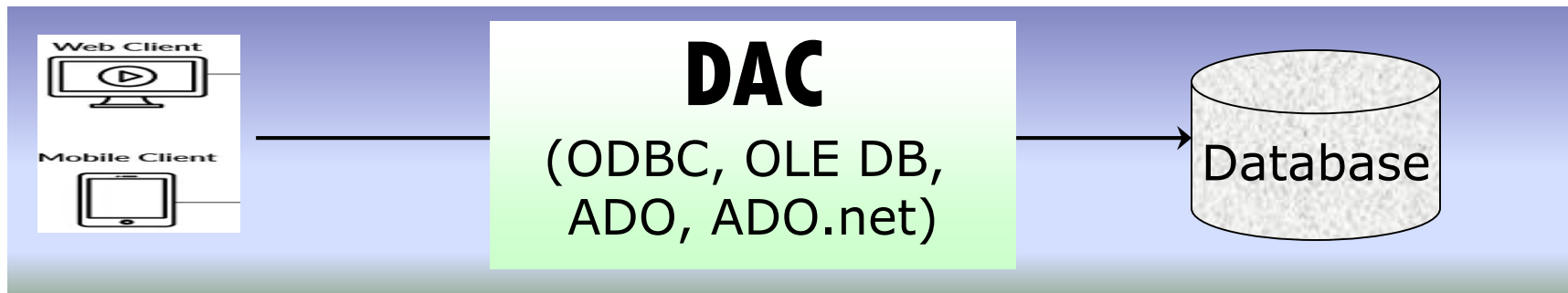
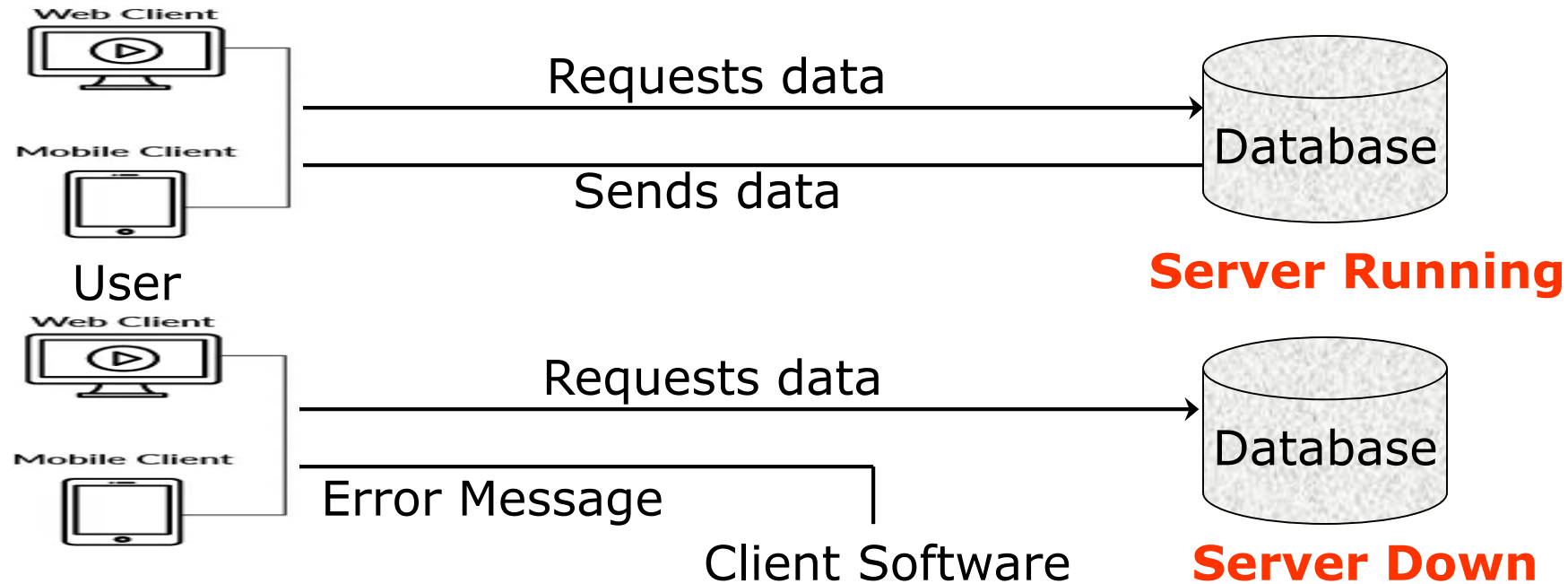


Accessing and Manipulating Cloud Data

Session 8

Data Access Components



Data Access Components [Cont...]

- The client can access the database using different data access components. Some of the data access components are:
 - ODBC
 - OLE DB
 - DAO
 - RDO
 - ADO
 - ADO.NET

Various Data Access Technologies

- ❑ The .NET Framework provides several technologies that you can use in enterprise Web application to work with data stored in a database or other data sources.
- ❑ The key data access technologies of the .NET Framework include:



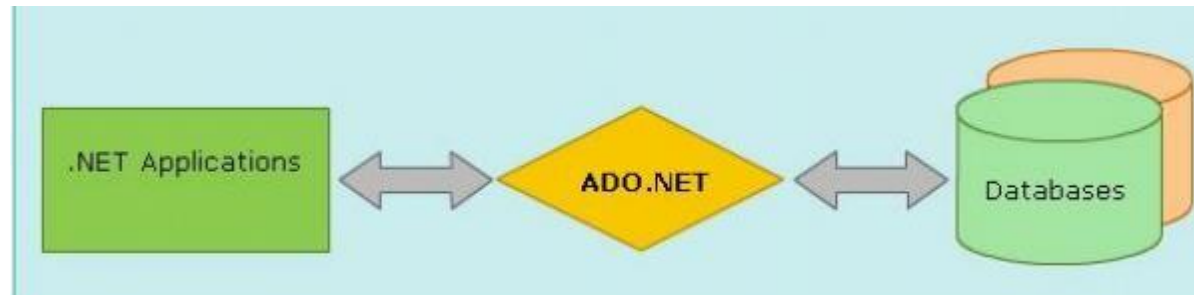
ADO.NET

Entity
Framework

ADO.NET 1-2

❑ ADO.NET:

- Consists of a set of classes provided by the .NET Framework that you can use to access the data stored in a database.



- Helps you to connect with a database, execute commands, and populate a data set that provides forward-only and read-only access to data.

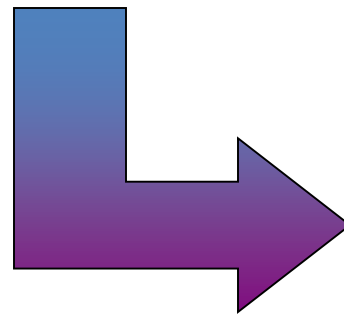


ADO.NET 2-2

Basic Components of ADO.net

DataSet

.Net Data Provider



Connection
Command
DataReader
DataAdapter

Connection Object 1-3

- ❑ It enables an application to connect with a database.
- ❑ The .NET Framework provides the abstract **DbConnection** class to represent a connection to a database.
- ❑ For specific databases, you can use concrete subclasses of the `DbConnection` class.
- ❑ You need to use:
 - **SqlConnection class** to connect to a Microsoft SQL Server database
 - **OracleConnection class** to connect to an Oracle database
 - **ConnectionString property** of the object when you create an object of a connection class

Connection Object 2-3

- ❑ Following code snippet shows an example of creating a `SqlConnection` object and setting the `ConnectionString` property:

```
SqlConnection testConnection = new SqlConnection();  
  
testConnection.ConnectionString = "Data Source = 172.23.3.59;  
Initial Catalog=AppDb; Integrated Security=SSPI; Persist Security  
Info=False";
```

- ❑ In this code:
 - A `SqlConnection` object is created and its `ConnectionString` property is set.
 - The connection string contains the `DataSource` property that specifies the address of the data source.
 - The `InitialCatalog` property specifies the name of the database to access.
 - The `SSPI` value of the `IntegratedSecurity` property specifies that the Windows user account should be used to connect to the database.
 - The `False` value of the `PersistSecurityInfo` property specifies that the authentication information used to connect to the database should be discarded once the connection is established.

Connection Object 3-3

- ❑ Once you have created the connection string, you need to open the connection by calling the `Open()` method of the `SqlConnection` object.
- ❑ Following code snippet opens a connection to the `AppDb` database, as specified in the connection string:

```
testConnection.Open();
```

Command Object

- ❑ Enables an application to execute commands against the database and retrieve results.
- ❑ The `SqlCommand` class represents a command object for Microsoft SQL Server database.
- ❑ This object specifies the SQL statement that needs to be executed and the connection that needs to be used to execute the statement.
- ❑ Following code snippet creates a `SqlCommand` object:

```
SqlCommand cmd = new SqlCommand("Select * from  
employees", testConnection);
```

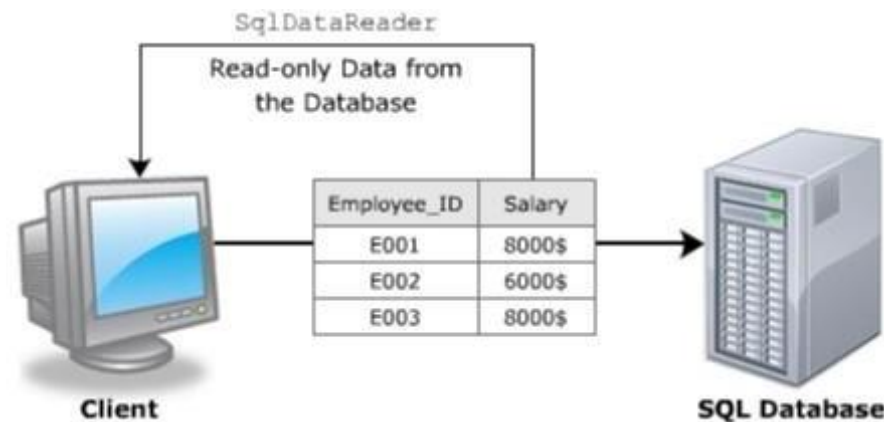
- This code creates a `SqlCommand` object initialized with a SQL SELECT statement and the opened `SqlConnection` object.

Command Object - Methods

Methods	Description
<i>ExecuteNonQuery</i>	This method is used to execute an sql statement that is specified as the command text for the command object. It also returns the number of rows that have been affected or retrieved by the SQL statement.
<i>ExecuteReader</i>	This method is used to execute a stored procedure that is specified as the command text for the command object. It also creates a DataReader object with the retrieved data.
<i>ExecuteScalar</i>	This method is typically used to execute a SQL query that returns a single value, such as an aggregate function (e.g., COUNT, SUM, AVG) or a scalar value from a table.

Data Reader Object 1-2

- ❑ It enables storing the data retrieved by executing the command object.
- ❑ The `SqlDataReader` class represents a data reader for Microsoft SQL Server database.
- ❑ You can create a data reader object by calling the `ExecuteReader()` method of the `SqlCommand` object.



Data Reader Object 2-2

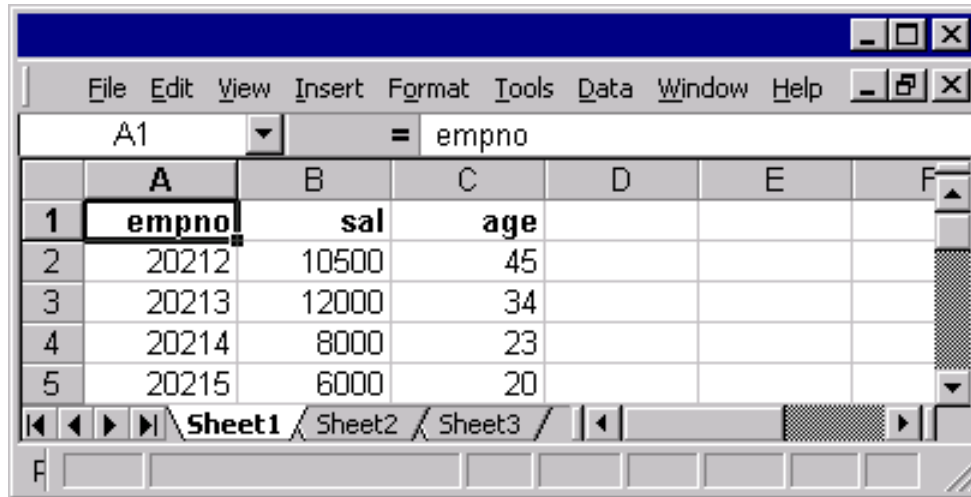
- ❑ Following code snippet creates a data reader object:

```
SqlDataReader dataReader = cmd. ExecuteReader();
```

- ❑ Once you have created the data reader object, you can retrieve a single row of data at a time by calling the `Read()` method of the `SqlDataReader` object.
- ❑ Following code shows reading the first row stored in the data reader object:

```
dataReader.Read();
```

Fetching Single Value



	A	B	C	D	E	F
1	empno	sal	age			
2	20212	10500	45			
3	20213	12000	34			
4	20214	8000	23			
5	20215	6000	20			

count(empno)

Returns a number



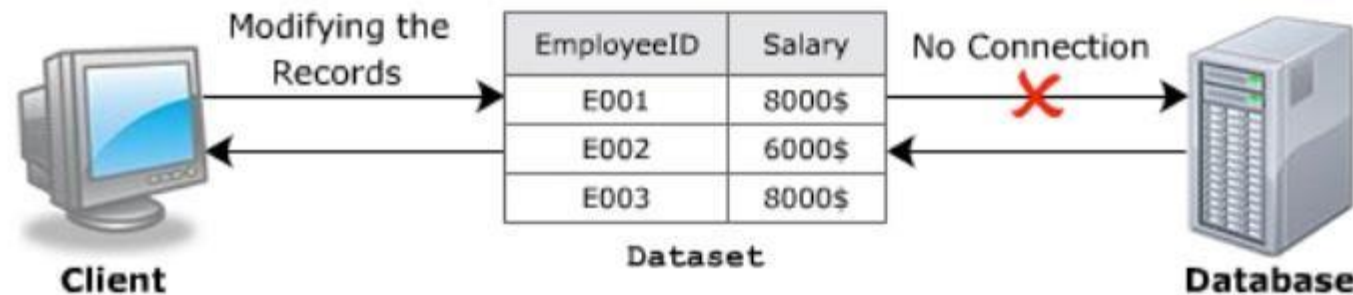
Client
Application

Example:

```
SqlCommand ordersCMD = new SqlCommand("SELECT  
Count(*) FROM Orders", nwindConn);  
Int32 count = (Int32)ordersCMD.ExecuteScalar();
```

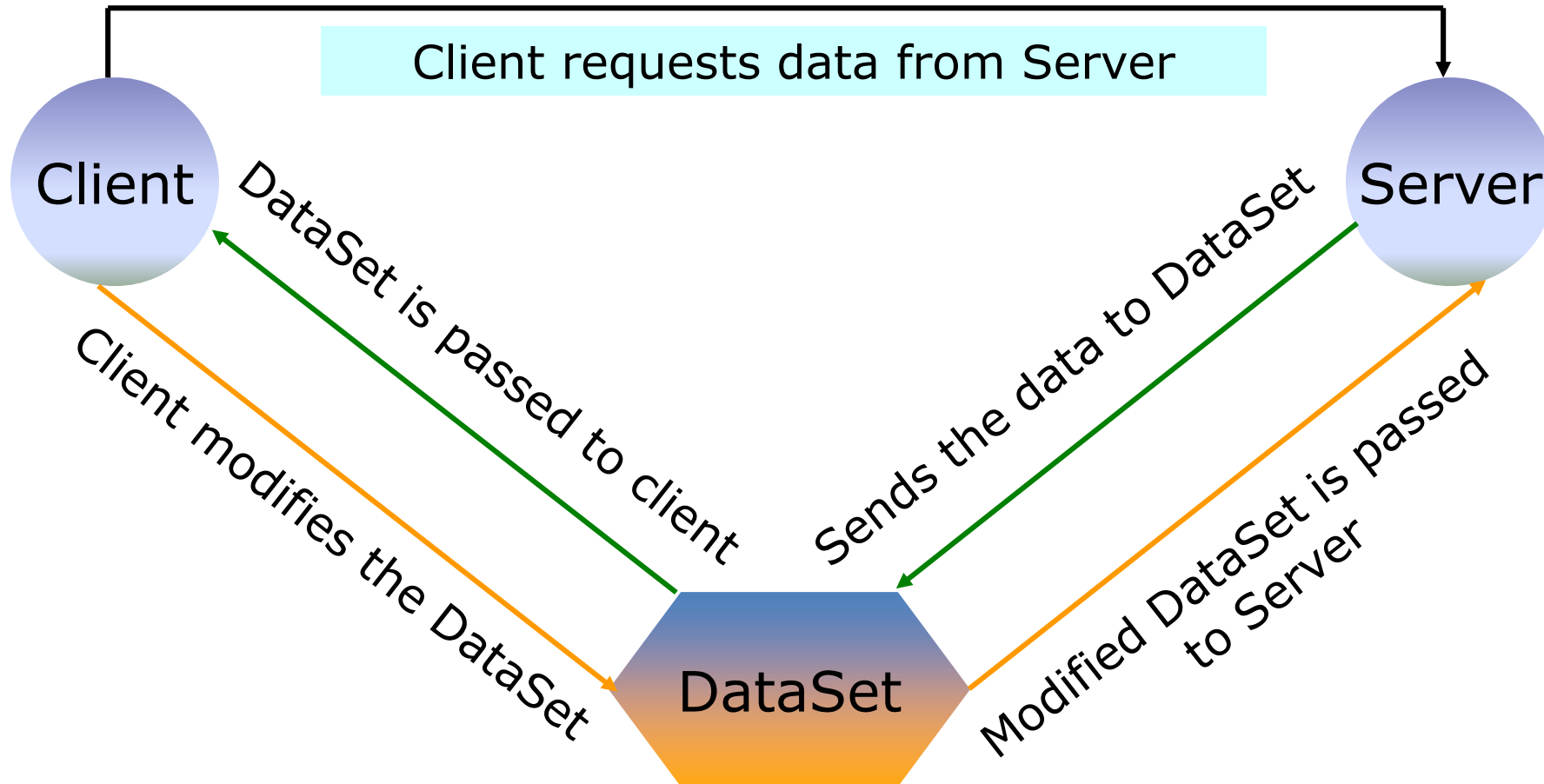
Data Set Object 1-3

- ❑ Represents a memory-based relational representation of data. A dataset is a disconnected, cached set of records that are retrieved from a database.
- ❑ The dataset acts like a virtual database containing tables, rows, and columns.
- ❑ Datasets are extensively used to retrieve data from data sources in Web applications because they do not require the connection to be opened all the time.



- ❑ Instead, they cache data from the database and after that, the connection can be closed.
- ❑ The `Dataset` object requires the data adapter to retrieve data from the data source.

DataSet - Example



Data Set Object 2-3

- ❑ Following code snippet creates a dataset filled with data:

```
SqlConnection testConnection = new SqlConnection();
testConnection.ConnectionString = "Data Source=172.23.3.59;Initial
Catalog=AppDb;Integrated Security=SSPI; Persist Security Info =
False";
DataSet ds = new DataSet();
testConnection.Open();
SqlDataAdapter da = new SqlDataAdapter("Select * from employees",
testConnection);
da.Fill(ds, "employees");
```

- ❑ In this code:
 - The data from the data source is retrieved by the data adapter and filled into the dataset.

Data Set Object 3-3

- ❑ Once a `DataSet` is created with data, the `DataSource` property of a control can be used to bind the `DataSet` with the control.
- ❑ This enables displaying data in the `DataSet` in the control.
- ❑ For example, you can use a `GridView` as a User Interface (UI) control.
- ❑ Following code snippet displays the data in the dataset in a `GridView` control named `gvwEmp`:

```
gvwEmp.DataSource = ds;  
gvwEmp.DataBind();
```

DataView

- Presentation layer for the data stored in DataTable
- Provides view of DataTable for sorting, filtering and searching
- Can be used to view a subset of the data stored in DataTable
- There can be two controls on the same DataTable, which provide different view of the data

DataView [Example]

```
SqlConnection sqlcon;  
DataView dv;  
sqlcon= new SqlConnection("server=SQLDB; uid=sa; pwd=; database=pubs");  
SqlDataAdapter sqlcom = new SqlDataAdapter("select * from employee", sqlcon);  
DataSet ds = new DataSet();  
sqlcom.Fill(ds, "employee");  
dv = new DataView (ds.Tables ['employee']);  
dv.RowFilter = "job_id >10";  
dv.Sort = "lname DESC";  
DataGrid1.DataSource=dv;
```

SqlTransaction Class

- The SqlTransaction class in C# is used to manage database transactions in SQL Server using the ADO.NET data provider.
- A transaction is a unit of work that must be completed in its entirety or not at all, ensuring data integrity and consistency.

```

public void RunSqlTransaction(string myConnString)
{
    SqlConnection myConnection = new SqlConnection(myConnString);
    myConnection.Open();

    SqlCommand myCommand = myConnection.CreateCommand();
    SqlTransaction myTrans;

    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;

    try
    {
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch (Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (SqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to roll back the transaction.");
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

Asynchronous Operations with ADO.NET

- Database operations can take a long time to complete.
- With ADO.NET, you can use asynchronous operations to execute long-running queries without creating and blocking a managed thread.
- ADO.NET supports asynchronous operations:
 - Background execution for long-running operations
 - Freeing up managed threads until the database responds
- You can use asynchronous command operations:
 - `ExecuteNonQueryAsync`
 - `ExecuteReaderAsync`
 - `ExecuteScalarAsync`
- In ASP.net Core, the asynchronous methods return a **Task<T>** object, where the generic type parameter T is the type returned by the corresponding synchronous method. For example, the **ExecuteReaderAsync** method returns a **Task<DbDataReader>** object, whereas the corresponding synchronous method, **ExecuteReader**, returns a **DbDataReader** object.

Cont...

- In ASP.NET Core, the **await** keyword is used to asynchronously wait for the completion of an asynchronous operation, typically represented by a Task or Task<T> object.
- The **await** keyword introduced in C# 5 to schedule a continuation when the operation completes. You can also use the **Task.ContinueWith** method to provide a delegate as the continuation of the task
- **Task.ContinueWith** method is used to chain multiple asynchronous operations together, allowing you to specify a continuation task that will be executed after the original task has completed.
- Other asynchronous methods:
 - `DbConnection.OpenAsync`
 - `DbDataReader.ReadAsync`

Cont...

```
using (var connection = new SqlConnection(connectionString))
{
    await connection.OpenAsync();
    using (var command = new SqlCommand(commandString, connection))
    {
        using (SqlDataReader reader = await command.ExecuteReaderAsync())
        {
            while (await reader.ReadAsync())
            {
                Console.WriteLine("{0}\t{1}", reader.GetInt32(0), reader.GetString(1));
            }
        }
    }
}
```