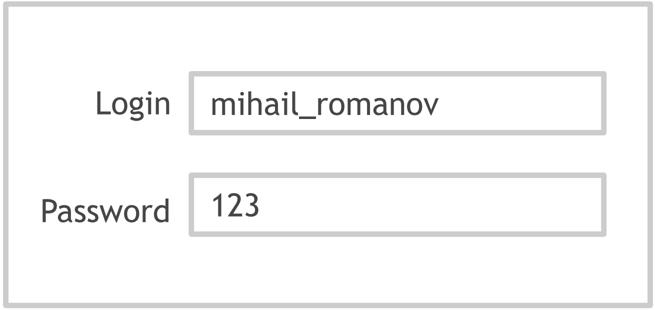
Single value	`command.CommandText =`
	`    "SELECT count(*) FROM Northwind.Customers";`
	
	`int count = (int)command.ExecuteScalar();`

No result	`command.CommandText =`
	`    "UPDATE Northwind.Products SET UnitPrice = UnitPrice - 0.0002";`
	
	`int affected = command.ExecuteNonQuery();`

Xml	`command.CommandText =`
	`    "SELECT * FROM Northwind.Customers FOR XML AUTO, ROOT('Customers')";`
	
	`XmlReader xmlReader = command.ExecuteXmlReader();`

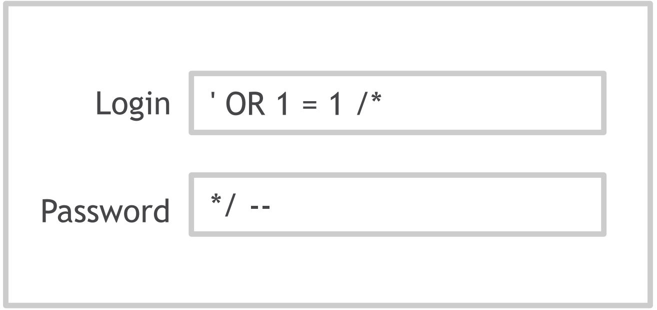
Parametrized query. SQL injections

```
string.Format(
    "select top 1 * from dbo.Users where Login = '{0}' and Password = '{1}'", login, password);
```



Login	mihail_romanov
Password	123

```
select top 1 * from dbo.Users
where Login = 'mihail_romanov' and Password = '123'
```



Login	' OR 1 = 1 /*
Password	*/ --

```
select top 1 * from dbo.Users
where Login = '' OR 1 = 1 /*' and Password = '123'*/ --
```

Command parameters

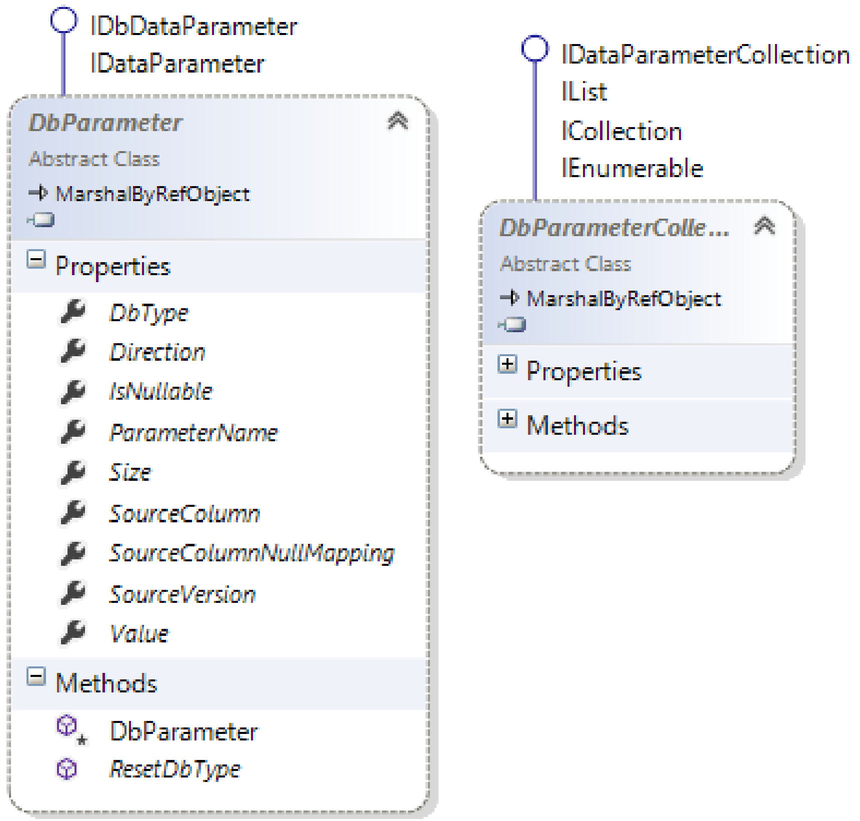
```
command.CommandText =
  "SELECT count(*) FROM Northwind.Products
   WHERE UnitPrice >= @minPrice";
```

IDbCommand

```
var minPrice = command.CreateParameter();
minPrice.ParameterName = "@minPrice";
minPrice.DbType = DbType.Decimal;
minPrice.Value = 50;

command.Parameters.Add(minPrice);
```

SqlCommand

```
command.Parameters.AddWithValue("@minPrice", 50m);
```



IDbDataParameter
IDataParameter

DbParameter
Abstract Class
MarshalByRefObject

Properties
- *DbType*
- *Direction*
- *IsNullable*
- *ParameterName*
- *Size*
- *SourceColumn*
- *SourceColumnNullMapping*
- *SourceVersion*
- *Value*

Methods
- DbParameter
- *ResetDbType*

IDataParameterCollection
IList
ICollection
IEnumerable

DbParameterColle...
Abstract Class
MarshalByRefObject

Properties

Methods

Call stored procedures

```
CREATE PROCEDURE [Northwind].[CustOrdersStatistic]
    @CustomerID nchar(5),
    @Shipped int OUTPUT,
    @All int OUTPUT
AS
...
```

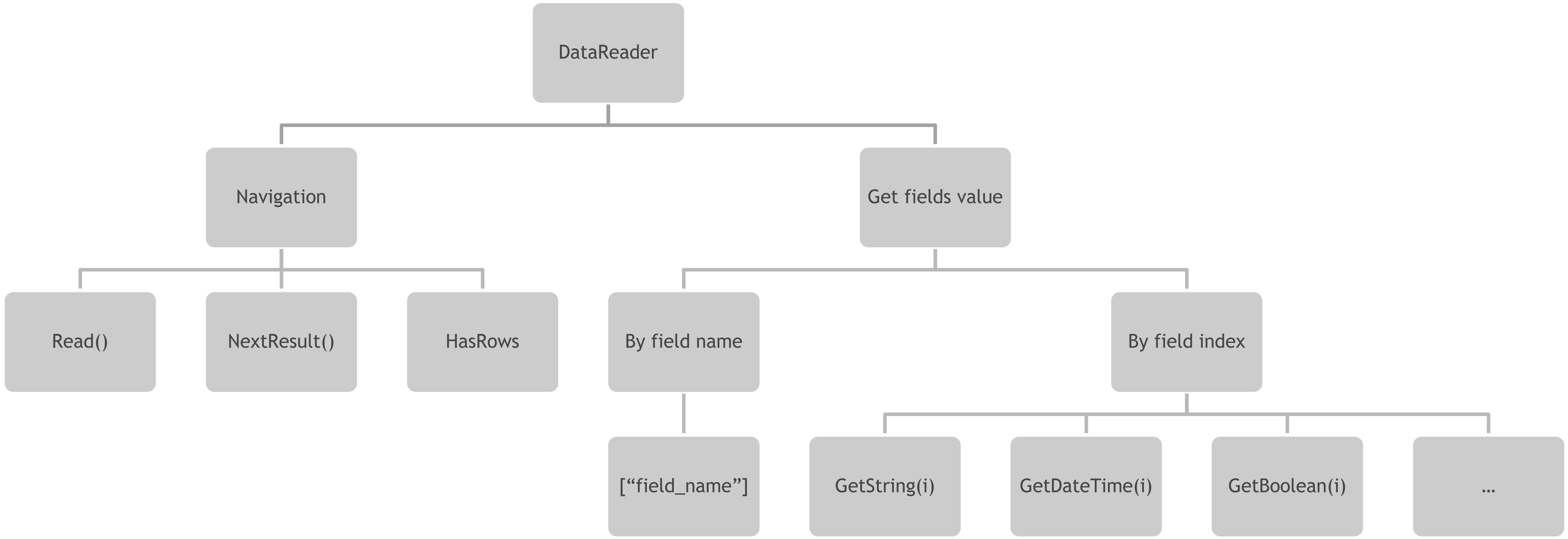
```
var command = connection.CreateCommand();
command.CommandText = "[Northwind].[CustOrdersStatistic]";
command.CommandType = CommandType.StoredProcedure;

command.Parameters.AddWithValue("@CustomerID", "BONAP");

var all = command.Parameters.Add(
    new SqlParameter()
    {
        ParameterName = "@All",
        DbType = DbType.Int32,
        Direction = ParameterDirection.Output
    });

var shipped = command.Parameters.Add(
    new SqlParameter()
    {
        ParameterName = "@Shipped",
        DbType = DbType.Int32,
        Direction = ParameterDirection.Output
    });

command.ExecuteNonQuery();

Console.WriteLine("{0} {1}", all.Value, shipped.Value);
```

CONNECTED MODEL. DATAREADER

Read result

```
using (IDbConnection connection = new SqlConnection(ConnectionString))
{
    var command = connection.CreateCommand();
    command.CommandText = "SELECT CompanyName, City, Region FROM Northwind.Customers";

    connection.Open();

    using (IDataReader reader = command.ExecuteReader())
    {

        while (reader.Read())
        {
            Console.WriteLine("{0} - {1}, {2}",
                reader["CompanyName"],
                reader["City"],
                reader["Region"]);
        }

    }
}
```

DataReader methods



CONNECTED MODEL. TRANSACTIONS

Transaction Types

- Local Transaction
  - On Client (usage IDbTransaction / DbTransaction)
  - On Server (explicit usage BEGIN TRANSACTION, COMMIT TRANSACTION, and ROLLBACK TRANSACTION statements – e.g. in Stored Procedure)
- Distributed Transaction
  - Only for SQL Client (usage System.Transactions)

Local transaction

```
using (IDbConnection connection = new SqlConnection(ConnectionString))
{
    connection.Open();

    using (var transaction = connection.BeginTransaction())
    {
        var command = connection.CreateCommand();
        command.CommandText = "delete from Northwind.[Order Details] where OrderID = @orderId;";
        command.CommandText += "delete from Northwind.Orders where OrderID = @orderId;";

        var orderIdParam = command.CreateParameter();
        orderIdParam.ParameterName = "@orderId";
        orderIdParam.Value = orderId;
        command.Parameters.Add(orderIdParam);

        command.Transaction = transaction;

        command.ExecuteNonQuery();

        transaction.Commit();
    }
}
```

Open connection

Start transaction

Specify transaction for command

Commit or Rollback transaction