



# Module 2 Day 7

Data Access Objects

# What makes an application?

- Program Data

- ✓ Variables & .NET Data Types
- ✓ Arrays
- ✓ More Collections (list, dictionary, stack, queue)
- ✓ Classes and objects (OOP)

- Program Logic

- ✓ Statements and expressions
- ✓ Conditional logic (if)
- ✓ Repeating logic (for, foreach, do, while)
- ✓ Methods (functions / procedures)
- ✓ Classes and objects (OOP)
- ❑ Frameworks (MVC)

- Input / Output

- User

- ✓ Console read / write
- ❑ HTML / CSS
- ❑ Front-end frameworks (HTML / CSS / JavaScript)

- Storage

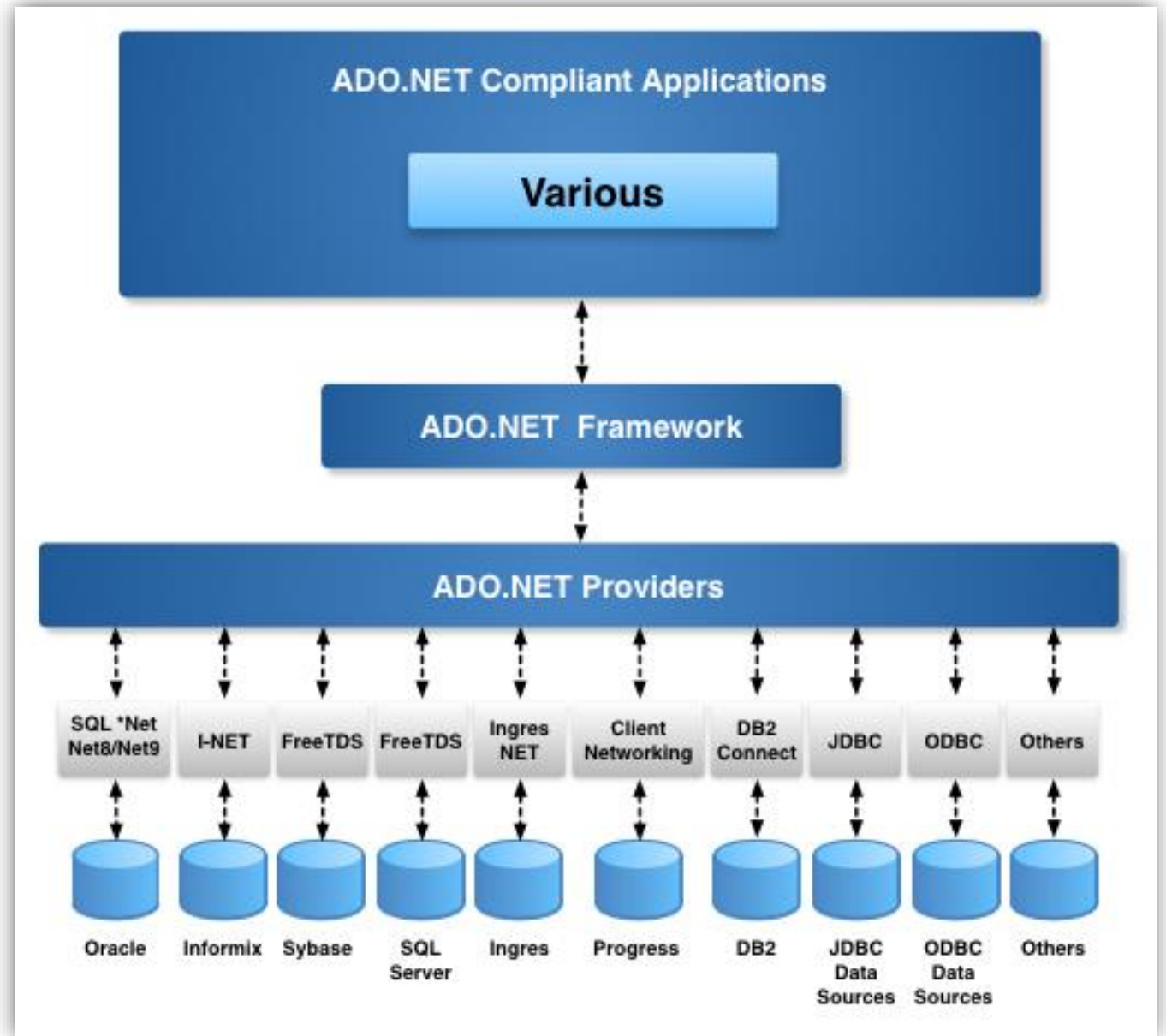
- ✓ File I/O
- ❖ Relational database
- ❑ APIs

# Database Connectivity

- SQL Server and other databases use a client-server architecture
- The DBMS (SQL Server) is the *server*
- Last week, SSMS was the *client*
- This week, C# programs will be the *client*

# ADO.Net

- *Consumers* – applications that need access to data
- *Providers* – components that provide data
- *ADO.Net Framework* – defines how consumers talk to providers to get data
- Allows access to lots of databases using a common model



# ADO.Net - Interfaces

- [IDbConnection](#)
  - Represents an open connection to a data source, and is implemented by .NET Framework data providers that access relational databases.
- [SqlCommand](#)
  - Represents an SQL statement that is executed while connected to a data source, and is implemented by .NET Framework data providers that access relational databases.
- [IDataReader](#)
  - Provides a means of reading one or more forward-only streams of result sets obtained by executing a command at a data source, and is implemented by .NET Framework data providers that access relational databases.
- SQL Server provider implements these in SqlConnection, SqlCommand, SqlDataReader classes

# Data Access

- using
- Connection string
- SqlCommand constructor
- ExecuteReader()
- Read()
- Accessing column data (dictionary-like access)

Let's  
Code

```
// Things can go awry, so put it in a try
try
{
    using (SqlConnection conn = new SqlConnection("conn str..."))
    {
        conn.Open();                // Open the connection to the DB
        // A command is a query statement
        SqlCommand cmd = new SqlCommand("SELECT * FROM city", conn);
        // Execute the statement and get the results
        SqlDataReader reader = cmd.ExecuteReader();
        // Read row by row
        while (reader.Read())
        {
            // Do something with the row
            string name = Convert.ToString(reader["name"]);
            int population = Convert.ToInt32(reader["population"]);
            Console.WriteLine($"City: {name}, population {population}");
        }
    } // End of the "using". Closes the connection in Dispose()
}
catch (SqlException ex)
{
    // There was an exception...do something with it here
}
```

# SQL Server / C# Data Types

- <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql-server-data-type-mappings>

SQL Server	C#		SQL Server	C#
bit	bool		char/nchar	string
date	DateTime		datetime	DateTime
decimal	decimal		float	double
int	int		money	decimal
ntext/text	string		nvarchar/ varchar	string
tinyint	byte			



# Parameterized Queries

- Placeholders for each parameter in the query
- Parameters collection (on Command)

```
string countryCode = "USA";  
SqlCommand cmd = new SqlCommand(  
    "SELECT * FROM city WHERE countryCode = @countryCode;",  
    conn);  
cmd.Parameters.AddWithValue("@countryCode", countryCode);
```



Let's  
Code



# Other methods to Execute SQL

- *ExecuteNonQuery*: When no results will be returned
  - Example: `UPDATE` or `DELETE`
- *ExecuteScalar*: When exactly one column and one row will be returned
  - That is, a single value
  - Example: `INSERT` with `Select @@Identity`



Let's  
Code

# The DAO Pattern

- Data Access Objects
- Only role is to store and retrieve data
- Decouples the application from the persistence layer
  - Could be DB, file system, test objects, etc.
  - Isolates changes needed if the schema changes
- Performs object-to-relational mapping (ORM)
- Use of Interfaces provides additional flexibility



Let's  
Code