# ADO.NET



# Upphovsrätt för detta verk

Detta verk är framtaget i anslutning till kursen ASP.NET Web Forms vid Linnéuniversitetet.

**Du får använda detta verk så här:**

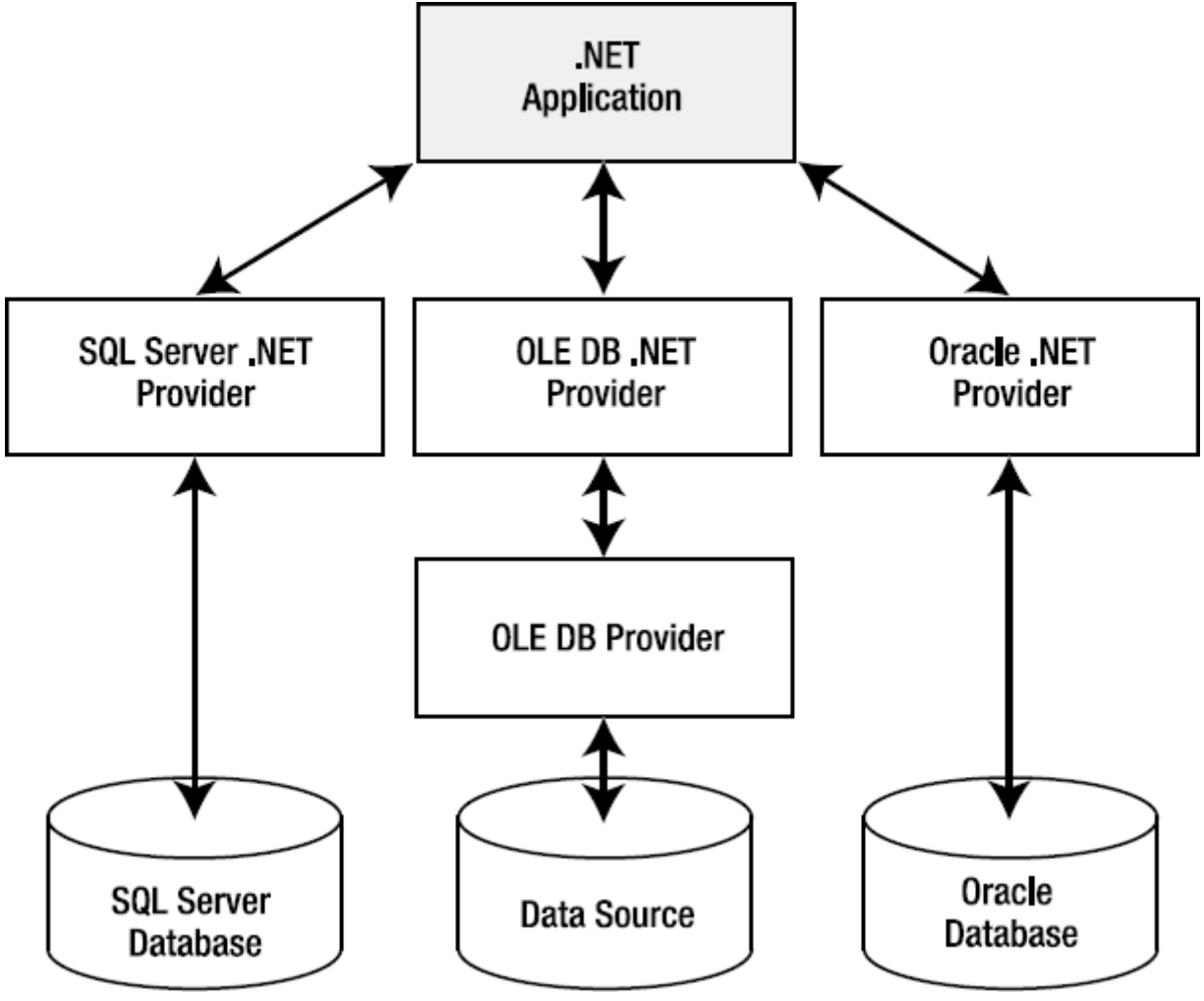
# Datadriven webbapplikation – en distribuerad arkitektur


- ✓ Huvuduppgiften är att hämta, visa och modifiera data.
- ✓ Applikationen finns "utspridd" på flera olika fysiska maskiner – fysiska lager.

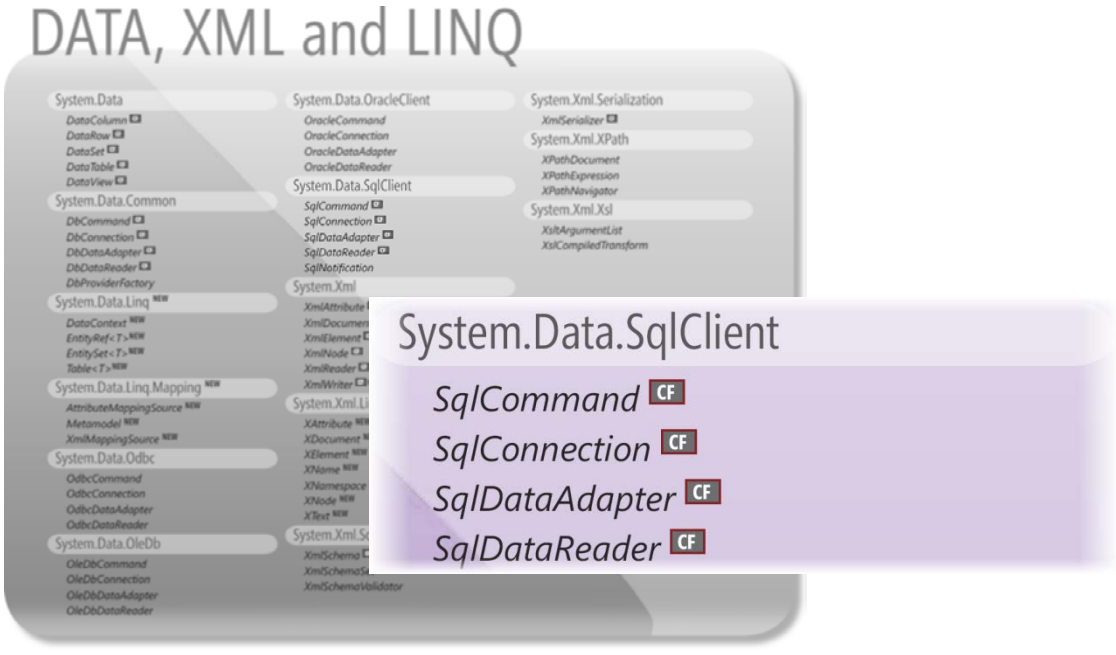
# ”ADO.NET Data Providers”

- En ”data provider” tillhandahåller ett antal tjänster som tillsammans utgör en länk mellan en databas och applikationen som använder databasen.
- En ”data provider” är helt enkelt en uppsättning klasser du använder för att ansluta till en databas, köra SQL-kommandon och ta emot data.

# Arkitekturen i ADO.NET

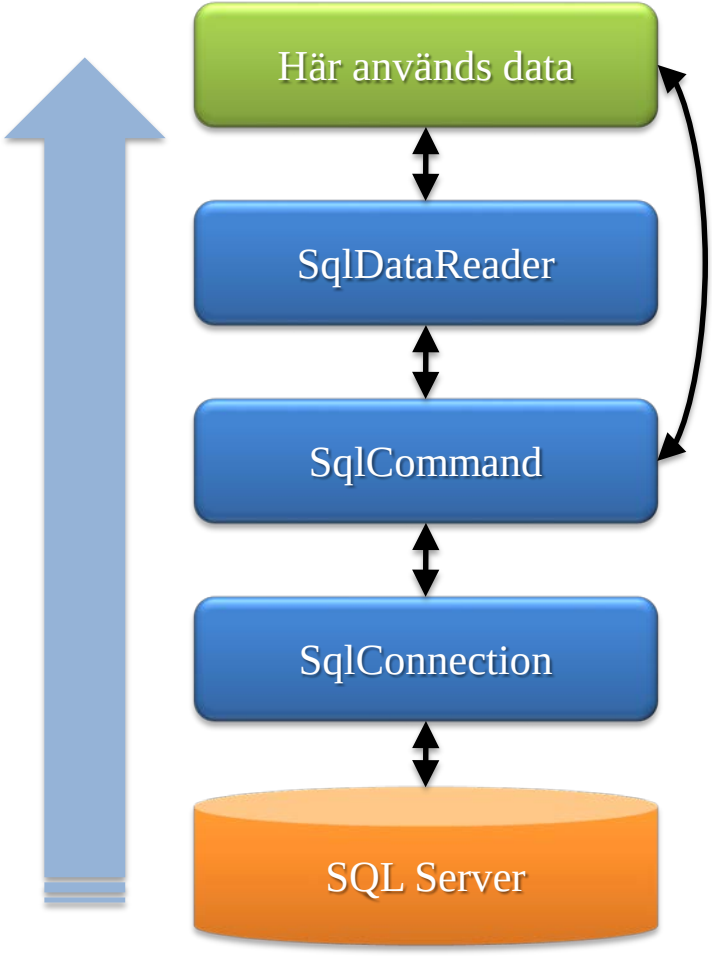


# Vilka klasser ska jag använda?




- Klasserna i namnutrymmet `System.Data.SqlClient` använder du då du vill ansluta till en databas av typen Microsoft SQL Server.
- Det finns andra klasser du kan använda om du har en annan databas.

# Connection, Command och DataReader




- Klassen `SqlConnection` använder du för att skapa en anslutning till databasen du vill kommunicera med.
- Du använder klassen `SqlCommand` då du vill exekvera ett SQL-uttryck.
- Hur tar jag hand om datat en SELECT-sats returnerar? Du kan använda `SqlDataReader` för att läsa datat som returneras post för post.

# Att ansluta till databas

- För att hämta information från en databas måste en anslutning öppnas till databasen. För att ansluta till en SQL Server används ett `SqlConnection`-objekt.
  1. Instansiera ett anslutningsobjekt från klassen `SqlConnection`. (Klassen implementerar interfacet `System.Data.IDbConnection`.)
  2. Konfigurera objektet genom att använda egenskapen `ConnectionString`.
  3. Öppna anslutningen genom att anropa metoden `Open`.
- eller
  1. Instansiera ett anslutningsobjekt från klassen `SqlConnection` och konfigurera objektet genom att skicka med anslutningssträngen i konstruktorn.
  2. Öppna anslutningen genom att anropa metoden `Open`.

# Anslutningsträng

- Då ett SqlConnection-objekt skapas måste du ange en anslutningssträng.
- Anslutningssträngen innehåller information som krävs för att kunna skapa en anslutning.
  - Var databasservern finns. (`Data Source`)
  - Vilken databas som ska användas. (`Initial Catalog`)
  - Hur autentiseringen ska gå till. (`Intergrated Security` / `Persist Security Info`)
- För att ansluta till en SQL Server på den lokala datorn med Windowsautentisering:

```
Data Source=localhost;Initial Catalog=AdventureWorks;Intergrated Security=True
```

- För att ansluta till en SQL Server på den lokala datorn då SQL Server sköter autentiseringen:

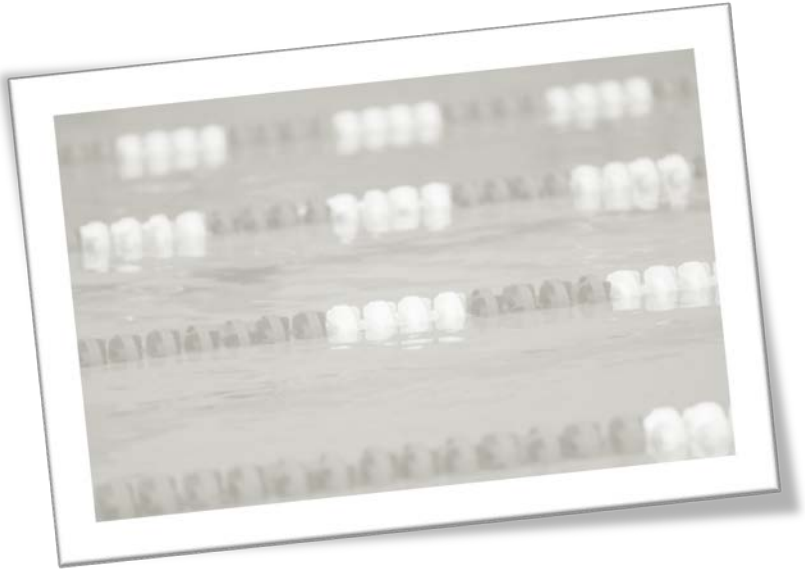
```
Data Source=localhost;Initial Catalog=AdventureWorks;Persist Security Info=True;
User ID=användarnamn;Password=lösenord
```

# Anslutningssträng och web.config

- Anslutningssträngen används av hela applikationen och det är olämpligt att hårdkoda den då ett `SqlConnection`-objekt skapas och initieras.
- Anslutningssträngen sparas lämpligen i web.config. Genom att spara anslutningssträngen i `web.config` behöver inte koden kompileras om vi skulle modifiera anslutningssträngen.
- En annan anledning att lagra anslutningssträngen i `web.config` är att exakt samma anslutningssträng måste användas för att något som heter "connection pooling" ska fungera (vilket är mycket viktigt för att få en så bra prestanda som möjligt).

```
<configuration>
    <appSettings/>
    <connectionStrings>
        <add name=”ApplicationService"
             connectionString="Data Source=localhost;Initial Catalog=AdventureWorks;
                   Persist Security Info=True;User ID=användarnamn;Password=lösenord"
             providerName="System.Data.SqlClient" />
    </connectionStrings>
    <system.web>…</system.web>
</configuration>
```

# ”Connection pooling”



- I en webbapplikation körs ofta flera korta anrop till databasen under kort tid.
- "Connection pooling" används för att undvika att anslutningar kopplas upp och stängs och på så sätt då blir en flaskhalsen i systemet.
- "Connection pooling" innebär att den aktuella webbapplikationen skapar en pool av öppna anslutningar mot databasen som sedan på ett effektivt sätt återanvänds. (Det finns exempel på att en pool med 20 anslutningar till en databas är fullt tillräckligt för 2000 klienter...)
- "Connection pooling" är inbyggt och du behöver inte göra något speciellt för att använda den.

# Skapa en anslutning

# Exekvering av SQL-satser

- Objekt av typen `SqlCommand` använder du för att köra SQL-satser.
- Klassen implementerar interfacet `IDBCommand`, som definierar en uppsättning metoder du kan använda för att exekvera en SQL-sats över en öppen anslutning.
- Det finns tre olika kommandotyper:
  - `CommandType.Text` – exekverar en SQL-fråga.
  - `CommandType.StoredProcedure` – exekverar en lagrad procedur.
  - `CommandType.TableDirect` – hämtar alla poster i en tabell. ANVÄND DEN INTE!
- En fråga kan exekveras på olika sätt beroende på om du är intresserad av alla poster, ett enskilt värde eller inte är intresserad av något värde alls.
  - `ExecuteNonQuery` – exekverar SQL-kommandon (inte SELECT) som används för att lägga till, uppdatera eller ta bort poster. Antalet påverkade poster returneras.
  - `ExecuteScalar` – exekverar en SELECT-fråga och returnerar värdet i det första fältet och första raden i posterna som frågan genererar.
  - `ExecuteReader` – Exekverar en SELECT-fråga och returnerar ett `DataReader`-objekt.

# ExecuteNonQuery

- Metoden `ExecuteNonQuery` returnerar inga poster. Den enda information som returneras är antalet påverkade poster. I exemplet tas inte returvärdet om hand.

```
// Hämtar anslutningssträngen från web.config.
string connectionString = WebConfigurationManager.ConnectionStrings["AdventureWorksConnectionString"].ConnectionString;

// Skapar och initierar ett anslutningsobjekt.
using (var conn = new SqlConnection(connectionString))
{
    try
    {
        // Öppnar anslutningen till databasen.
        conn.Open();

        // Skapar och initierar ett SqlCommand-objekt.
        var cmd = new SqlCommand("DELETE FROM Person.Contact WHERE ContactID = 1", conn);

        // Exekverar SQL-satsen.
        cmd.ExecuteNonQuery();
    }
    catch
    {
        throw new ApplicationException("Database error.");
    }
}
```

# ExecuteScalar

- Metoden `ExecuteScalar` returnerar ett värde av typen `object` varför det behöver typomvandlas (i detta fall till `int?`).
- Om SQL-frågan returnerar flera fält och/eller poster ignoreras de.
- Returnera SQL-frågan inga poster alls, returnerar `ExecuteScalar` värdet `null`.

```
// Hämtar anslutningssträngen från web.config.
string connectionString = WebConfigurationManager.ConnectionStrings["AdventureWorksConnectionString"].ConnectionString;

// Skapar och initierar ett anslutningsobjekt.
using (var conn = new SqlConnection(connectionString))
{
    try
    {
        // Öppnar anslutningen till databasen.
        conn.Open();

        // Skapar och initierar ett SqlCommand-objekt.
        var cmd = new SqlCommand("SELECT COUNT(*) FROM Person.Contact", conn);

        // Exekverar SQL-satsen.
        var numberOfContacts = cmd.ExecuteScalar() as int?;

        // Gör något med numberOfContacts...
    }
    catch
    {
        throw new ApplicationException("Database error.");
    }
}
```

# ExecuteReader och SqlDataReader

- Metoden `ExecuteReader` returnerar ett `SqlDataReader`-objekt.
- Med hjälp av klassen `SqlDataReader` kan du läsa en post i taget av de poster en SELECT-fråga returnerar och som exekverats med metoden `ExecuteReader`.
- Du loopar igenom posterna i `SqlDataReader`-objektet med hjälp av metoden `Read`, som returnerar `true` så länge som det går att läsa en post.
- För att hämta data från ett fält i den aktuella posten använder du t.ex. `GetString`, `GetInt32` eller `GetDataTime`, som alla returnerar ett typat värde. Genom att skicka med ett (0-baserat) index med anropet väljer du vilket fält datat ska hämtas från.
- Du måste alltid stänga ett `SqlDataReader`-objekt då du inte längre behöver det. Du stänger det genom att anropa metoden `Close` (eller genom att använda `using`, som i princip är ett makro som expanderas till en "`try-finally`"-sats).

# ExecuteReader

```
// Lista med OTYPADE referenser (på gränsen till fulhack men passar
// bra i detta korta exempel).
var contacts = new List<dynamic>();

// Hämtar anslutningssträngen från web.config.
string connectionString = WebConfigurationManager.ConnectionStrings["AdventureWorksConnectionString"].ConnectionString;

// Skapar och initierar ett anslutningsobjekt.
using (var conn = new SqlConnection(connectionString))
{
    try
    {
        // Öppnar anslutningen till databasen.
        conn.Open();

        // Skapar och initierar ett SqlCommand-objekt.
        var cmd = new SqlCommand("SELECT FirstName, LastName FROM Person.Contact", conn);

        // SQL-satsen kan returnera flera poster varför ett SqlDataReader-objekt måste ta hand om alla poster.
        // Metoden ExecuteReader skapar ett SqlDataReader-objekt och returnerar en referens till objektet.
        using (var reader = cmd.ExecuteReader())
        {
            // Så länge som det finns poster att läsa returnerar Read true. Finns det inte fler
            // poster returnerar Read false.
            while (reader.Read())
            {
                // Hämtar ut datat för en post. Använder metoden GetString för att hämta datat
                // från specificerad kolumn.
                contacts.Add(new
                {
                    FirstName = reader.GetString(0),
                    LastName = reader.GetString(1)
                });
            }
        }
    }
    catch
    {
        throw new ApplicationException("Database error.");
    }
}

ContactRepeater.DataSource = contacts;
ContactRepeater.DataBind();
```

- ✓ `SqlDataReader`-objektet stängs automatiskt då `using`-blocket är slut. Det är mycket viktigt att stänga det då bara ett åt gången kan användas av en anslutning.

# Exekvera lagrade procedurer

```
// Hämtar anslutningssträngen från web.config.
string connectionString = WebConfigurationManager.ConnectionStrings["AdventureWorksConnectionString"].ConnectionString;

// Skapar och initierar ett anslutningsobjekt.
using (var conn = new SqlConnection(connectionString))
{
    try
    {
        // Skapar och initierar ett SqlCommand-objekt som används till att
        // exekveras specifierad lagrad procedur.
        var cmd = new SqlCommand("dbo.uspGetManagerEmployees", conn);
        cmd.CommandType = CommandType.StoredProcedure;

        // Lägger till den parameter den lagrade proceduren kräver. Använder här det MINDRE effektiva
        // sätttet att göra det på - enkelt, men ASP.NET behöver "jobba" rätt mycket.
        cmd.Parameters.AddWithValue("@ManagerID", 1);

        // Öppnar anslutningen till databasen.
        conn.Open();

        // SQL-satsen kan returnera flera poster varför ett SqlDataReader-objekt måste ta hand om alla poster.
        // Metoden ExecuteReader skapar ett SqlDataReader-objekt och returnerar en referens till objektet.
        using (var reader = cmd.ExecuteReader())
        {
            // Så länge som det finns poster att läsa returnerar Read true, annars false.
            while (reader.Read())
            {
                // Gör något med posten reader-objektet hämtat.
            }
        }
    }
    catch
    {
        throw; // new ApplicationException("Database error.");
    }
}
```

- ✓ Metoden `AddWithValue()` är ett sätt, och det enklaste, att lägga till en parameter till ett kommando.
- ✓ OBS! Glöm inte att ändra `CommandType` till `StoredProcedure`. Standardvärdet är `Text`.

# …och mycket mer finns att läsa…

- ✓ …om ”ADO.NET Fundamentals” i kapitel 7.