

DB Apps Introduction

Intro to ADO.NET



SoftUni Team
Technical Trainers



**Software
University**



**SoftUni
Foundation**



Software University

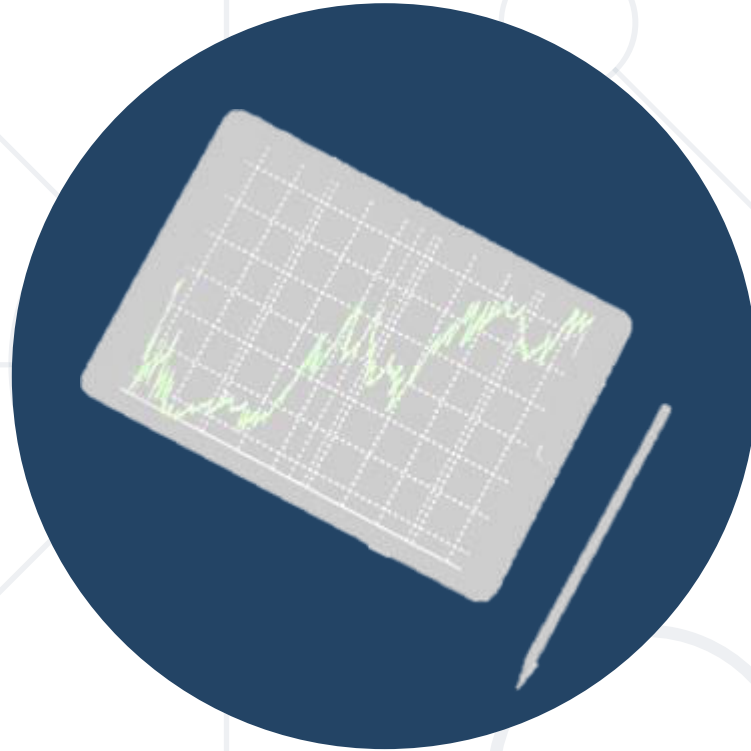
<http://softuni.bg>

1. Data Access Models
2. ADO.NET Architecture
3. Accessing SQL Server from **ADO.NET**
4. SQL Injection



sli.do

#csharp-db



Data Access Models

Connecting to a DB through C#

Connected Model

- **Connected data access model**
 - Applicable to an environment where the database is constantly available



ADO.NET client

constantly open
connection



Database

- **Connected data access model (SqlClient)**
 - **Benefits:**
 - Concurrency control is easier to maintain
 - Better chance to work with the most recent version of the data
 - **Drawbacks:**
 - Needs a constant reliable network
 - Problems when scalability is an issue



ADO.NET Architecture

What Is ADO.NET?

- ADO.NET is a standard **.NET class library** for accessing databases, processing data and XML
- Supports connected, disconnected and ORM data access models
 - Excellent integration with **LINQ**
 - Allows executing SQL in **RDBMS** systems
 - Allows accessing data in the **ORM** approach

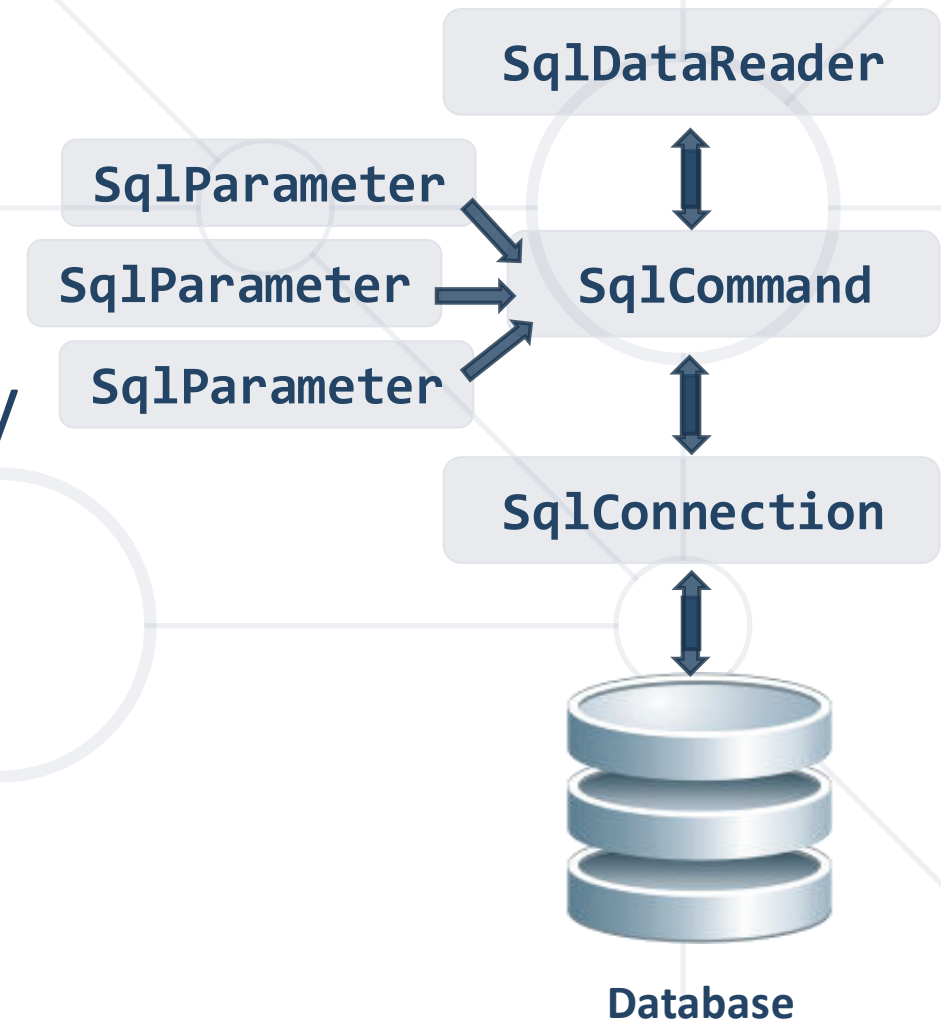
- Data Providers are collections of classes that provide access to various databases
 - For different RDBMS systems different **Data Providers** are available
- Several common objects are defined
 - **Connection** – to connect to the database
 - **Command** – to run an SQL command
 - **DataReader** – to retrieve data

Data Providers in ADO.NET (2)

- Several standard ADO.NET Data Providers come as part of .NET Framework
 - **SqlClient** – accessing **SQL Server**
 - **OleDb** – accessing standard **OLE DB** data sources
 - **Odbc** – accessing standard **ODBC** data sources
 - **Oracle** – accessing **Oracle** databases
- Third party Data Providers are available for:
 - **MySQL, PostgreSQL, Interbase, DB2, SQLite**
 - Other RDBMS systems and data sources
 - SQL Azure, Salesforce CRM, Amazon SimpleDB, ...

SqlClient and ADO.NET Connected Model

- Retrieving data in connected model
 - Open a connection (**SqlConnection**)
 - Execute command (**SqlCommand**)
 - Process the result set of the query by using a reader (**SqlDataReader**)
 - Close the reader
 - Close the connection



- **ORM data access model** (Entity Framework Core)
 - Maps **database tables** to **classes** and **objects**
 - Objects can be **automatically persisted** in the database
 - Can operate in both connected and disconnected modes



- **ORM** model benefits
 - Less code
 - Use objects with **associations** instead of tables and SQL
 - Integrated object query mechanism
- ORM model drawbacks:
 - Less flexibility
 - SQL is automatically generated
 - Performance issues (sometimes)

- **Entity Framework Core** is a generic **ORM** framework
 - Create entity data model mapping the database
 - Open an object context
 - Retrieve data with LINQ / modify the tables in the object context
 - Persist the object context changes into the DB
 - Connection is automatically managed



SQL Client Data Provider

- **SqlConnection**
 - Establish database connection to SQL Server
- **SqlCommand**
 - Executes SQL commands on the SQL Server through an established connection
 - Could accept parameters (**SqlParameter**)
- **SqlDataReader**
 - Retrieves data (record set) from SQL Server as a result of SQL query execution

The SqlConnection Class

- **SqlConnection** establishes a connection to SQL Server database
 - Requires a valid connection string
- Connection string example:

```
Data Source=(local)\SQLEXPRESS;Initial  
Catalog=SoftUni;Integrated Security=true;
```

- Connecting to SQL Server:

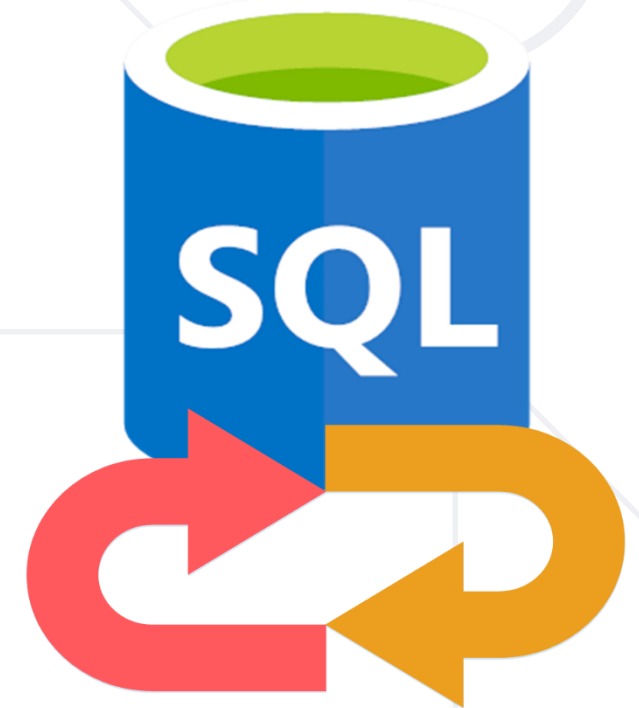
```
SqlConnection con = new SqlConnection(  
    @"Server=.;  
    Database=SoftUni;  
    Integrated Security=true");  
con.Open();
```

- Creating and opening connection to SQL Server (database **SoftUni**)

```
SqlConnection dbCon = new SqlConnection(  
    "Server=.\SQLEXPRESS; " +  
    "Database=SoftUni; " +  
    "Integrated Security=true");  
dbCon.Open();  
using (dbCon)  
{  
    //TODO: Use the connection to execute SQL commands here ...  
}
```

- **Database connection string**
 - Defines the parameters needed to establish the connection to the database
- Settings for **SQL Server connections**:
 - **Data Source / Server** – server name / IP address + database instance name
 - **Database / Initial Catalog** – database name
 - **User ID / Password** – credentials
 - **Integrated Security** – false if credentials are provided

- Explicitly opening and closing a connection
 - **Open()** and **Close()** methods
 - Works through the connection pool
- DB connections are **IDisposable** objects
 - Always use the **using** construct in C#!



- More important methods
 - **ExecuteScalar()**
 - Returns a single value - the value in the first column of the first row of the result set (as **System.Object**)
 - **ExecuteReader()**
 - Returns a **SqlDataReader**
 - It is a cursor over the returned records (result set)
 - **CommandBehavior** – assigns some options
 - **ExecuteNonQuery()**
 - Used for non-query SQL commands, e.g. **INSERT**
 - Returns the number of affected rows (**int**)

SqlCommand – Example

```
SqlConnection dbCon = new SqlConnection(  
    "Server=.; " +  
    "Database=SoftUni; " +  
    "Integrated Security=true");  
dbCon.Open();  
using(dbCon)  
{  
    SqlCommand command = new SqlCommand(  
        "SELECT COUNT(*) FROM Employees", dbCon);  
    int employeesCount = (int) command.ExecuteScalar();  
    Console.WriteLine("Employees count: {0} ", employeesCount);  
}
```

- **SqlDataReader** retrieves a sequence of records (cursor) returned as result of an SQL command
 - Data is available for reading only (can't be changed)
 - Forward-only row processing (no move back)
- Important properties and methods:
 - **Read()** – moves the cursor forward and returns false if there is no next record
 - **Indexer[]** – retrieves the value in the current record by given column name or index
 - **Close()** – closes the cursor and releases resources

SqlDataReader – Example

```
SqlConnection dbCon = new SqlConnection(...);
dbCon.Open();
using(dbCon)
{
    SqlCommand command = new SqlCommand("SELECT * FROM Employees", dbCon);
    SqlDataReader reader = command.ExecuteReader();
    using (reader)
    {
        while (reader.Read())
        {
            string firstName = (string)reader["FirstName"];
            string lastName = (string)reader["LastName"];
            decimal salary = (decimal)reader["Salary"];
            Console.WriteLine("{0} {1} - {2}", firstName, lastName, salary);
        }
    }
}
```

Fetch more rows
until finished



SQL Injection

What is SQL Injection? How to Prevent It?

What is SQL Injection? (1)

```
bool IsPasswordValid(string username, string password)
{
    string sql =
        $"SELECT COUNT(*) FROM Users " +
        $"WHERE UserName = '{username}' AND" +
        $"PasswordHash = '{CalcSHA1(password)}'";
    SqlCommand cmd = new SqlCommand(sql, dbConnection);

    int matchedUsersCount = (int)cmd.ExecuteScalar();
    return matchedUsersCount > 0;
}
```

What is SQL Injection? (2)

```
bool normalLogin =  
    IsPasswordValid("peter", "qwerty123"); // true  
  
bool sqlInjectedLogin =  
    IsPasswordValid("' or 1=1 --", "qwerty123"); // true  
  
bool evilHackerCreatesNewUser =  
    IsPasswordValid("' INSERT INTO Users VALUES('hacker','') --",  
    "qwerty123");
```

How Does SQL Injection Work?

- The following SQL commands are executed:

- Usual password check (no SQL injection):

```
SELECT COUNT(*) FROM Users WHERE UserName = 'peter'  
AND PasswordHash = 'X0wXWxZePV5iyeE86Ejvb+rIG/8='
```

- SQL-injected password check:

```
SELECT COUNT(*) FROM Users WHERE UserName = ' ' or 1=1  
-- ' AND PasswordHash = 'X0wXWxZePV5iyeE86Ejvb+rIG/8='
```

- SQL-injected INSERT command:

```
SELECT COUNT(*) FROM Users WHERE UserName = ''  
INSERT INTO Users VALUES('hacker','')  
--' AND PasswordHash = 'X0wXWxZePV5iyeE86Ejvb+rIG/8='
```

- Ways to prevent the SQL injection:
 - SQL-escape all data coming from the user:

```
string escapedUsername = username.Replace("'", "''");
string sql =
    "SELECT COUNT(*) FROM Users " +
    "WHERE UserName = '" + escapedUsername + "' and " +
    "PasswordHash = '" + CalcSHA1(password) + "'";
```

 - Not recommended: use as last resort only!
 - Preferred approach:
 - Use **parameterized queries**
 - Separate the SQL command from its arguments

- What are **SqlParameter**s?
 - SQL queries and stored procedures can have input and output parameters
 - Accessed through the **Parameters** property of the **SqlCommand** class
- Properties of **SqlParameter**:
 - **ParameterName** – name of the parameter
 - **DbType** – SQL type (**NVarChar**, **Timestamp**, ...)
 - **Size** – size of the type (if applicable)
 - **Direction** – input / output

Parameterized Commands – Example

```
void InsertProject(string name, string description, DateTime startDate)
{
    SqlCommand cmd = new SqlCommand(
        "INSERT INTO Projects " +
        "(Name, Description, StartDate, EndDate) VALUES " +
        "(@name, @desc, @start, @end)", dbCon);

    cmd.Parameters.AddWithValue("@name", name);
    cmd.Parameters.AddWithValue("@desc", description);
    cmd.Parameters.AddWithValue("@start", startDate);

    cmd.ExecuteNonQuery();
}
```



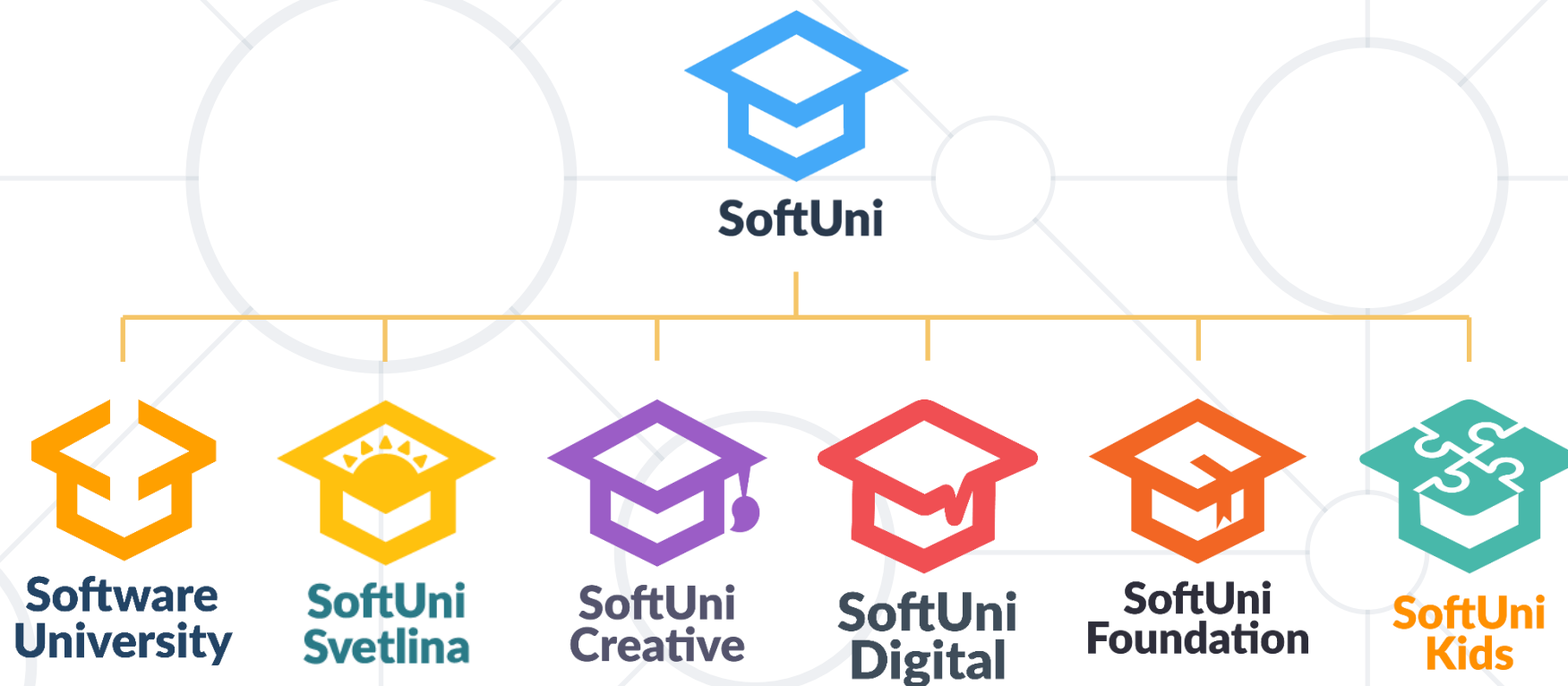
Connecting to Non-Microsoft Databases

- ADO.NET supports accessing various databases via their Data Providers:
 - **OLE DB** – supported internally in ADO.NET
 - Access any OLE DB-compliant data source
 - E.g. MS Access, MS Excel, MS Project, MS Exchange, Windows Active Directory, text files
 - **Oracle** – supported internally in ADO.NET
 - **MySQL** – third party extension

- ADO.NET provides an interface between our apps and the database engine
- Different engines can be used with other data providers
- SQL commands must be parametrized to prevent malicious behavior



Questions?



SoftUni Diamond Partners



XSsoftware



SBTech
we know sports



telenor



SoftwareGroup
doing it right

NETPEAK



SmartIT



Postbank

Решения за твоето утре



INDEAVR

Serving the high achievers



INFRAGISTICS®



STEMO®
Computer Systems & Software

SUPERHOSTING.BG

SoftUni Organizational Partners



OneBit
SOFTWARE



WORLD
OF
MYTHS

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
 - softuni.bg
- Software University Foundation
 - <http://softuni.foundation/>
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

