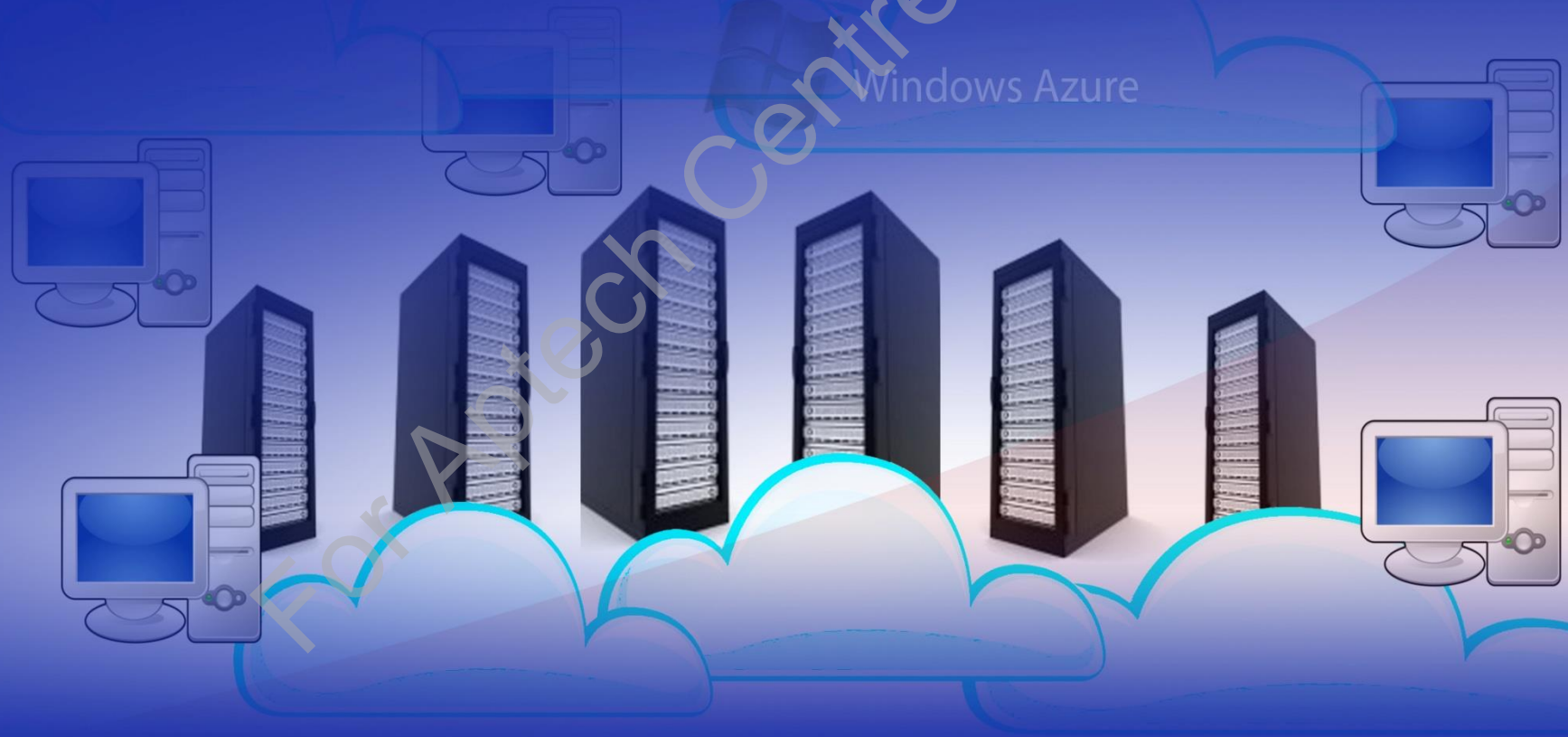


Enterprise Application Development Using Windows Azure and Web Services

Session 5

Accessing and Manipulating Data



Learning Objectives



- Define and describe various data access technologies
- Explain asynchronous data access using ADO.NET
- Explain XML data access and manipulation

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

WCF Data
Services

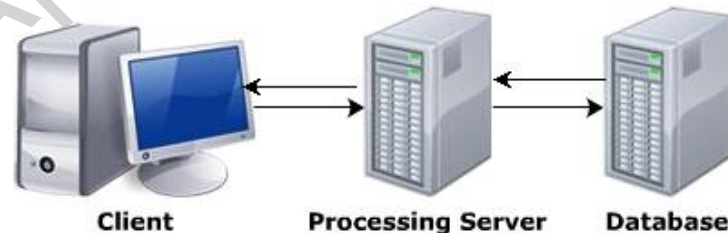
ADO.NET 1-2

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

- ❑ ADO.NET provides the following key objects to enable data access:



Connection Object

Command Object

Data Reader Object

Data Set Object

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

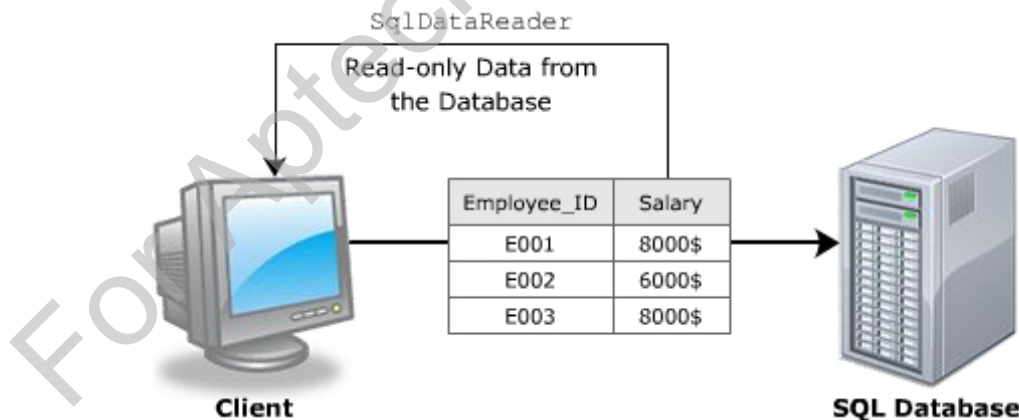
- ❑ Enables an application to execute commands against the database and retrieve results.
- ❑ The `SqlCommand` class represents a command object for Microsoft SQL Server database.
- ❑ This object specifies the SQL statement that needs to be executed and the connection that needs to be used to execute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

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:

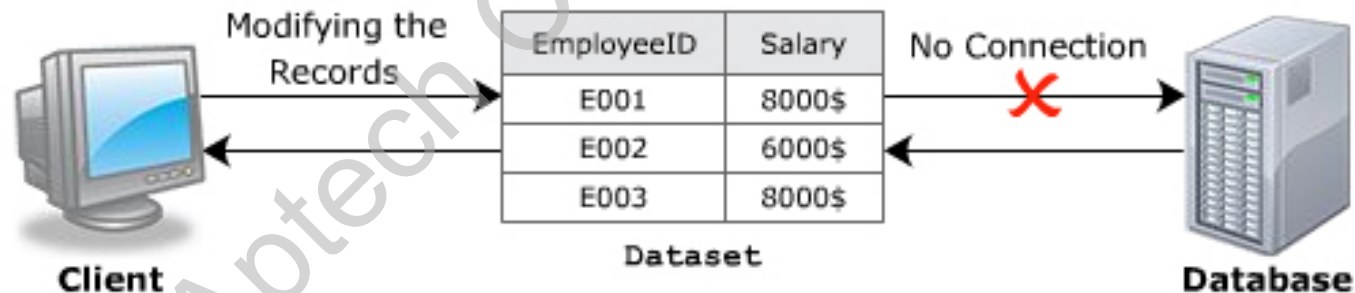
```
SqlDataReader dataReader = cmd. ExecuteReader();
```

- ❑ 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();
```

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.

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();
```

Entity Framework 1-3

In an ASP.NET Web application, you can use an Object Relationship Mapping (ORM) framework to simplify the process of accessing data from the application.

An ORM framework performs the necessary conversions between incompatible type systems in relational databases and object-oriented programming languages.

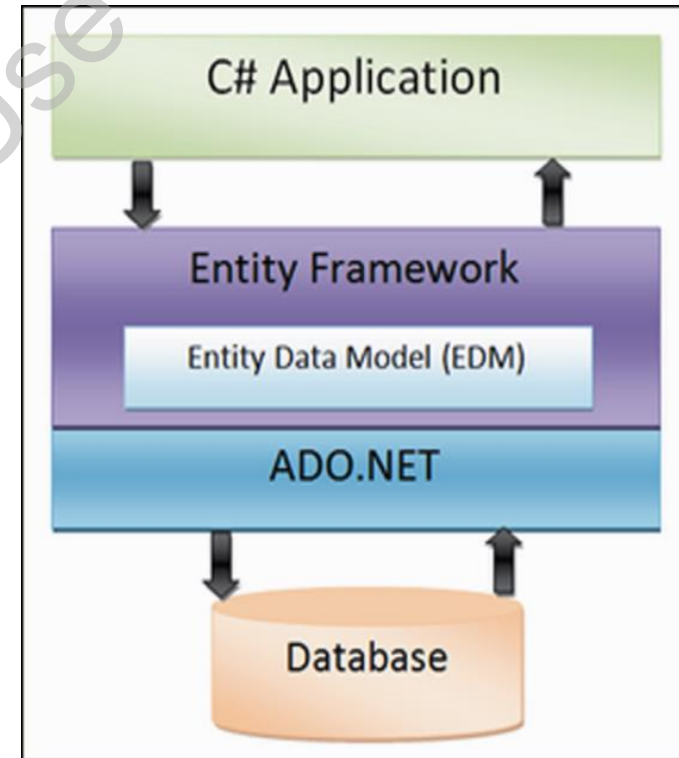
The ADO.NET Entity Framework is an ORM framework often used in .NET applications.

The Entity Framework is an implementation of the Entity Data Model (EDM) describing the entities and the associations that participate in an application.

EDM allows you to handle data access logic by programming against entities without having to worry about the structure of the underlying data store and how to connect with it.

Entity Framework 2-3

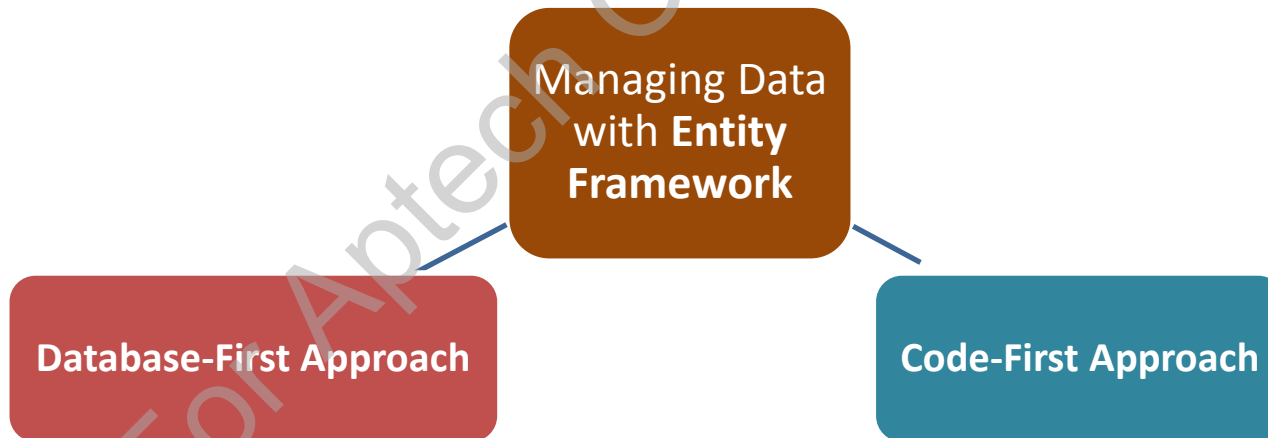
- ❑ The `System.Data.Entity` namespace of the Entity Framework:
 - Provides classes that you can use to synchronize between the model classes and its associated database.
 - Also provides the `DbContext` class that coordinates with Entity Framework and allows you to query and save application data in the database.



Entity Framework 3-3

❑ Entity Framework:

- Eliminates the need to write most of the data-access code that would otherwise need to be written.
- Uses different approaches to manage data related to an application:



Database-First and Code-First

Database-First Approach

- The Entity Framework creates model classes and properties corresponding to the existing database objects, such as tables and columns.
- Applicable in scenarios where a database already exists for the application.

Code-First Approach

- The Entity Framework creates database objects based on custom classes that a programmer creates to represent the entities and their relationships in the application.
- It allows you to develop your application by coding model classes and properties and delegate the process of creating the database objects to the Entity Framework.
- The classes and properties will later correspond to tables and columns in the database.

Code-First Approach

- ❑ Allows you to provide the description of a model by using the C# classes.
- ❑ Based on the class definitions, the code-first conventions detect the basic structure for the model.
- ❑ The `System.Data.Entity.ModelConfiguration.Conventions` namespace provides several code-first conventions that enable automatic configuration of a model.

Conventions

- ❑ Some of these conventions are as follows:

Table Naming Convention

- Entity Framework by default creates a table named `Users` when you have created an object of the `User` model and need to store its data in the database.

Primary Key Convention

- When you create a property named `User Id` in the `User` model, the property is accepted as a primary key.
- The Entity Framework sets up an auto-incrementing key column to hold the property value.

Relationship Convention 1-5

Relationship Convention

- Entity Framework provides different conventions to identify a relationship between two models.
- You can use navigational properties in order to define relationship between two models.
- You should define a foreign key property on types that represents dependent objects.

Relationship Convention 2-5

❑ Consider a scenario:

- You are developing an online shopping store.
- For the application, you have created two model classes named `Customer` and `Order`.
- Now, you need to declare properties in each class that allows navigating to the properties of another class.
- You can then, define the relationship between these two classes.

Online Shopping Store



Relationship Convention 3-5

- ❑ Following code snippet creates the Customer model class:

```
public class Customer
{
    public int CustId { get; set; }
    public string Name { get; set; }
    // Navigation property
    public virtual ICollection<Order> Orders { get; set; }
}
```

- This code creates a model named `Customer` that contains two properties named `CustId` and `Name`.

Relationship Convention 4-5

- ❑ Following code snippet creates the Order model class:

```
public class Order
{
    public int Id { get; set; }

    public string ProductName { get; set; }

    public int Price { get; set; }

    // Foreign key

    public int CustId { get; set; }

    // Navigation properties
    public virtual Customer cust { get; set; }
}
```

Relationship Convention 5-5

❑ In the code:

- **Orders**: Is the navigational property in the `Customer` class.
- **Cust**: Is the navigational property in the `Order` class.
- These two properties are known as navigational properties as they allow to navigate to the properties of another class.

❑ For example:

- You can use the `cust` property to navigate to the orders associated with that customer.
- In the `Customer` class, the `Orders` navigational property is declared as a collection, as one customer can place multiple orders.
- This indicates a one-to-many relationship between the `Customer` and `Order` classes.
- The `CustId` property in the `Order` class is inferred as the foreign key by Entity Framework.

Database Context 1-10

- ❑ The `System.Data.Entity` namespace provides a `DbContext` class.
- ❑ After creating the model class, you can use the `DbContext` class to define the database context class.
- ❑ This class coordinates with Entity Framework and allows you to query and save the data in the database.
- ❑ The database context class uses the `DbSet<T>` type to define one or more properties.
- ❑ In the type, `DbSet<T>`, `T` represents the type of an object that needs to be stored in the database.

Database Context 2-10

- ❑ Following code snippet shows how to use the DbContext class:

```
public class OLShopDataContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

- ❑ In the code:
 - A database context class named OLShopDataContext is created that derives from the DbContext class.
 - This class creates the DbSet property for both the Customer class and the Product class.

Database Context 3-10

❑ In a Web application:

- You might need to change the model classes to implement various new features.
- This also requires maintaining the database related to the model based on the changes made in the model class.
- So while modifying the model classes, you should ensure that any changes in the model are reflected back in the database.
- To maintain the synchronization between the model classes and its associated database, you need to recreate databases.

Database Context 4-10

- ❑ Entity Framework provides the `System.Data.Entity` namespace that contains the following two classes to recreate databases:

`DropCreateDatabaseAlways`

- Allows recreating an existing database whenever the application starts.

`DropCreateDatabaseIfModelChanges`

- Allows recreating an existing database whenever the associated model class changes.

- ❑ Based on your requirements, you can use one of these two classes in your application:
 - To recreate a database.
 - While calling the `SetInitializer()` method of the Database class defined in the `System.Data.Entity` namespace.

Database Context 5-10

❑ Following code snippet shows:

- Creation of a new instance of the `DropCreateDatabaseAlways` class inside the `Application_Start()` method of the `Global.asax.cs` file:

```
. . .  
Database.SetInitializer(new  
    DropCreateDatabaseAlways<ShopDataContext>());
```

❑ In this code:

- The `DropCreateDatabaseAlways` class is used while calling the `SetInitializer()` method to ensure that the existing database is recreated whenever the application starts.
- You can use the `DropCreateDatabaseIfModelChanges` class to recreate a database only when the model changes.

Database Context 6-10

❑ The code snippet demonstrates:

- Creation of an instance of the `DropCreateDatabaseIfModelChanges` class inside the `Application_Start()` method of the `Global.asax.cs` file:

```
Database.SetInitializer(new  
DropCreateDatabaseIfModelChanges<ShopDataContext>());
```

❑ In this code:

- The `DropCreateDatabaseIfModelChanges` class is used while calling the `SetInitializer()` method to ensure that the existing database is recreated whenever the model changes.

Database Context 7-10

- ❑ You can also populate a database with sample data for an application by creating a class that derives from either:
 - The `DropCreateDatabaseIfModelChanges` class or
 - The `DropCreateDatabaseAlways` class
- ❑ In this class:
 - You need to override the `Seed()` method that enables you to define the initial data for the application.

Database Context 8-10

❑ Following code snippet shows:

- The `MyDbInitializer` class that uses the `Seed()` method to insert some sample data in the `Customers` database:

```
public class MyDbInitializer :
DropCreateDatabaseIfModelChanges<OLShopDataContext>
{
    protected override void Seed(OLShopDataContext context)
    {
        context.Customers.Add(new Customer() { Name = "John
Parker", Address="Park Street", Email =
"john@webexample.com" });base.Seed(context);
    }
}
```

Database Context 9-10

❑ In the code:

- The `MyDbInitializer` class is derived from the `DropCreateDatabaseIfModelChanges` class.
- Then, the `Seed()` method is overridden to define the initial data for the `Customer` model.
- After defining the initial data for the customers, you need to register the `MyDbInitializer` model class in the `Global.asax.cs` file by calling the `SetInitializer()` method.

Database Context 10-10

❑ Following code snippet shows:

- Use of the `SetInitializer()` method inside the `Application_Start()` method:

```
protected void Application_Start()  
{  
    System.Data.Entity.Database.SetInitializer(new  
        MyDbInitializer());  
}
```

- This code uses the `SetInitializer()` method to register the `MyDbInitializer` model class.

Implementing the Code-First Approach 1-5

- ❑ To implement a code-first approach using Visual Studio 2013, you need to perform the following steps:

Step 1

- Create an ASP.NET MVC Web application named **EFCodeFirstDemo**.

Step 2

- Right-click the **Models** folder in the **Solution Explorer** window. The context menu options of the **Models** folder are displayed.

Step 3

- Click **Add** → **New Item**. The **Add New Item - EFCodeFirstDemo** dialog box is displayed.

Implementing the Code-First Approach 2-5

Step 4

- Select **Code** in the left pane of the **AddNewItem - EFCodeFirstDemo** dialog box and the **Class** template in the right pane.

Step 5

- Type `Employee.cs` in the **Name** field.

Step 6

- Click **Add**. The Code Editor window displays the default code of the `Employee` model class.

Step 7

- Add the code in the `Employee` class to define three properties named `Name`, `Designation`, and `Department`.

Step 8

- To create the database context class, add a new class named `AppDBContext` to the project.

Implementing the Code-First Approach 3-5

Step 9

- Create another class named `AppData` to the project to update the `AppData` class to create a database whenever the model class changes.

Step 10

- Right-click `Global.asax` file in the **Solution Explorer** window. The Code Editor window displays the code of the `Global.asax` file.

Step 11

- To register the `AppData` class, use the `SetInitializer()` method inside the `Application_Start()` method of the `Global.asax` file.

Step 12

- Add a controller named `EmployeeController` to the project using the MVC 5 Controller - Empty template.

Step 13

- Right-click inside the **Index** action method and select **Add View**. The **Add View** dialog box is displayed.

Implementing the Code-First Approach 4-5

Step 14

- Select **Create** from the **Scaffold template** drop-down list and then, select **Employee(EFCodeFirstDemo.Models)** from the **Model** class drop-down list.

Step 15

- Click **Add**. The Code Editor window displays the code of the **Index.cshtml** view.

Step 16

- Add another view named **Message** for the `Message()` action method. Update the code of the Message view to display a message to indicate that user specified data is added to the database.

Step 17

- In the menu bar, click **Debug** → **Start Without Debugging**. The Home page of the project is displayed in a browser.

Step 18

- In the address bar of the browser, type the URL: <http://localhost:2175/Employee/Index>. The browser displays the Index view.

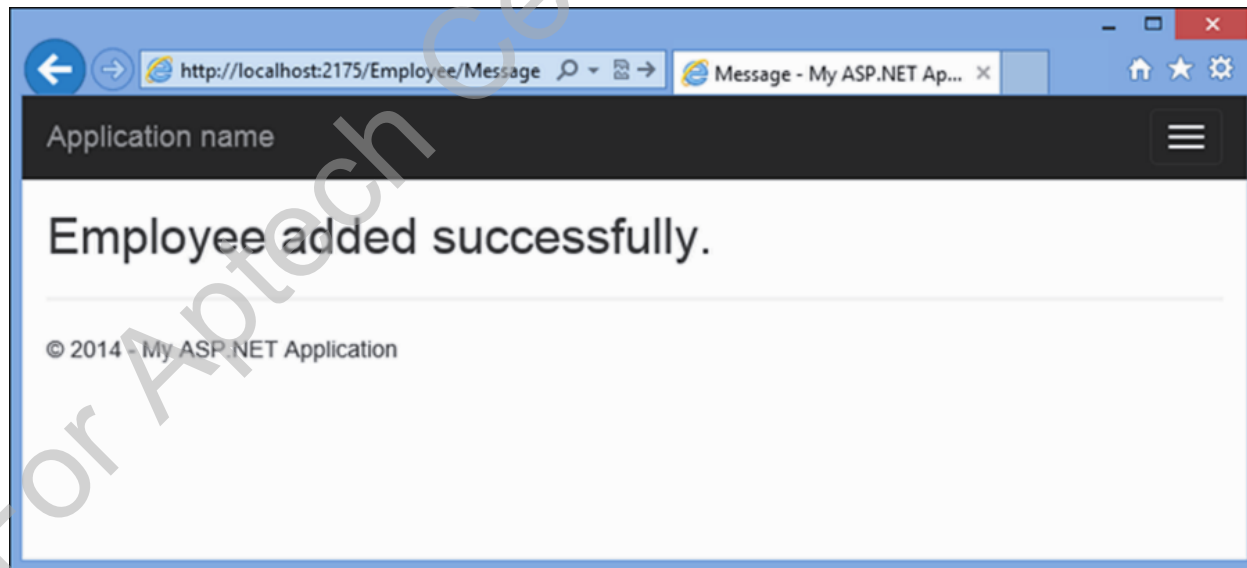
Implementing the Code-First Approach 5-5

Step 19

- Specify employee data in the Name, Designation, and Department fields.

Step 20

- Click **Create**. The browser displays the Message view with a message indicating that the user-specified employee data has been added to the database. The figure shows the Message view:



WCF Data Services 1-2

❑ WCF Data Services:

Are provided by the .NET Framework

Formerly known as ADO.NET Data Services

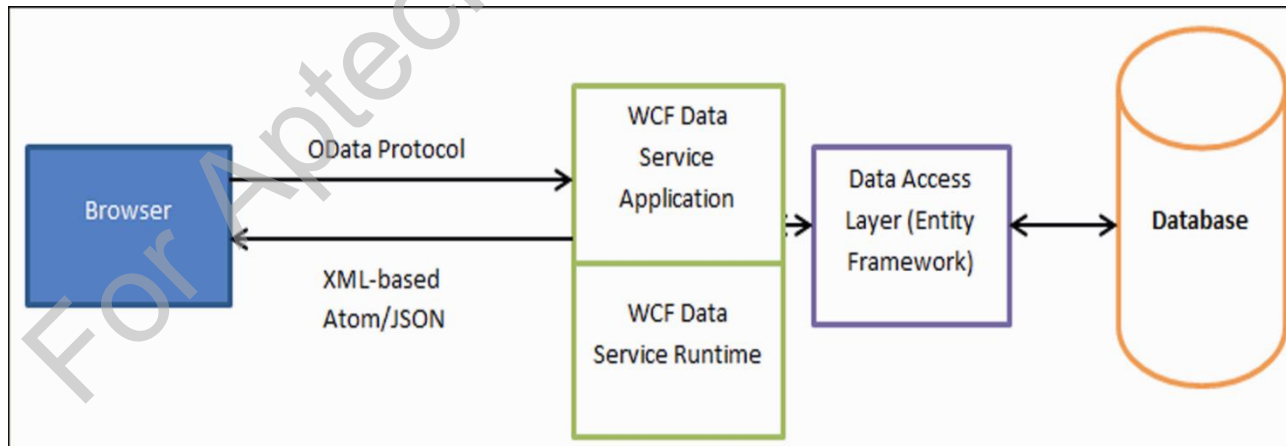
Can be used to create services using the Open Data Protocol (OData)

❑ OData:

- Is a REST-based protocol that allows you to perform operations on resources, such as Create, Read, Update, and Delete (CRUD) through Uniform Resource Locators (URLs).
- Uses the Entity Data Model of the Entity Framework to expose resources as related entities.

WCF Data Services 2-2

- ❑ You can perform CRUD operations on the entities using the standard HTTP methods such as:
 - GET, PUT, POST, and DELETE
- ❑ The result returned by the WCF data service can be in XML-based Atom and JSON formats.
- ❑ Following figure shows the role of WCF data service in an enterprise application:



Creating the Database 1-2

- ❑ To create the database and table in Visual Studio 2013, you need to perform the following tasks:

Step 1

- Open the **SQL Server Object Explorer** window in Visual Studio 2013.

Step 2

- Expand the **(Localdb)\v11.0 → Databases** node in the **SQL Server Object Explorer** window.

Step 3

- Right-click the **Databases** node and select **Add New Database** from the context menu. The **Create Database** dialog box is displayed.

Step 4

- Type **EmployeeDB** in the **Database Name** field.

Creating the Database 2-2

Step 5

- Click **OK**. The SQL Server Object Explorer window displays the newly added database under the **Databases** node.

Step 6

- Expand the **EmployeeDB** node.

Step 7

- Right-click the **Tables** node and select **Add New Table** from the context menu. The **Table Design** window is displayed.

Step 8

- Specify the name of the table fields, such as Name, Designation, and Department, then specify the data types for each fields.

Step 9

- Specify the name of the table as **Employee** in the T-SQL window that displays the SQL statement to create the table.

Step 10

- Click **Update**. The **Preview Database Updates** dialog box is displayed.

Step 11

- Click **Update Database**. The SQL Server Object Explorer window displays the newly created **Employee** table under the **Tables** node.

Creating the EDM 1-4

- ❑ To create the EDM in Visual Studio 2013, you need to perform the following tasks:

Step 1

- Create an ASP.NET Web Application project by selecting the **Empty** template and name it as **WCFFDataServiceDemo**.

Step 2

- Right-click **WCFFDataServiceDemo** in the Solution Explorer window and select **Add → New Item**. The **Add New Item – WCFFDataServiceDemo** dialog box is displayed.

Step 3

- Select **Data** in the left pane and **ADO.NET Entity Data Model** in the right pane of the **Add New Item – WCFFDataServiceDemo** dialog box.

Creating the EDM 2-4

Step 4

- Specify the name of the model as **EmpModel.edmx**.

Step 5

- Click **Add**. The **Entity Data Model Wizard** dialog box is displayed.

Step 6

- Select **Generate from database** in the Entity Data Model Wizard dialog box.

Step 7

- Click **Next**. The **Choose Your Data Connection** screen is displayed.

Step 8

- Click **New Connection**. The **Connection Properties** dialog box is displayed.

Creating the EDM 3-4

Step 9

- Specify **(localdb)\v11.0** in the **Server Name** field and select **EmployeeDB** from the **Select or enter a database name** drop-down list.

Step 10

- Click **OK**. The **Choose Your Data Connection** screen displays the new connection and the database name.

Step 11

- Click **Next**. The **Choose Your Version** screen is displayed.

Step 12

- Select the **Entity Framework 5.0** option.

Step 13

- Click **Next**. The **Choose Your Database Objects and Settings** screen is displayed.

Creating the EDM 4-4

Step 14

- Select the **Tables** check box.

Step 15

- Click **Finish**. The Entity Data Model Designer displays the newly added **Employee** entity.

Step 16

- Build the project.

Creating the WCF Data Service

1-4

- ❑ To create WCF Data Service in Visual Studio 2013, you need to perform the following tasks:

Step 1

Right-click **WCFFDataServiceDemo** in the Solution Explorer window and select **Add → New Item**. The **Add New Item – WCFFDataServiceDemo** dialog box is displayed.

Step 2

Select **Web** in the left pane and **WCF Data Service 5.6** in the right pane of the **Add New Item – WCFFDataServiceDemo** dialog box.

Step 3

Click **Add**. The Code Editor window displays the code of the newly added data service **EmpWcfDataService.svc.cs**.

Creating the WCF Data Service

2-4

Step 4

Update the code of the **EmpWcfDataService.svc.cs** data service.

Step 5

Open Internet Explorer. From the **Tools** menu, select **Internet Options**. The **Internet Options** dialog box is displayed.

Step 6

Click the **Content** tab.

Step 7

Click **Settings** under the **Feeds and Web Sites** section. The **Feeds and Web Sites** dialog box is displayed.

Step 8

Clear the **Turn on feed reading view** check box.

Step 9

Click **OK**. The main screen of the **Internet Options** dialog box is displayed.

Creating the WCF Data Service

3-4

Step 10

Click **OK** to close the dialog box.

Step 11

Execute the project. The browser displays the output of the data service.

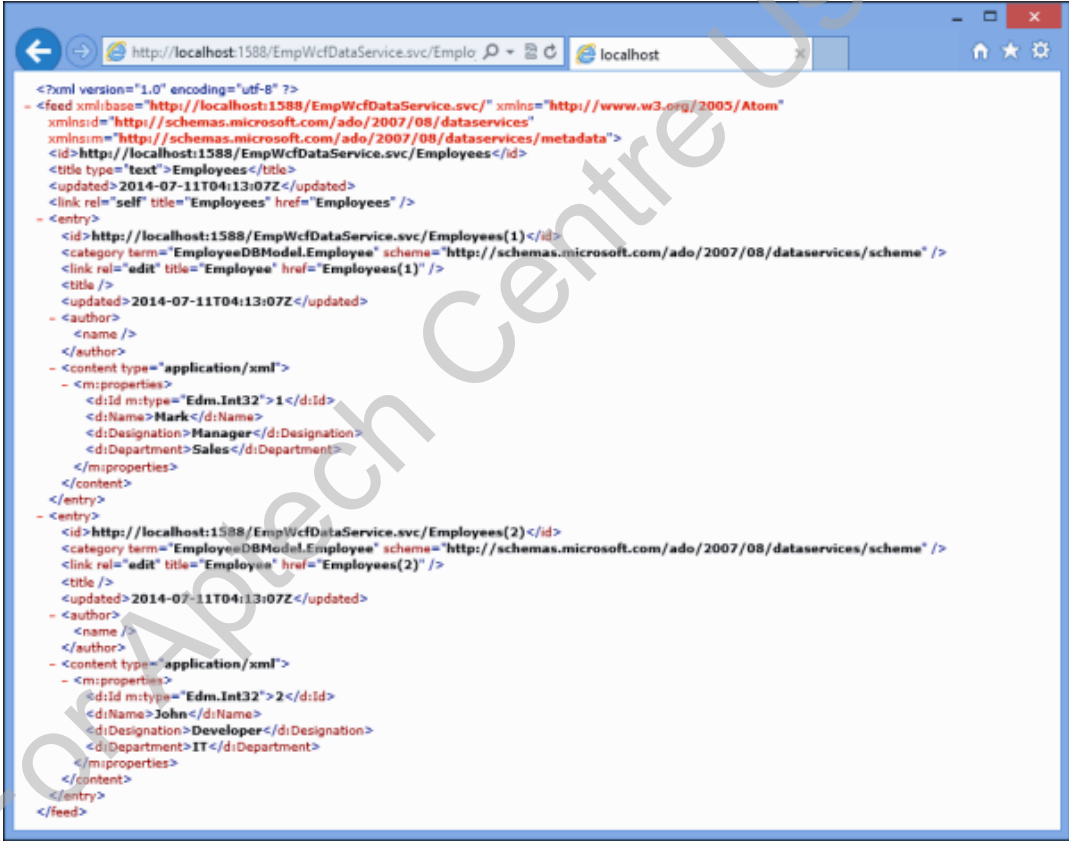
Step 12

Type the following URL in the address bar of the browser:
`http://localhost:1588/EmpWcfDataService.svc/Employees.`
The browser displays the employee data.

Creating the WCF Data Service

4-4

□ Following figure shows the employee data in the browser:



The screenshot shows a web browser window with the address bar displaying `http://localhost:1588/EmpWcfDataService.svc/Employ`. The browser content displays an Atom feed in XML format. The feed contains two entries, each representing an employee. The first entry is for an employee named Mark, with the designation Manager and Department Sales. The second entry is for an employee named John, with the designation Developer and Department IT. The XML is color-coded, with red text for the root elements and blue text for the entry details.

```
<?xml version="1.0" encoding="utf-8" ?>
- <feed xmlns="http://localhost:1588/EmpWcfDataService.svc/" xmlns="http://www.w3.org/2005/Atom"
  xmlnsid="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlnsim="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
  <id>http://localhost:1588/EmpWcfDataService.svc/Employees</id>
  <title type="text">Employees</title>
  <updated>2014-07-11T04:13:07Z</updated>
  <link rel="self" title="Employees" href="Employees" />
  - <entry>
    <id>http://localhost:1588/EmpWcfDataService.svc/Employees(1)</id>
    <category term="EmployeeDBModel.Employee" schema="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <link rel="edit" title="Employee" href="Employees(1)" />
    <title />
    <updated>2014-07-11T04:13:07Z</updated>
    - <author>
      <name />
    </author>
    - <content type="application/xml">
      - <m:properties>
        <d:Id m:type="Edm.Int32">1</d:Id>
        <d:Name>Mark</d:Name>
        <d:Designation>Manager</d:Designation>
        <d:Department>Sales</d:Department>
      </m:properties>
    </content>
  </entry>
  - <entry>
    <id>http://localhost:1588/EmpWcfDataService.svc/Employees(2)</id>
    <category term="EmployeeDBModel.Employee" schema="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <link rel="edit" title="Employee" href="Employees(2)" />
    <title />
    <updated>2014-07-11T04:13:07Z</updated>
    - <author>
      <name />
    </author>
    - <content type="application/xml">
      - <m:properties>
        <d:Id m:type="Edm.Int32">2</d:Id>
        <d:Name>John</d:Name>
        <d:Designation>Developer</d:Designation>
        <d:Department>IT</d:Department>
      </m:properties>
    </content>
  </entry>
</feed>
```


Asynchronous Data Access with ADO.NET

- ❑ Enterprise application needs to be responsive and the responsiveness of an application depends on how the application performs data access.
- ❑ Accessing a database across a network can cause delays.
- ❑ Delays also occur when the database processes a query and returns some results.
- ❑ Delays affect the responsiveness of an application.
- ❑ A solution to avoid such delays caused due to data access is to implement asynchronous data access in the application.



async Keyword 1-2

❑ **async Keyword:**

- Enables support for asynchronous programming in .NET Framework 4.5.
- Can be used to asynchronously access data.
- A method marked with this keyword notifies the compiler that the method will contain at least one `await` keyword.
- If the compiler finds a method marked as `async` but without an `await` keyword, it reports a compilation error.
- The `await` keyword is applied to an operation to temporarily stop the execution of the `async` method until the operation completes.
- In the meantime, control returns to the `async` method's caller.
- Once the operation marked with `await` completes, execution resumes in the `async` method.

*Temporarily
stop
execution*



async Keyword 2-2

- ❑ A method marked with the `async` keyword can have either one of the following return types:

`void`

`Task`

`Task<TResult>`

- ❑ Some methods for asynchronous programming with the ADO.NET classes in .NET Framework 4.5 are:

Method	Description
<code>SqlConnection.OpenAsync()</code>	Asynchronously opens a database connection.
<code>SqlCommand.ExecuteNonQueryAsync()</code>	Asynchronously executes a SQL statement on a database connection.
<code>SqlDataReader.ReadAsync()</code>	Asynchronously advances a reader to the next record of a result set.
<code>SqlDataReader.NextResultAsync()</code>	Asynchronously advances a reader on the results of a batch of statements.

Accessing and Manipulating XML

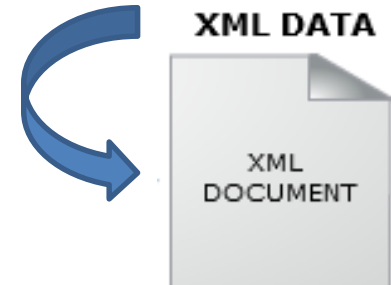
Data 1-7

- ❑ Enterprise applications often need to store data in XML format.
- ❑ In such applications, you will need to access data from XML documents.
- ❑ For example:

Consider an XML document used to store names and e-mail IDs of employees in a Web application.



EmployeeId	Name	Salary
1	Terry Bull	75000
2	Smith Waltz	10000
3	Billy John	15000



Accessing and Manipulating XML Data 2-7

- ❑ Following code snippet creates the `Employees.xml` file:

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
<user>
<name>John Smith</name>
<email>john.smith@webexample.com</email>
</user>
<user>
<name>Mark Parker</name>
<email>mark.parker@webexample.com</email>
</user>
</root>
```

- ❑ This code:
 - Creates a `<root>` element with two `<user>` sub elements.
 - Each `<user>` sub element further contains the `<name>` and `<email>` sub elements.

Accessing and Manipulating XML

Data 3-7

- ❑ In order to read the data from the XML file, you need to load the XML file into an `XmlDocument` object.
- ❑ This object represents an XML document.
- ❑ You can then retrieve the `<root>` element of the XML document as an `XmlNode` object.
- ❑ Once you have access to the `XmlNode` object, you can use a `foreach` loop to traverse the different elements of the document and access the values of the elements.

Accessing and Manipulating XML

Data 4-7

- Following code snippet uses the `XmlDocument` class to read data stored in the `employees.xml` file.

```
XmlDocument doc = new XmlDocument();  
doc.Load(Server.MapPath("~/Xml/Employees.xml"));  
XmlNode node = doc["root"];  
foreach (XmlNode childNode in node.ChildNodes)  
{  
    string value = childNode.InnerText;  
}
```

- This code creates the `XmlDocument` object and calls the `Load()` method to load the content of the `employees.xml` file in the `XmlDocument` object.
- An `XmlNode` object that represents the `<root>` element is then retrieved.
- Finally, a `foreach` loop traverses the `XmlNode` object to retrieve the values of the XML elements.

Accessing and Manipulating XML Data 5-7

- ❑ You can also use the following classes to manipulate an XML document:

XmlDocument

XmlNode

- ❑ For example:

Consider that you need to update the value of the `<email>` element whose `<name>` element contains the value John Smith.

Accessing and Manipulating XML

Data 6-7

- ❑ Following code snippet updates the `employees.xml` document:

```
XmlDocument doc = new XmlDocument();
doc.Load(Server.MapPath("~/Xml/Employees.xml"));
XmlNode node = doc["root"];

foreach (XmlNode childNode in node.ChildNodes)
{
    if (childNode["name"].InnerText.Equals("John Smith"))
    {
        childNode["email"].InnerText = "johns@webexample.com";
    }
}
doc.Save(Server.MapPath("~/Xml/Employees.xml"));
```

Accessing and Manipulating XML

Data 7-7

- ❑ This code creates a `XmlDocument` object and calls the `Load()` method to load the content of the `Employees.xml` file in the `XmlDocument` object.
- ❑ An `XmlNode` object that represents the `<root>` element is then retrieved.
- ❑ A `foreach` loop traverses the `XmlNode` object to retrieve the values of the XML elements.
- ❑ Inside the `foreach` loop, an `if` loop checks for the `XmlNode` object that represents the `<name>` element with the value is John Smith.
- ❑ When that `XmlNode` is encountered, the value of the `XmlNode` that represents the `<email>` element is changed to `johns@webexample.com`.

Summary 1-2

- ❑ The .NET Framework provides various data access technologies, such as ADO.NET, Entity Framework, and WCF Data Services.
- ❑ ADO.NET allows you to connect with a database, execute commands, and populate a dataset that provides forward-only and read-only access to data.
- ❑ In an ASP.NET Web application, to simplify the process of accessing data from the application, you can use an Object Relationship Mapping (ORM) framework.

Summary 2-2

- ❑ The Entity Framework is an implementation of the EDM, which is a conceptual model that describes the entities and the associations they participate in an application.
- ❑ The .NET Framework provides WCF data service that you can use to create services using OData.
- ❑ You can use the standard HTTP methods, such as GET, PUT, POST, and DELETE to perform CRUD operations on the entities.
- ❑ The `XmlDocument` and `XmlNode` classes enable accessing and manipulating XML data.