

Data Access with ADO.NET



Filip Ekberg

Principal Consultant & CEO

@fekberg | fekberg.com

When to Use ADO.NET

High performance situations

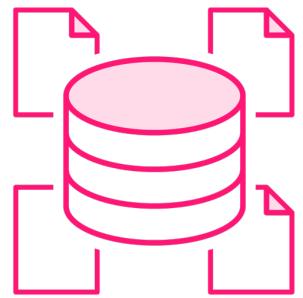
You need total control over the SQL



Entity Framework builds on **ADO.NET**

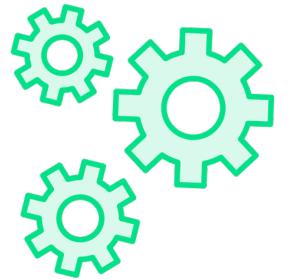


Using ADO.NET



Access data

Use with different data sources



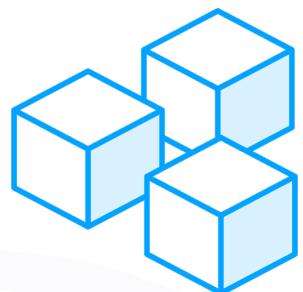
Support for many providers

SQL Server, SQLite, MySQL, XML, and many more



Powerful & Flexible

Gives you total control



Consistent API for accessing data

This abstraction gives you a similar way of accessing data in different data stores/sources



**Requires that you know the
SQL syntax for each
database you are working
with**



Overview



Learn about different components of ADO.NET

How to connect to a database

How to execute a SQL query

Handle potential results

Avoiding SQL injections and security considerations



Example ADO.NET Providers

```
# SQLite  
dotnet add package Microsoft.Data.Sqlite
```



```
# SQL Server  
dotnet add package Microsoft.Data.SqlClient
```



Creating a Connection

```
# SQLite  
using SqliteConnection    connection = new(connectionString)  
  
# SQL Server  
using SqlConnection      connection = new(connectionString)
```



DO NOT cache the instance!



Using a Factory Method

```
SqlConnection CreateConnection()
{
    // Load connection string from a secure place
    return new SqlConnection(connectionString);
}

using connection = CreateConnection();
```



Creating a Command



Creating a Command

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



Creating a Command

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



Returns the correct type of command.

Example: SqlCommand or SqliteCommand



Creating a Command

```
var command = connection.CreateCommand();  
command.CommandText = sqlQuery;
```



Creating a Command

```
var command = connection.CreateCommand();  
command.CommandText = sqlQuery;
```

```
var command = new SqlCommand(sqlQuery, connection);
```



Creating a Command

```
var command = connection.CreateCommand();  
command.CommandText = sqlQuery;
```

```
var command = new SqlCommand(sqlQuery, connection);
```



Creating a Command

```
var command = connection.CreateCommand();  
command.CommandText = sqlQuery;
```

```
var command = new SqlCommand(sqlQuery, connection);
```

```
var command = new SqliteCommand(sqlQuery, connection);
```



Creating a Command

```
var command = connection.CreateCommand();  
command.CommandText = sqlQuery;
```

```
var command = new SqlCommand(sqlQuery, connection);
```

```
var command = new SqliteCommand(sqlQuery, connection);
```

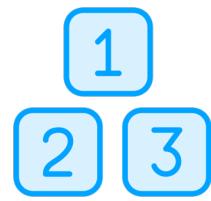


Retrieve All Orders and Their Customer Data

```
SELECT * FROM [Orders]
INNER JOIN [Customers] ON [Customers].Id =
[Orders].CustomerId
```

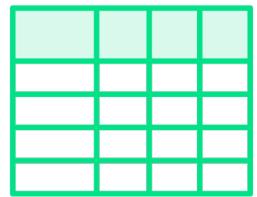


Execute the Command



Execute Non Query

Returns the number of affected rows



Execute Reader

Lets you consume the result through a data reader



Execute Scalar

Returns the first column from the first row in the result



Execute XmlReader

Requires an XML result



Forgot to dispose or close?

May occupy a connection
to the database!



Up Next:

Inserting Data



Command with Parameters

Requires that you know SQL

You have to know the column types



Inserting Data Using a Parameterized Query



Inserting Data Using a Parameterized Query

```
INSERT INTO [Customers]
```



Inserting Data Using a Parameterized Query

```
INSERT INTO [Customers]
(Id, Name, Address, PostalCode, Country, PhoneNumber)
```



Inserting Data Using a Parameterized Query

```
INSERT INTO [Customers]
(Id, Name, Address, PostalCode, Country, PhoneNumber)
```

```
VALUES
```

```
(NEWID(), @Name, @Address, @PostalCode, @Country, @PhoneNumber)
```



Potential SQL Injection

```
string byeByeTable = "'); DROP TABLE [Customers] --";
```

```
string query = "INSERT INTO [Customers] (Name) VALUES ('" + byeByeTable + "')";
```



Potential SQL Injection

```
string byeByeTable = "'); DROP TABLE [Customers] --";
```

```
string query = "INSERT INTO [Customers] (Name) VALUES ('" + byeByeTable + "')";
```

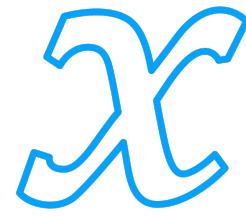
WARNING



DO NOT DO THIS



Parameterized SQL to the Rescue!



Entity Framework and other ORMs rely on parameters



Values are treated as literals instead of a segment in the SQL query



The values in a parameter is treated as the type specified



Update and Delete Data

```
UPDATE [Customers] SET Name = @Name WHERE Id = @Id
```

```
DELETE FROM [Customers] WHERE Id = @Id
```



Creating and Opening a Connection

```
# SQLite
using SqliteConnection    connection = new(connectionString)

# SQL Server
using SqlConnection    connection = new(connectionString)

connection.Open();
```



A consistent API no matter what data source



Use parameters to map values into your queries!



Use a Parameter

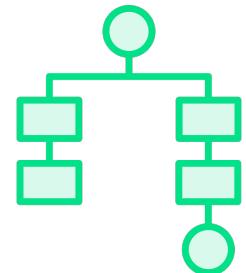
```
string name = "Filip Ekberg";  
  
string query = "INSERT INTO [Customers] (Name) VALUES (@NameParameter)";  
  
var nameParameter =  
    new SqlParameter("NameParameter", System.Data.SqlDbType.NVarChar);  
  
nameParameter.Value = name;  
  
command.Parameters.Add(nameParameter);
```



**There's much more to
discover on your own!**



Data Access in C#



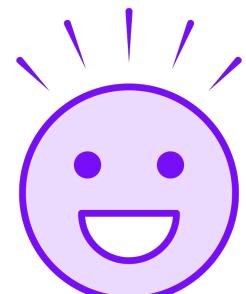
You will most of the time rely on an ORM

Entity Framework Core will be what you experience for the most part



ADO.NET is great for high performance applications

You have full control and can tweak each query



You can now choose what is best suited in your application!

