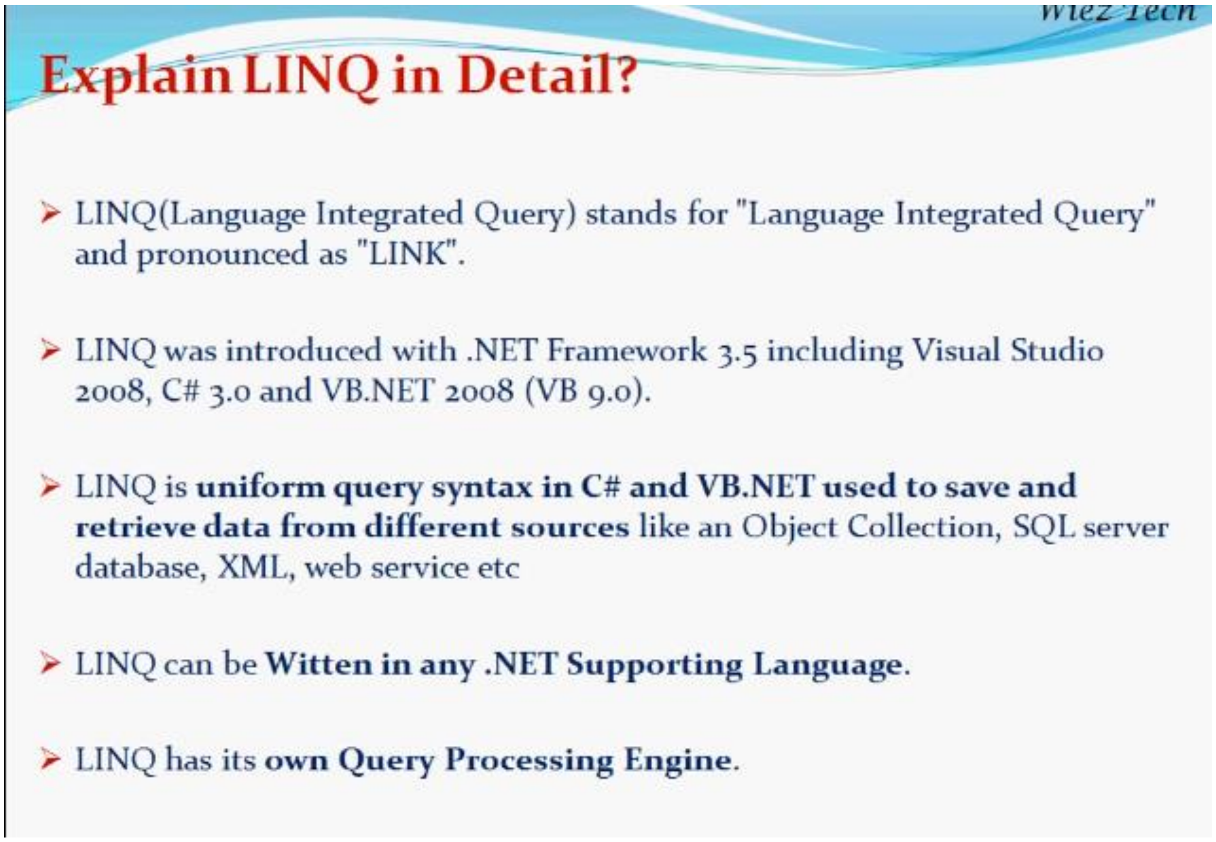
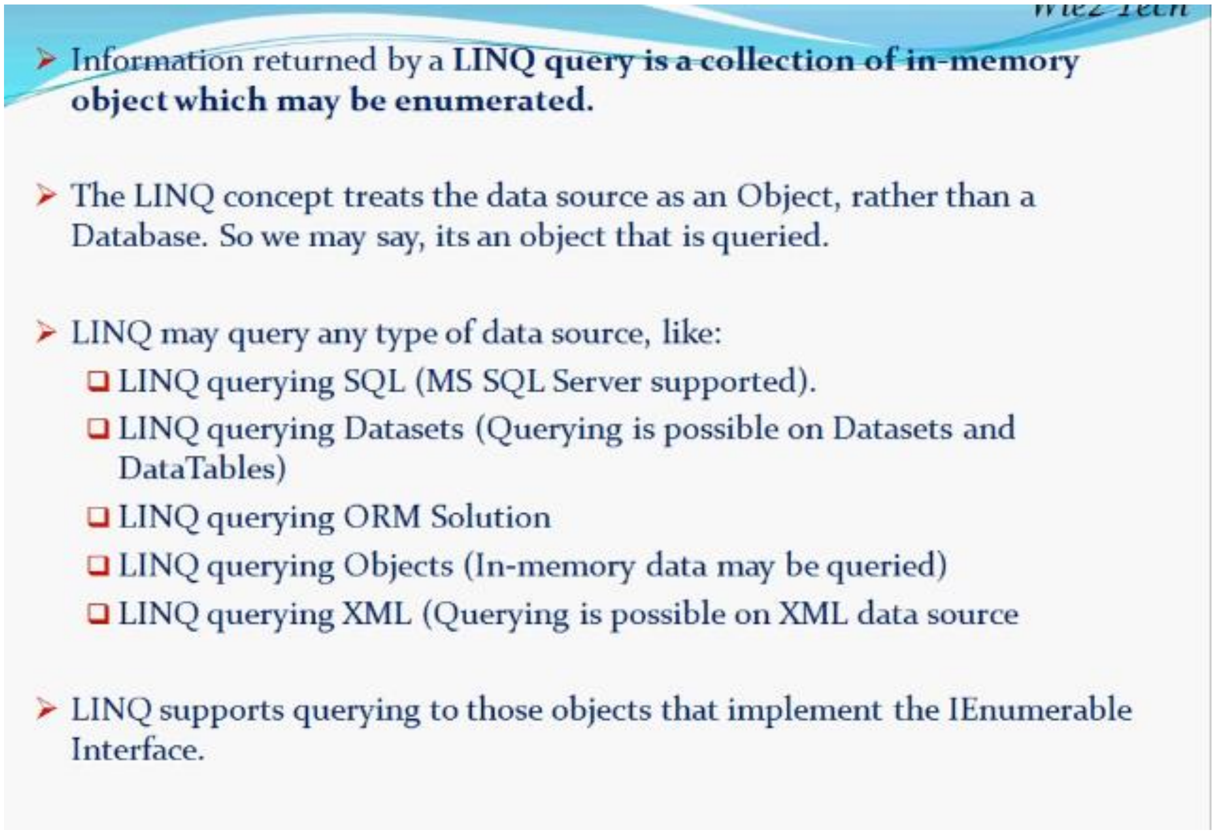
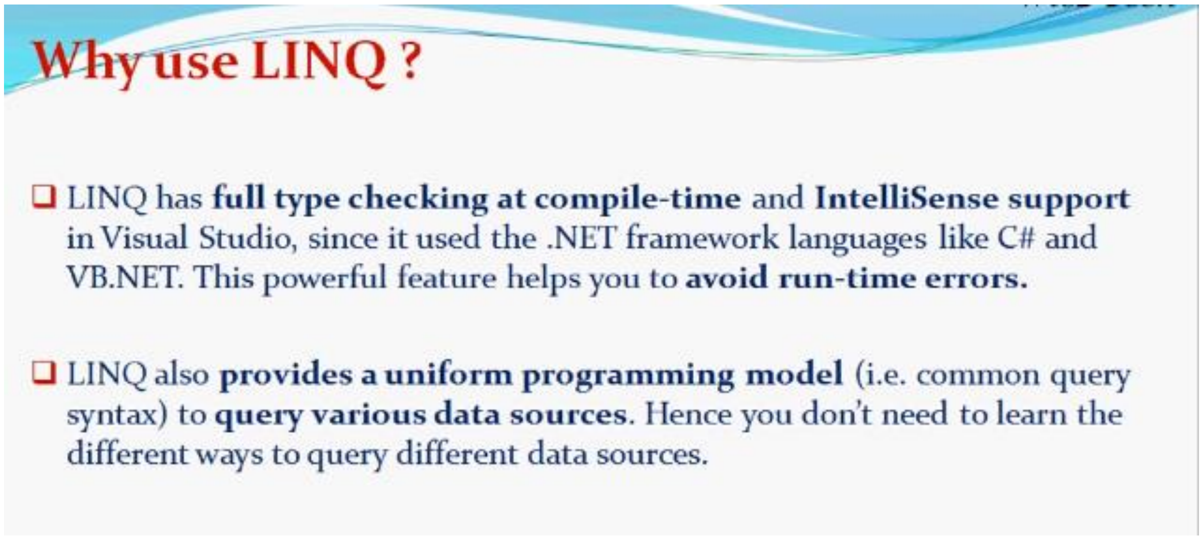


Explain LINQ in Detail?

- LINQ(Language Integrated Query) stands for "Language Integrated Query" and pronounced as "LINK".
- LINQ was introduced with .NET Framework 3.5 including Visual Studio 2008, C# 3.0 and VB.NET 2008 (VB 9.0).
- LINQ is **uniform query syntax in C# and VB.NET used to save and retrieve data from different sources** like an Object Collection, SQL server database, XML, web service etc
- LINQ can be **Witten in any .NET Supporting Language**.
- LINQ has its **own Query Processing Engine**.

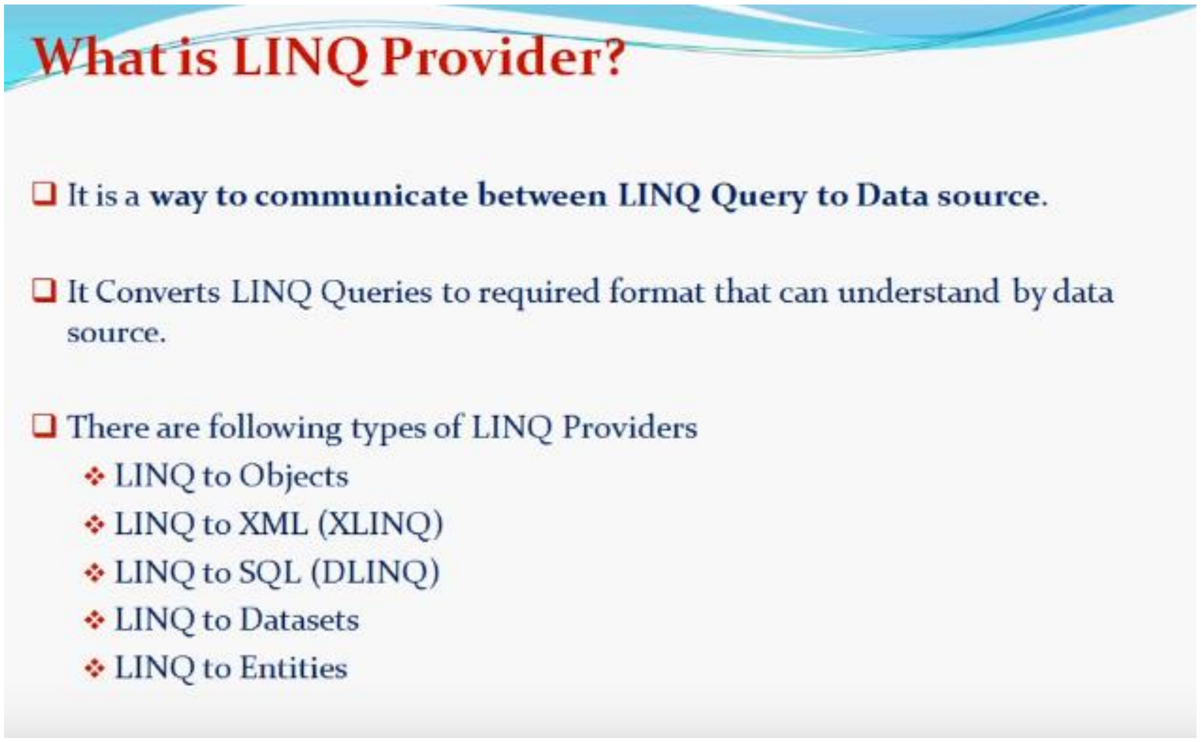


- Information returned by a **LINQ query is a collection of in-memory object which may be enumerated.**
- The LINQ concept treats the data source as an Object, rather than a Database. So we may say, its an object that is queried.
- LINQ may query any type of data source, like:
  - LINQ querying SQL (MS SQL Server supported).
  - LINQ querying Datasets (Querying is possible on Datasets and DataTables)
  - LINQ querying ORM Solution
  - LINQ querying Objects (In-memory data may be queried)
  - LINQ querying XML (Querying is possible on XML data source
- LINQ supports querying to those objects that implement the IEnumerable Interface.



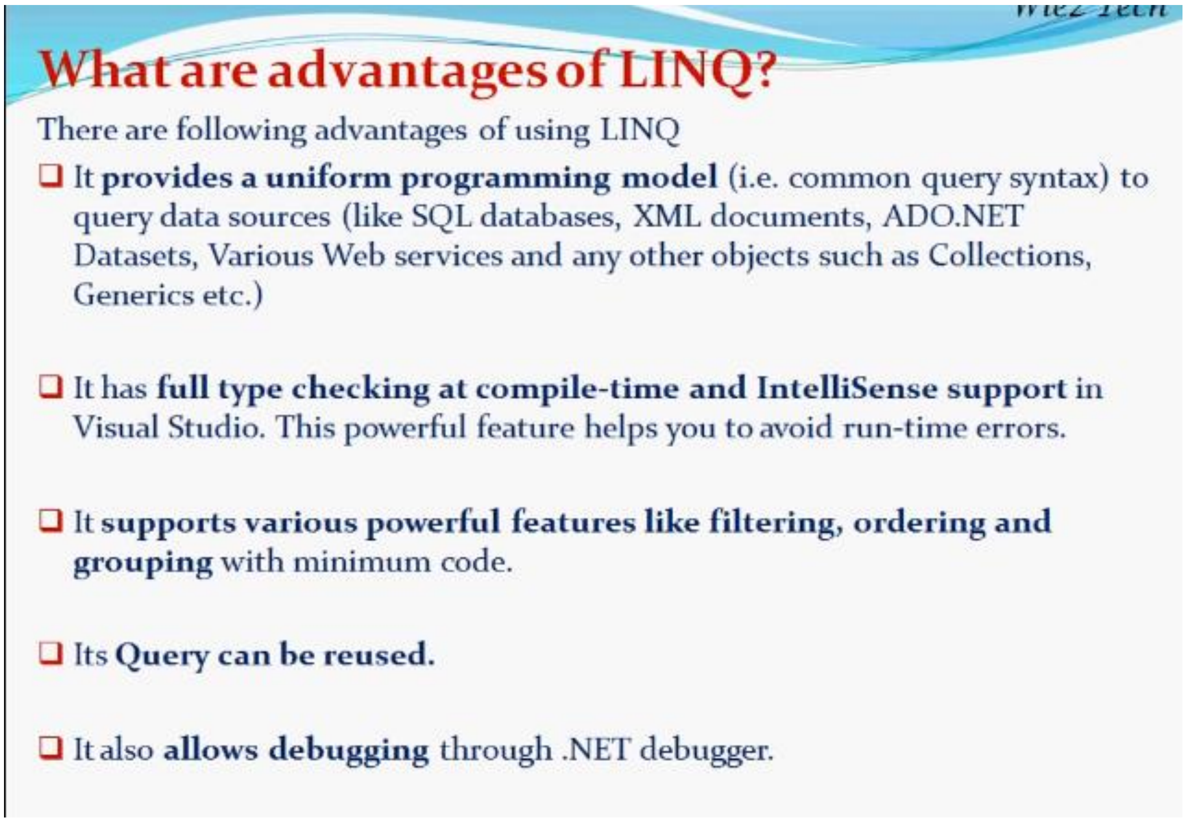
Why use LINQ ?

- ❑ LINQ has **full type checking at compile-time** and **IntelliSense support** in Visual Studio, since it used the .NET framework languages like C# and VB.NET. This powerful feature helps you to **avoid run-time errors.**
- ❑ LINQ also **provides a uniform programming model** (i.e. common query syntax) to **query various data sources**. Hence you don't need to learn the different ways to query different data sources.



What is LINQ Provider?

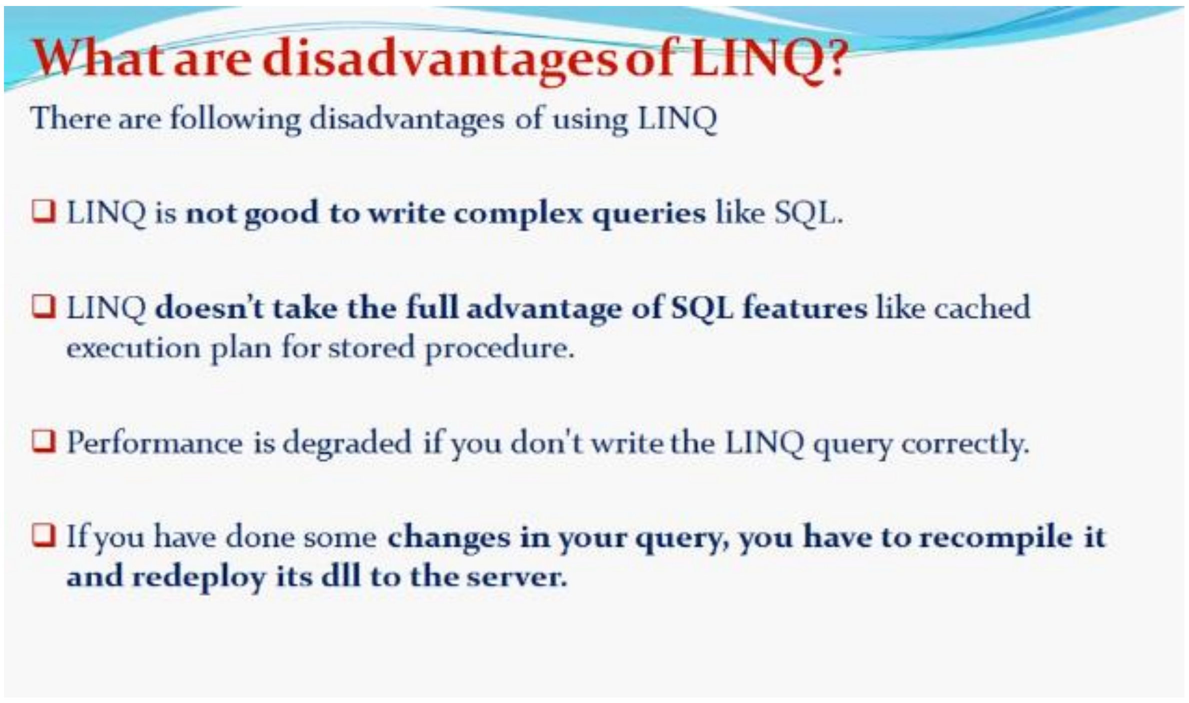
- ❑ It is a **way to communicate between LINQ Query to Data source**.
- ❑ It Converts LINQ Queries to required format that can understand by data source.
- ❑ There are following types of LINQ Providers
  - ❖ LINQ to Objects
  - ❖ LINQ to XML (XLINQ)
  - ❖ LINQ to SQL (DLINQ)
  - ❖ LINQ to Datasets
  - ❖ LINQ to Entities



What are advantages of LINQ?

There are following advantages of using LINQ

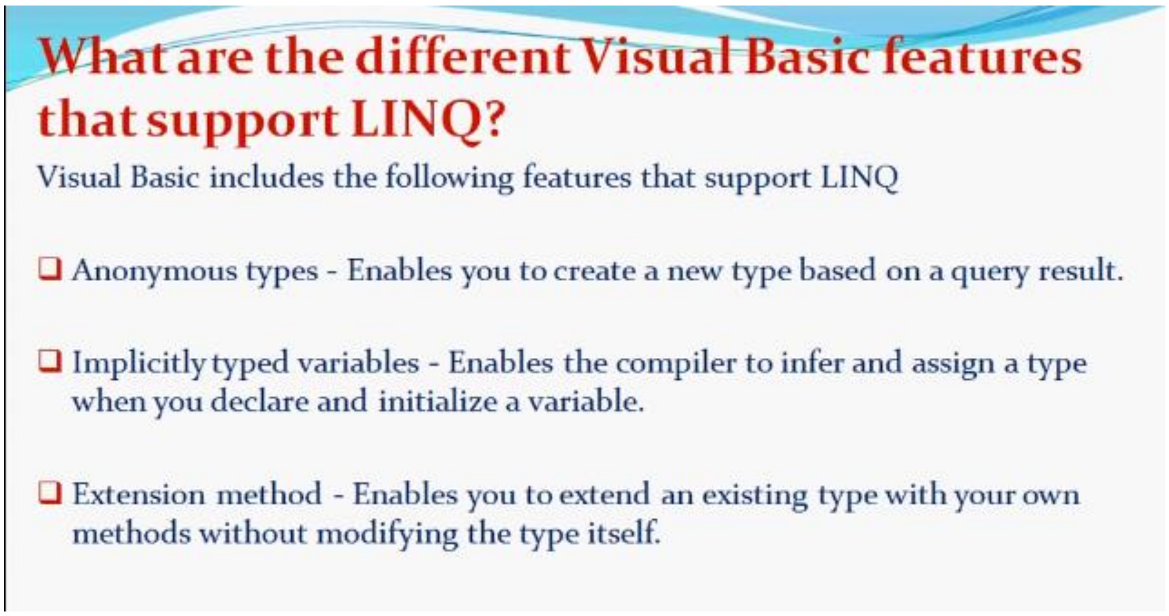
- It **provides a uniform programming model** (i.e. common query syntax) to query data sources (like SQL databases, XML documents, ADO.NET Datasets, Various Web services and any other objects such as Collections, Generics etc.)
- It has **full type checking at compile-time and IntelliSense support** in Visual Studio. This powerful feature helps you to avoid run-time errors.
- It **supports various powerful features like filtering, ordering and grouping** with minimum code.
- Its **Query can be reused.**
- It also **allows debugging** through .NET debugger.



What are disadvantages of LINQ?

There are following disadvantages of using LINQ

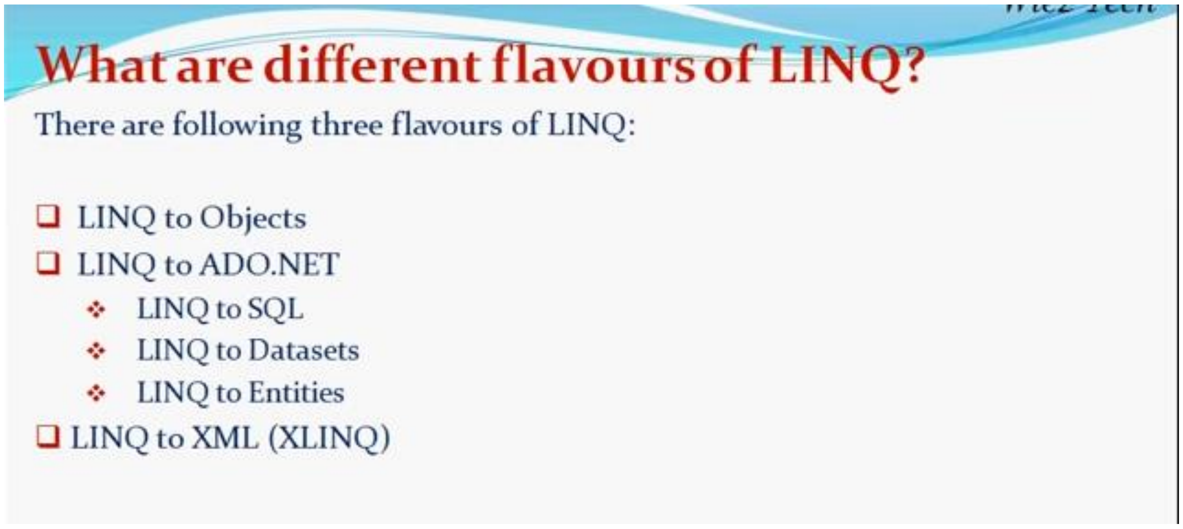
- LINQ is **not good to write complex queries** like SQL.
- LINQ **doesn't take the full advantage of SQL features** like cached execution plan for stored procedure.
- Performance is degraded if you don't write the LINQ query correctly.
- If you have done some **changes in your query, you have to recompile it and redeploy its dll to the server.**



What are the different Visual Basic features that support LINQ?

Visual Basic includes the following features that support LINQ

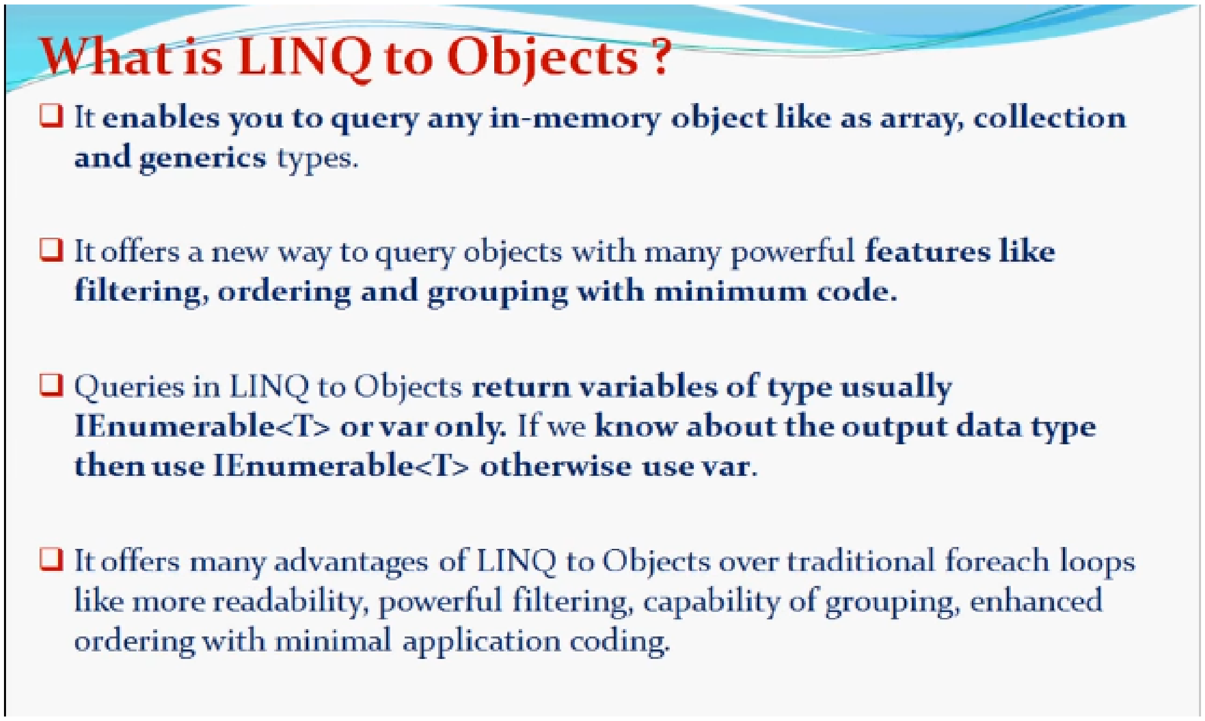
- ❑ Anonymous types - Enables you to create a new type based on a query result.
- ❑ Implicitly typed variables - Enables the compiler to infer and assign a type when you declare and initialize a variable.
- ❑ Extension method - Enables you to extend an existing type with your own methods without modifying the type itself.



What are different flavours of LINQ?

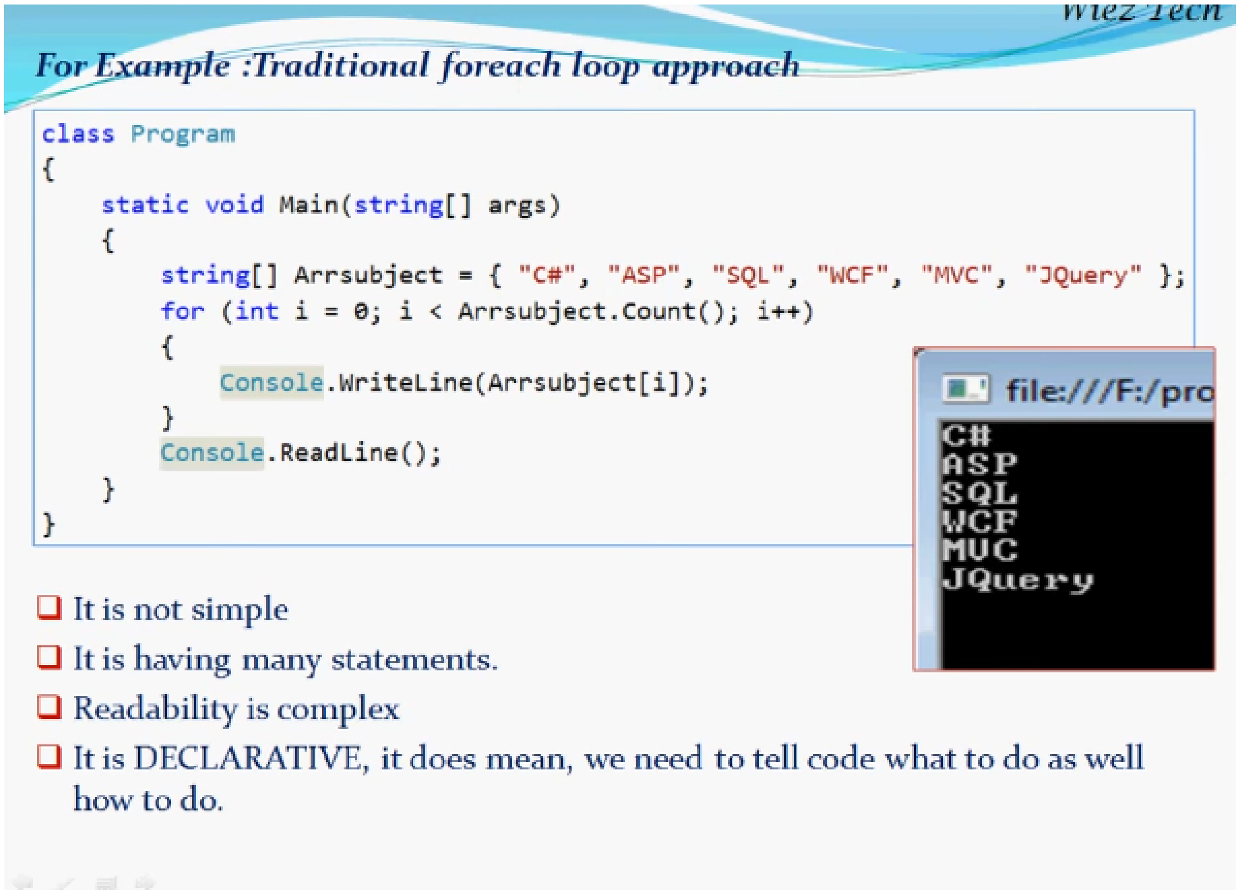
There are following three flavours of LINQ:

- ❑ LINQ to Objects
- ❑ LINQ to ADO.NET
  - ❖ LINQ to SQL
  - ❖ LINQ to Datasets
  - ❖ LINQ to Entities
- ❑ LINQ to XML (XLINQ)



What is LINQ to Objects ?

- It **enables you to query any in-memory object like as array, collection and generics** types.
- It offers a new way to query objects with many powerful **features like filtering, ordering and grouping with minimum code.**
- Queries in LINQ to Objects **return variables of type usually IEnumerable<T> or var only.** If we **know about the output data type then use IEnumerable<T> otherwise use var**.
- It offers many advantages of LINQ to Objects over traditional foreach loops like more readability, powerful filtering, capability of grouping, enhanced ordering with minimal application coding.



Wiez Tech

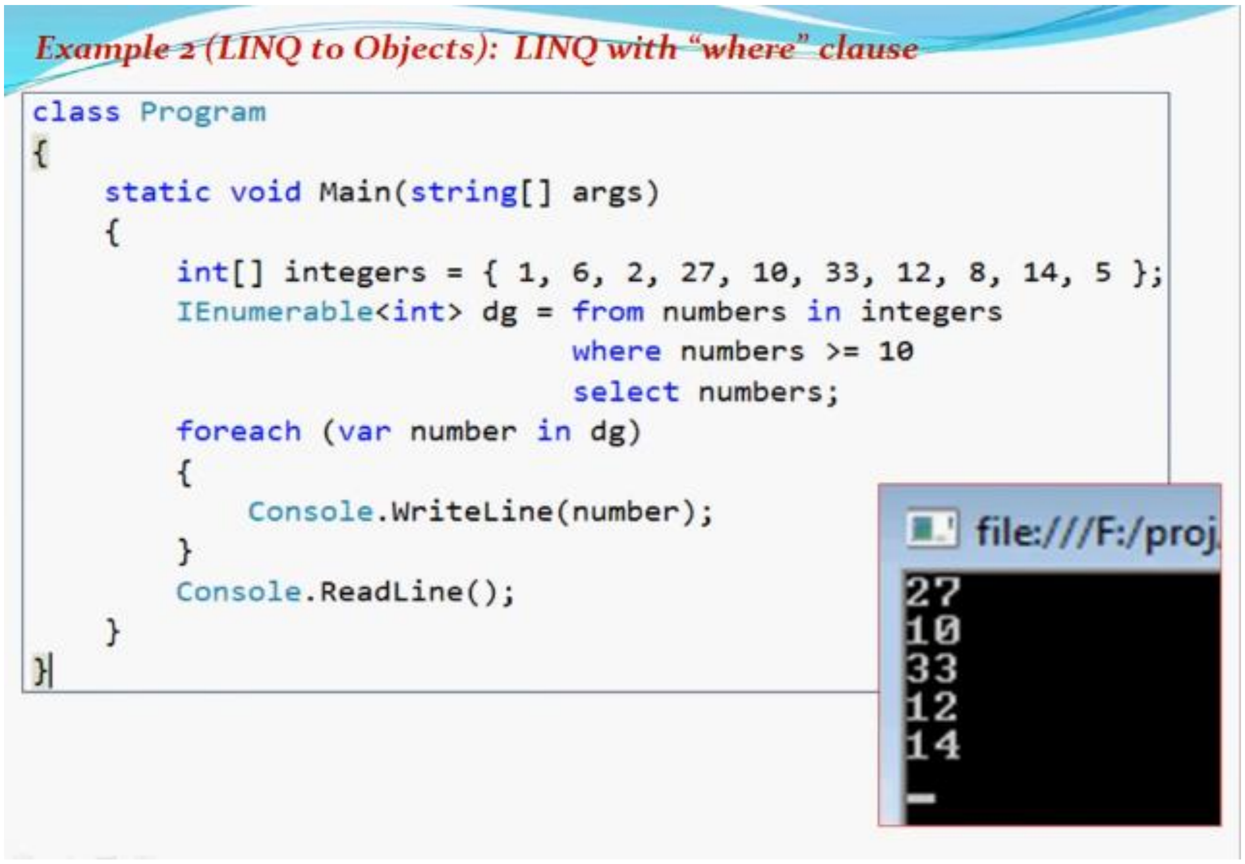
For Example :Traditional foreach loop approach

```
class Program
{
    static void Main(string[] args)
    {
        string[] Arrsubject = { "C#", "ASP", "SQL", "WCF", "MVC", "JQuery" };
        for (int i = 0; i < Arrsubject.Count(); i++)
        {
            Console.WriteLine(Arrsubject[i]);
        }
        Console.ReadLine();
    }
}
```

file:///F:/pro

```
C#
ASP
SQL
WCF
MUC
JQuery
```

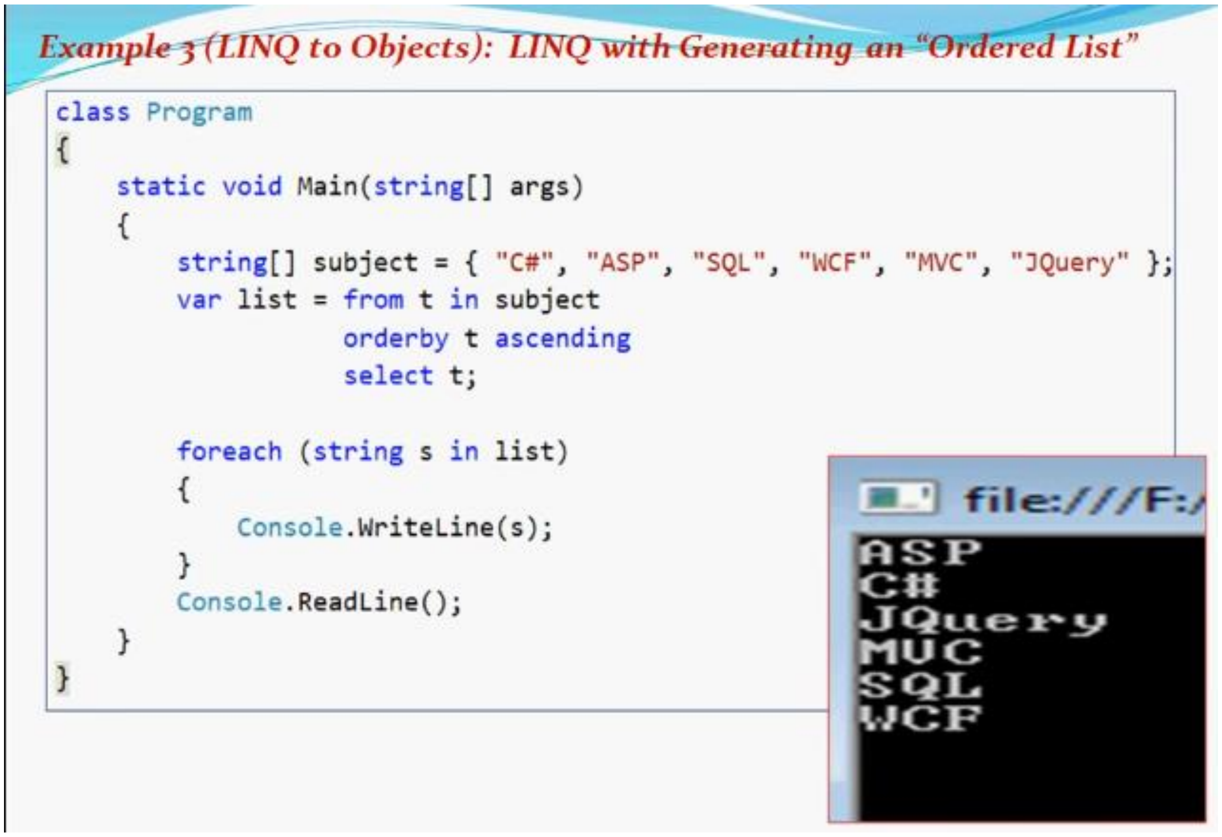
- It is not simple
- It is having many statements.
- Readability is complex
- It is DECLARATIVE, it does mean, we need to tell code what to do as well how to do.



Example 2 (LINQ to Objects): LINQ with "where" clause

```
class Program
{
    static void Main(string[] args)
    {
        int[] integers = { 1, 6, 2, 27, 10, 33, 12, 8, 14, 5 };
        IEnumerable<int> dg = from numbers in integers
                              where numbers >= 10
                              select numbers;
        foreach (var number in dg)
        {
            Console.WriteLine(number);
        }
        Console.ReadLine();
    }
}
```

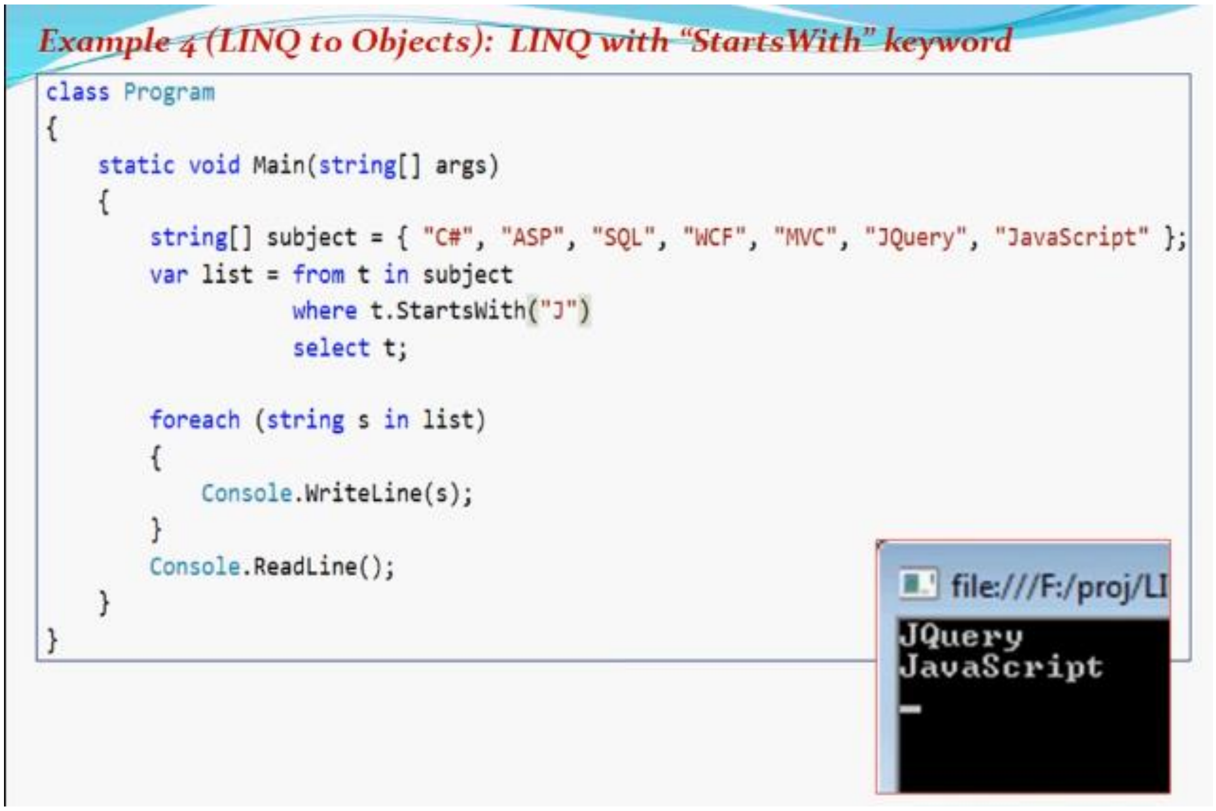
```
27
10
33
12
14
_
```



Example 3 (LINQ to Objects): LINQ with Generating an "Ordered List"

```
class Program
{
    static void Main(string[] args)
    {
        string[] subject = { "C#", "ASP", "SQL", "WCF", "MVC", "JQuery" };
        var list = from t in subject
                   orderby t ascending
                   select t;

        foreach (string s in list)
        {
            Console.WriteLine(s);
        }
        Console.ReadLine();
    }
}
```

```
ASP
C#
JQuery
MVC
SQL
WCF
```

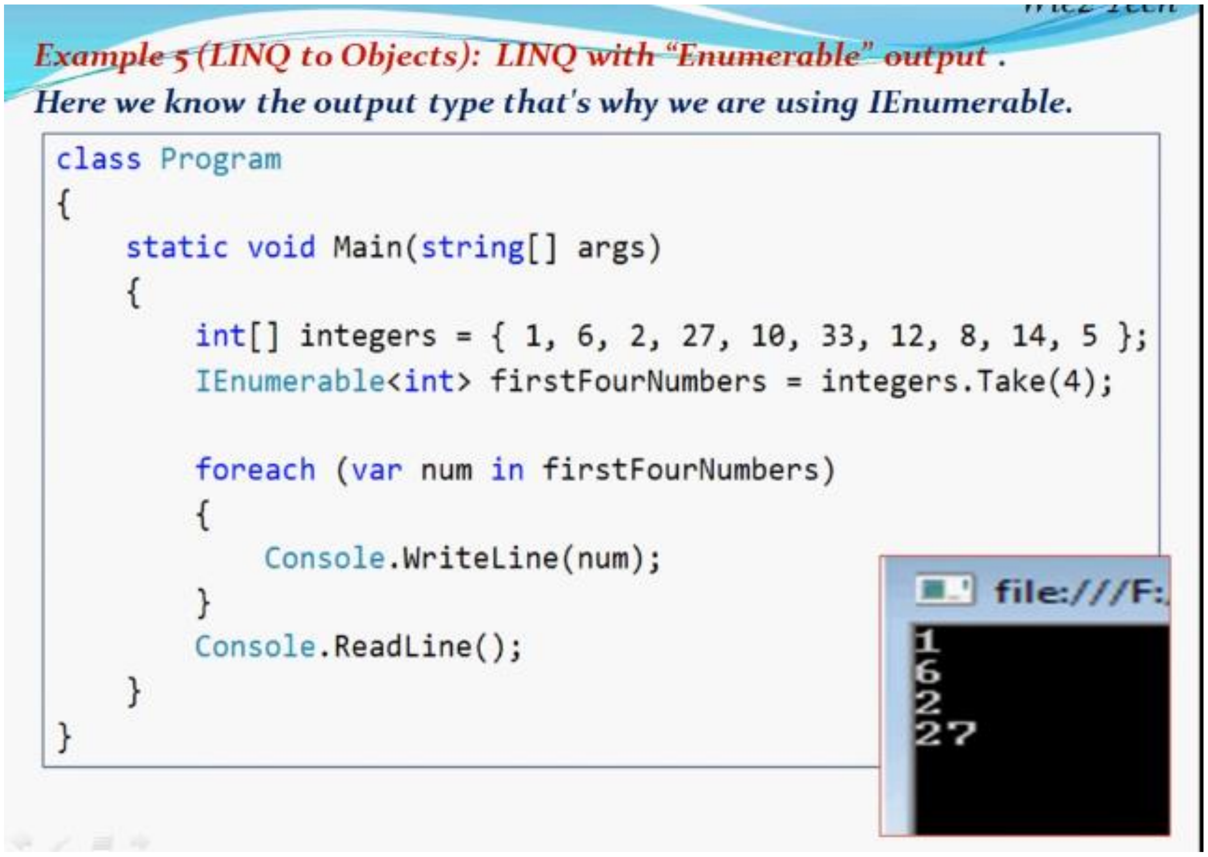


Example 4 (LINQ to Objects): LINQ with "StartsWith" keyword

```
class Program
{
    static void Main(string[] args)
    {
        string[] subject = { "C#", "ASP", "SQL", "WCF", "MVC", "JQuery", "JavaScript" };
        var list = from t in subject
                   where t.StartsWith("J")
                   select t;

        foreach (string s in list)
        {
            Console.WriteLine(s);
        }
        Console.ReadLine();
    }
}
```

file:///F:/proj/LI

```
JQuery
JavaScript
_
```



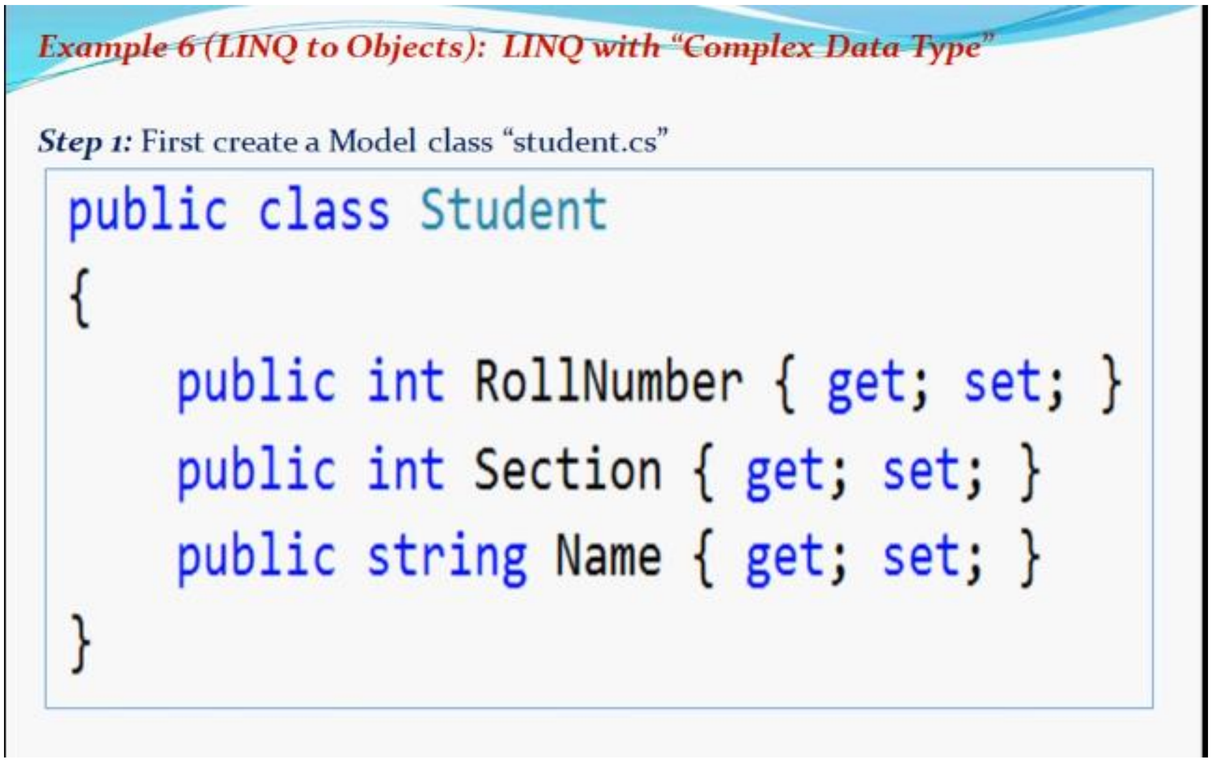
Example 5 (LINQ to Objects): LINQ with "Enumerable" output .

Here we know the output type that's why we are using IEnumerable.

```
class Program
{
    static void Main(string[] args)
    {
        int[] integers = { 1, 6, 2, 27, 10, 33, 12, 8, 14, 5 };
        IEnumerable<int> firstFourNumbers = integers.Take(4);

        foreach (var num in firstFourNumbers)
        {
            Console.WriteLine(num);
        }
        Console.ReadLine();
    }
}
```

file:///F:

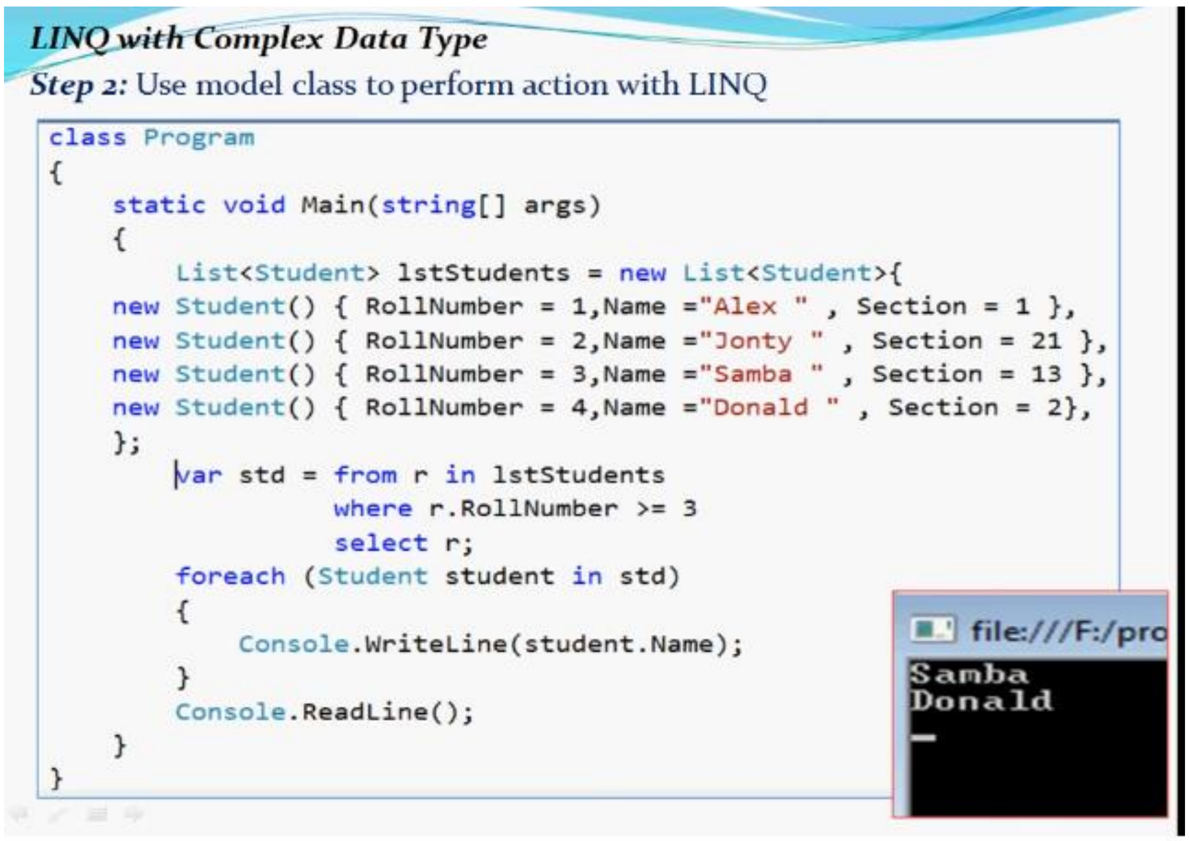
```
1
6
2
27
```



Example 6 (LINQ to Objects): LINQ with "Complex Data Type"

Step 1: First create a Model class "student.cs"

```
public class Student
{
    public int RollNumber { get; set; }
    public int Section { get; set; }
    public string Name { get; set; }
}
```



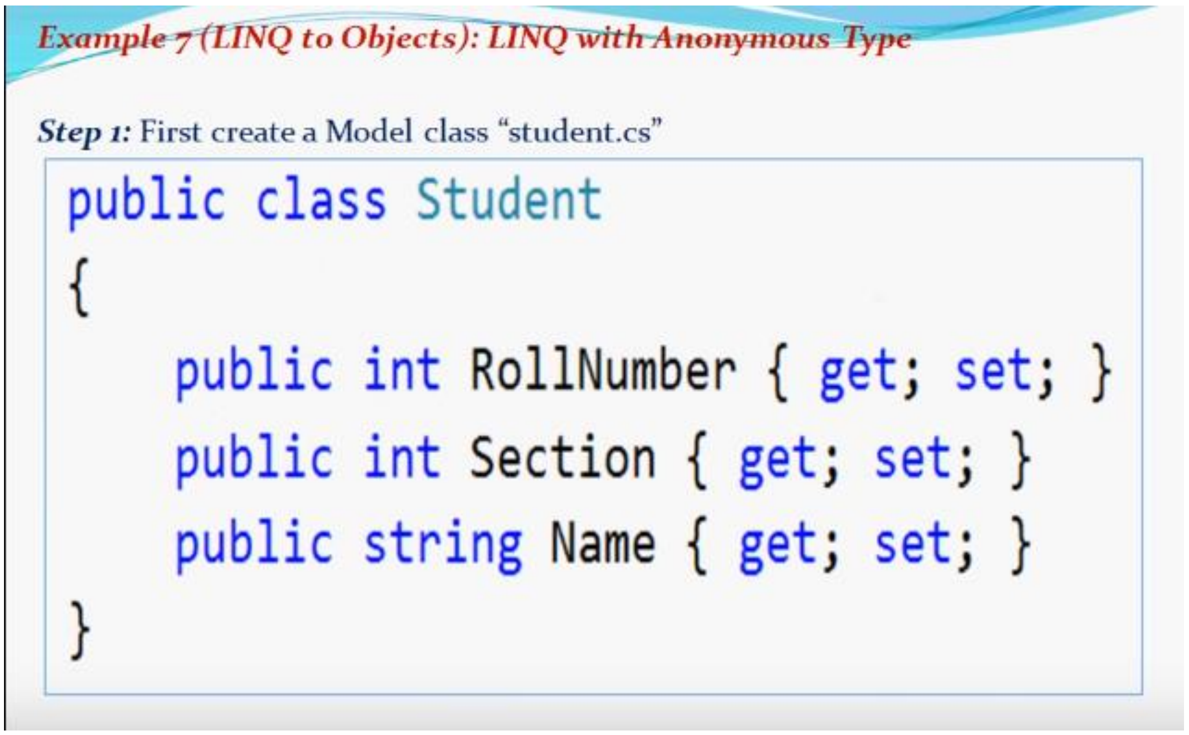
LINQ with Complex Data Type

Step 2: Use model class to perform action with LINQ

```
class Program
{
    static void Main(string[] args)
    {
        List<Student> lstStudents = new List<Student>{
    new Student() { RollNumber = 1,Name ="Alex " , Section = 1 },
    new Student() { RollNumber = 2,Name ="Jonty " , Section = 21 },
    new Student() { RollNumber = 3,Name ="Samba " , Section = 13 },
    new Student() { RollNumber = 4,Name ="Donald " , Section = 2},
    };
        var std = from r in lstStudents
                  where r.RollNumber >= 3
                  select r;
        foreach (Student student in std)
        {
            Console.WriteLine(student.Name);
        }
        Console.ReadLine();
    }
}
```

file:///F:/pro

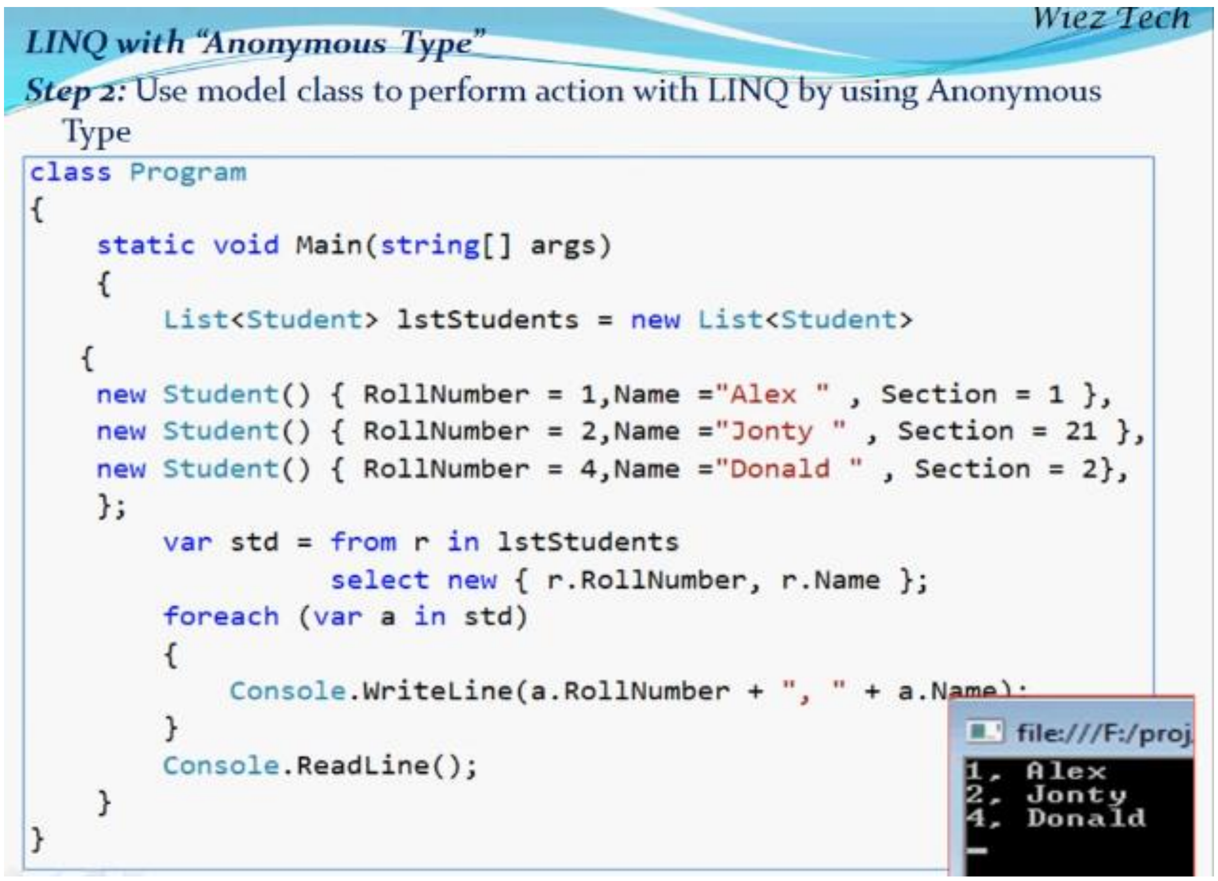
```
Samba
Donald
_
```



Example 7 (LINQ to Objects): LINQ with Anonymous Type

Step 1: First create a Model class “student.cs”

```
public class Student
{
    public int RollNumber { get; set; }
    public int Section { get; set; }
    public string Name { get; set; }
}
```



Wiez Tech

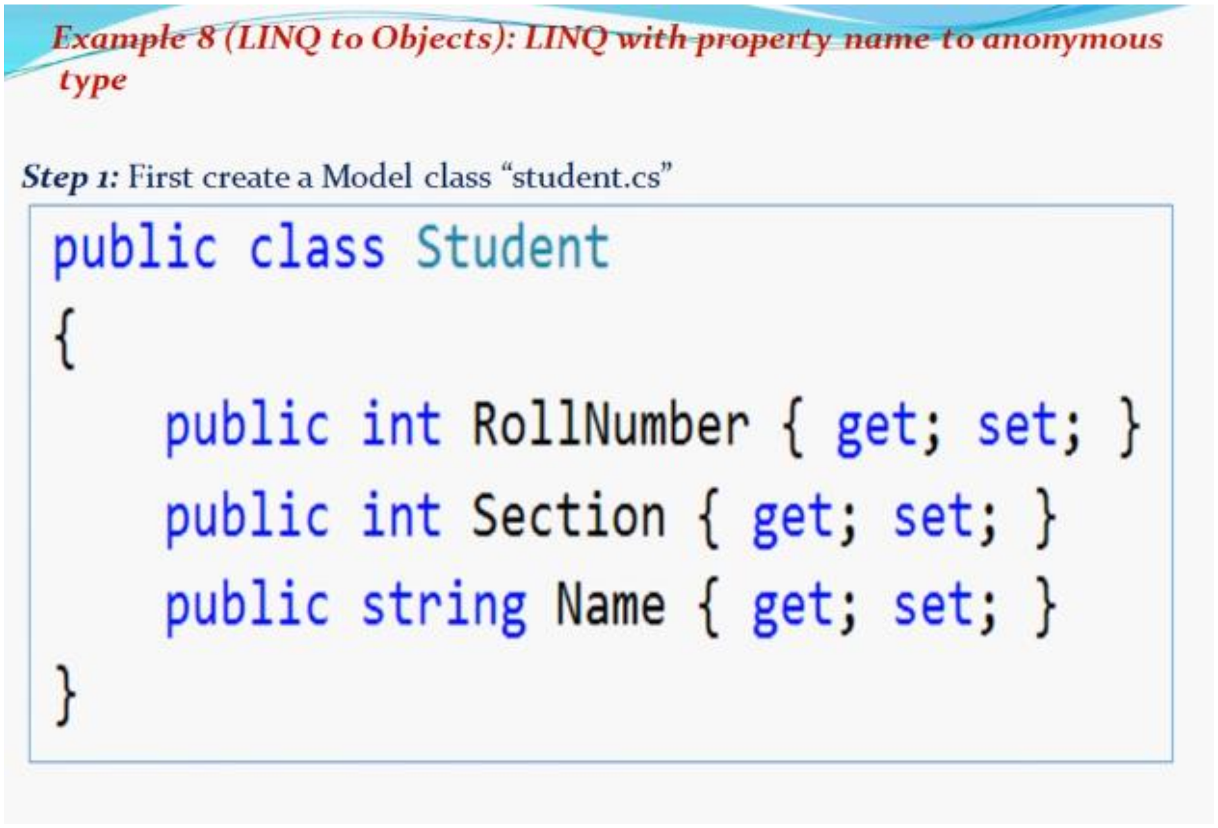
LINQ with “Anonymous Type”

Step 2: Use model class to perform action with LINQ by using Anonymous Type

```
class Program
{
    static void Main(string[] args)
    {
        List<Student> lstStudents = new List<Student>
   {
    new Student() { RollNumber = 1,Name ="Alex " , Section = 1 },
    new Student() { RollNumber = 2,Name ="Jonty " , Section = 21 },
    new Student() { RollNumber = 4,Name ="Donald " , Section = 2},
    };
        var std = from r in lstStudents
                  select new { r.RollNumber, r.Name };
        foreach (var a in std)
        {
            Console.WriteLine(a.RollNumber + ", " + a.Name);
        }
        Console.ReadLine();
    }
}
```

file:///F:/proj

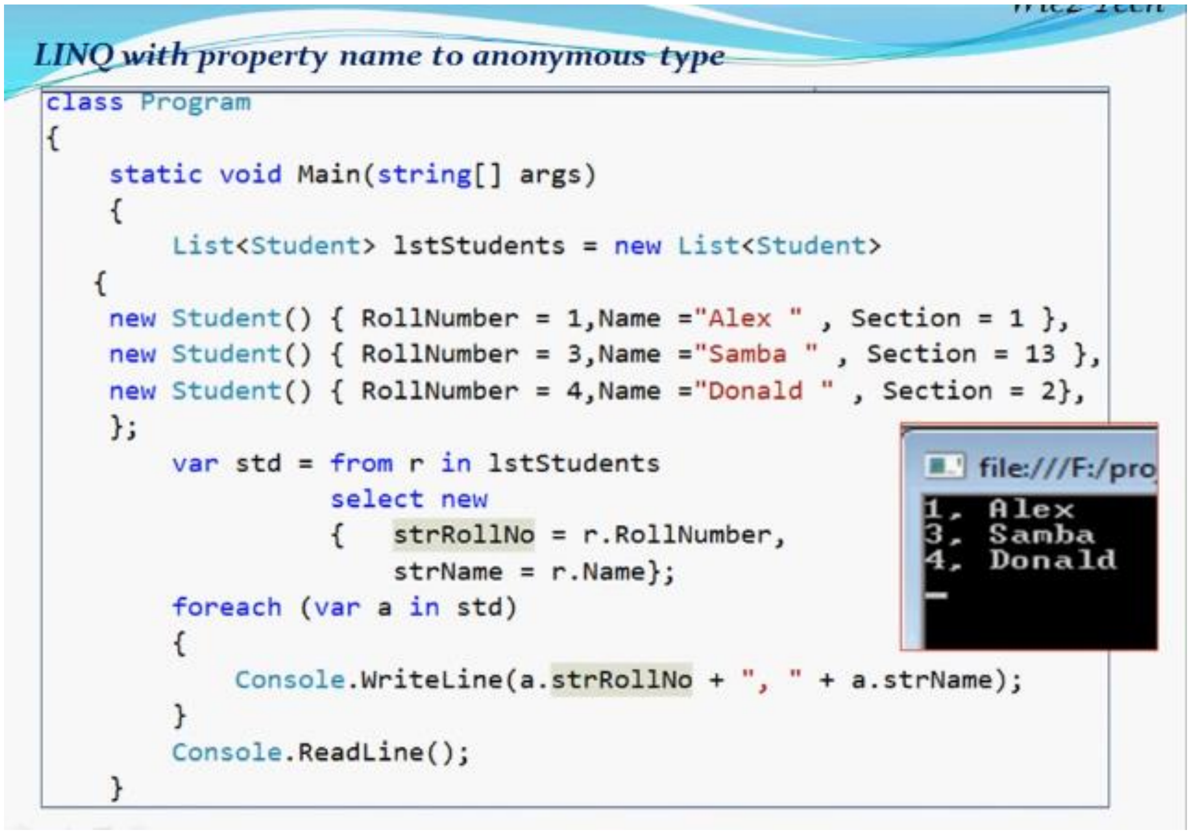
```
1, Alex
2, Jonty
4, Donald
_
```



Example 8 (LINQ to Objects): LINQ with property name to anonymous type

Step 1: First create a Model class “student.cs”

```
public class Student
{
    public int RollNumber { get; set; }
    public int Section { get; set; }
    public string Name { get; set; }
}
```

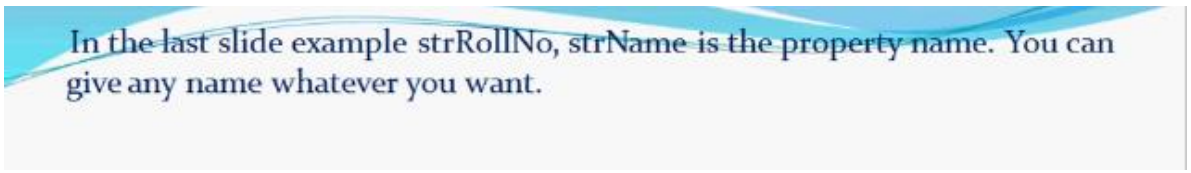


LINQ with property name to anonymous type

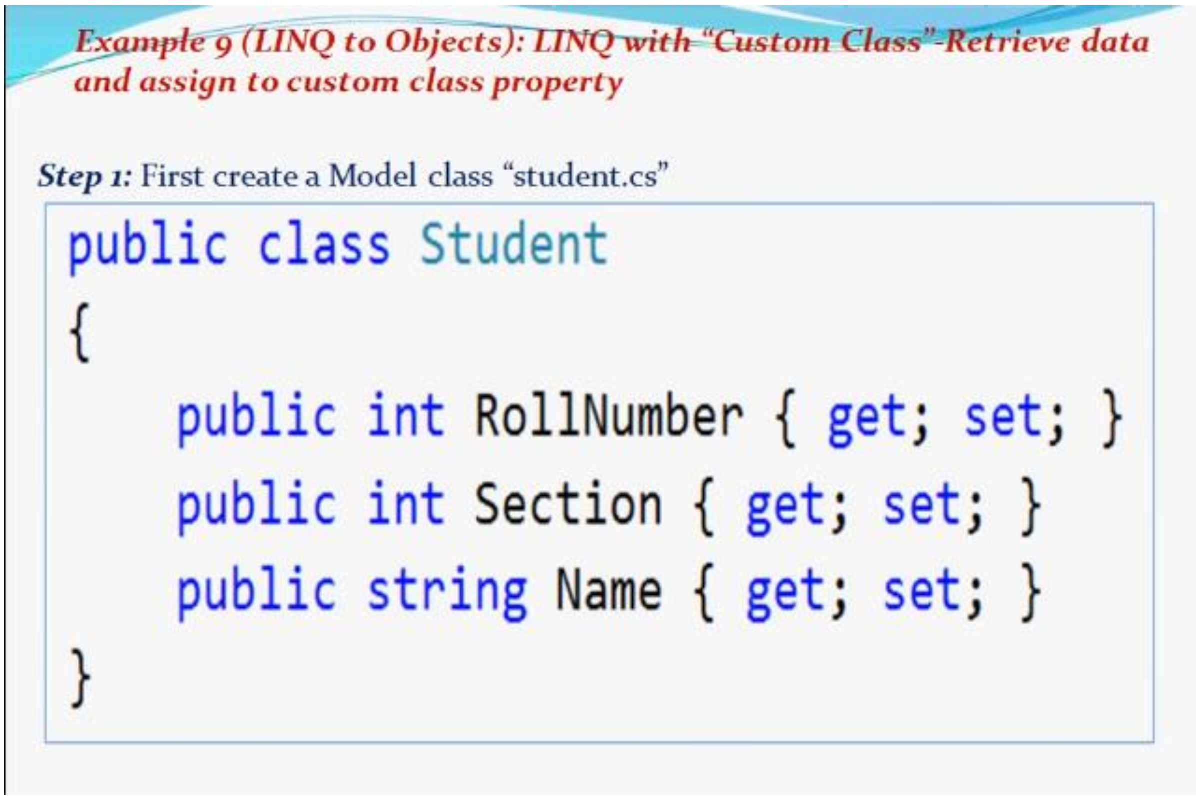
```
class Program
{
    static void Main(string[] args)
    {
        List<Student> lstStudents = new List<Student>
   {
    new Student() { RollNumber = 1,Name ="Alex " , Section = 1 },
    new Student() { RollNumber = 3,Name ="Samba " , Section = 13 },
    new Student() { RollNumber = 4,Name ="Donald " , Section = 2},
    };
        var std = from r in lstStudents
                  select new
                  {   strRollNo = r.RollNumber,
                      strName = r.Name};
        foreach (var a in std)
        {
            Console.WriteLine(a.strRollNo + ", " + a.strName);
        }
        Console.ReadLine();
    }
```

file:///F:/pro

```
1, Alex
3, Samba
4, Donald
_
```



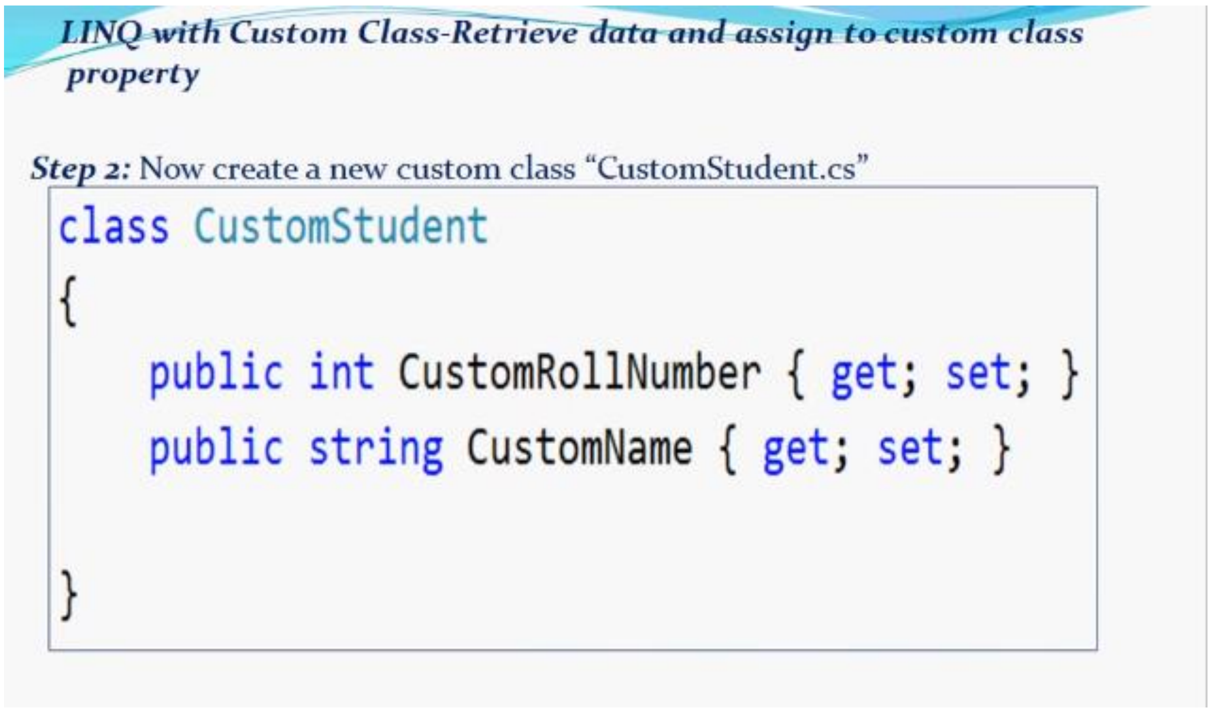
In the last slide example strRollNo, strName is the property name. You can give any name whatever you want.



Example 9 (LINQ to Objects): LINQ with "Custom Class"-Retrieve data and assign to custom class property

Step 1: First create a Model class "student.cs"

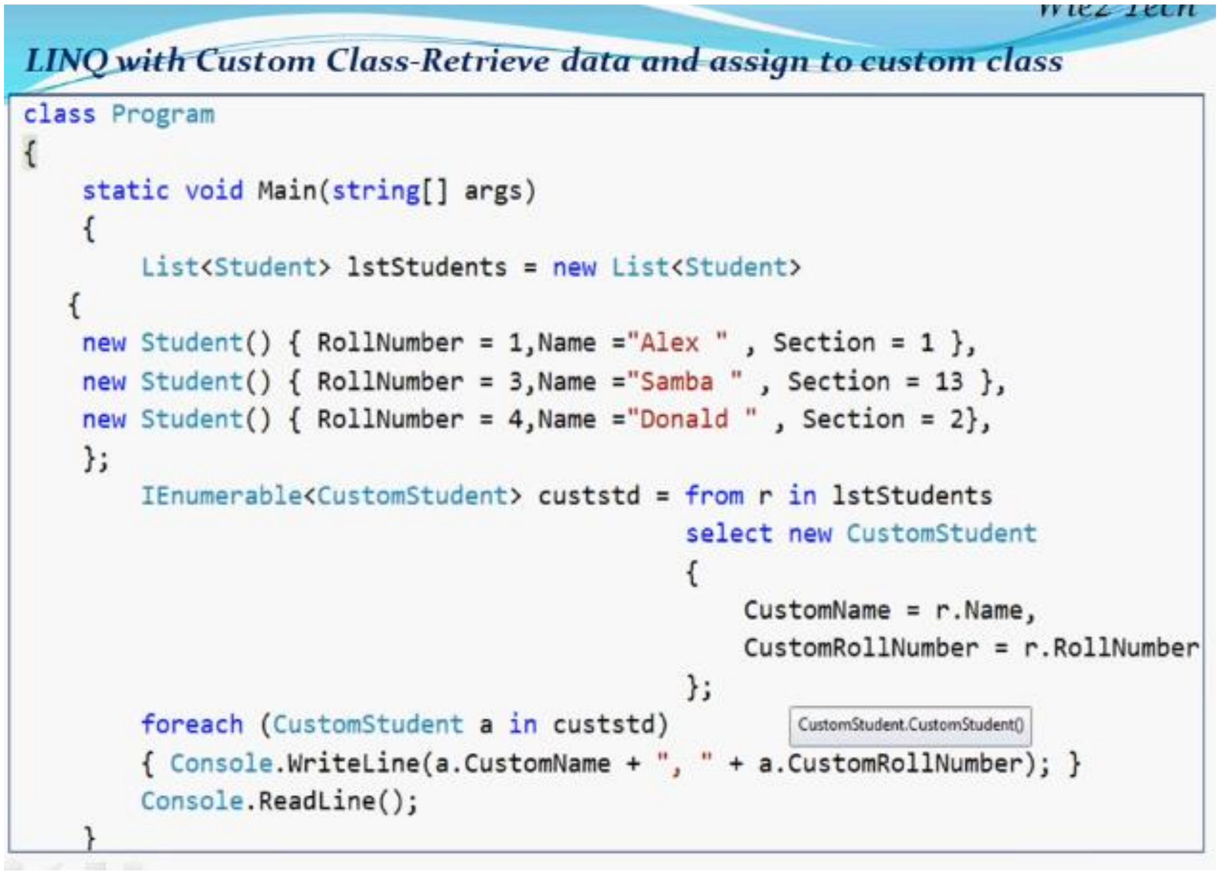
```
public class Student
{
    public int RollNumber { get; set; }
    public int Section { get; set; }
    public string Name { get; set; }
}
```



LINQ with Custom Class-Retrieve data and assign to custom class property

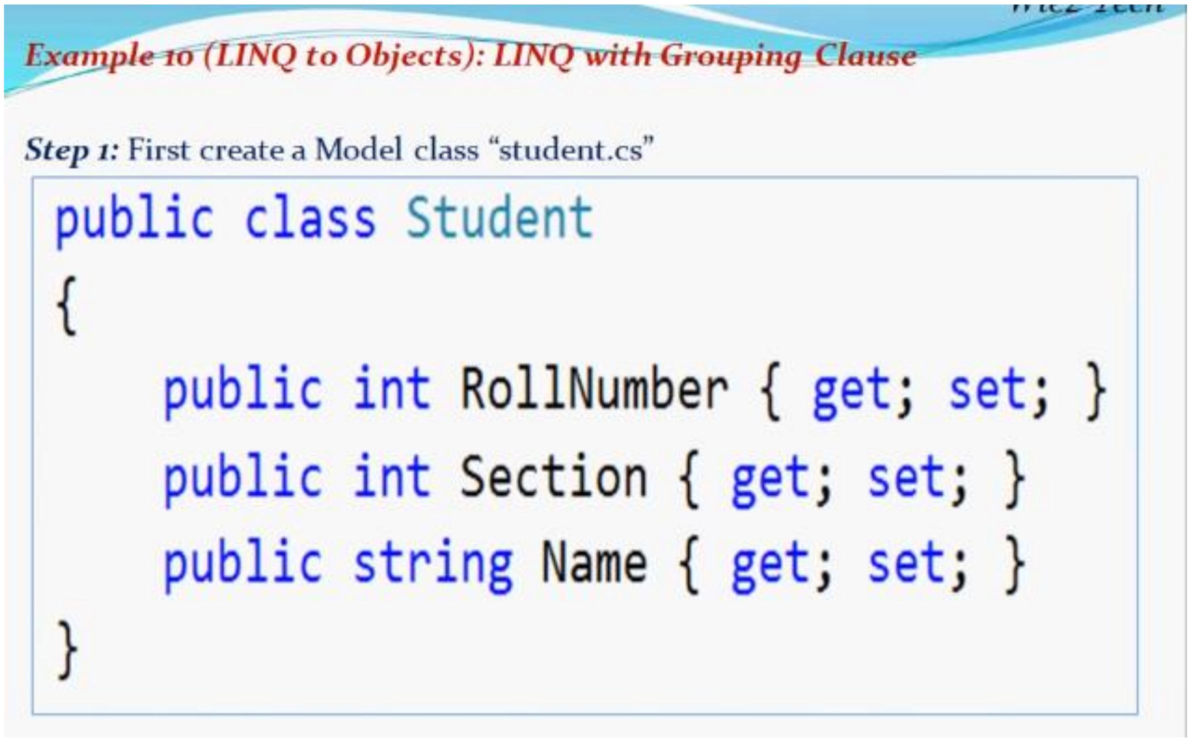
Step 2: Now create a new custom class "CustomStudent.cs"

```
class CustomStudent
{
    public int CustomRollNumber { get; set; }
    public string CustomName { get; set; }


}
```



LINQ with Custom Class-Retrieve data and assign to custom class

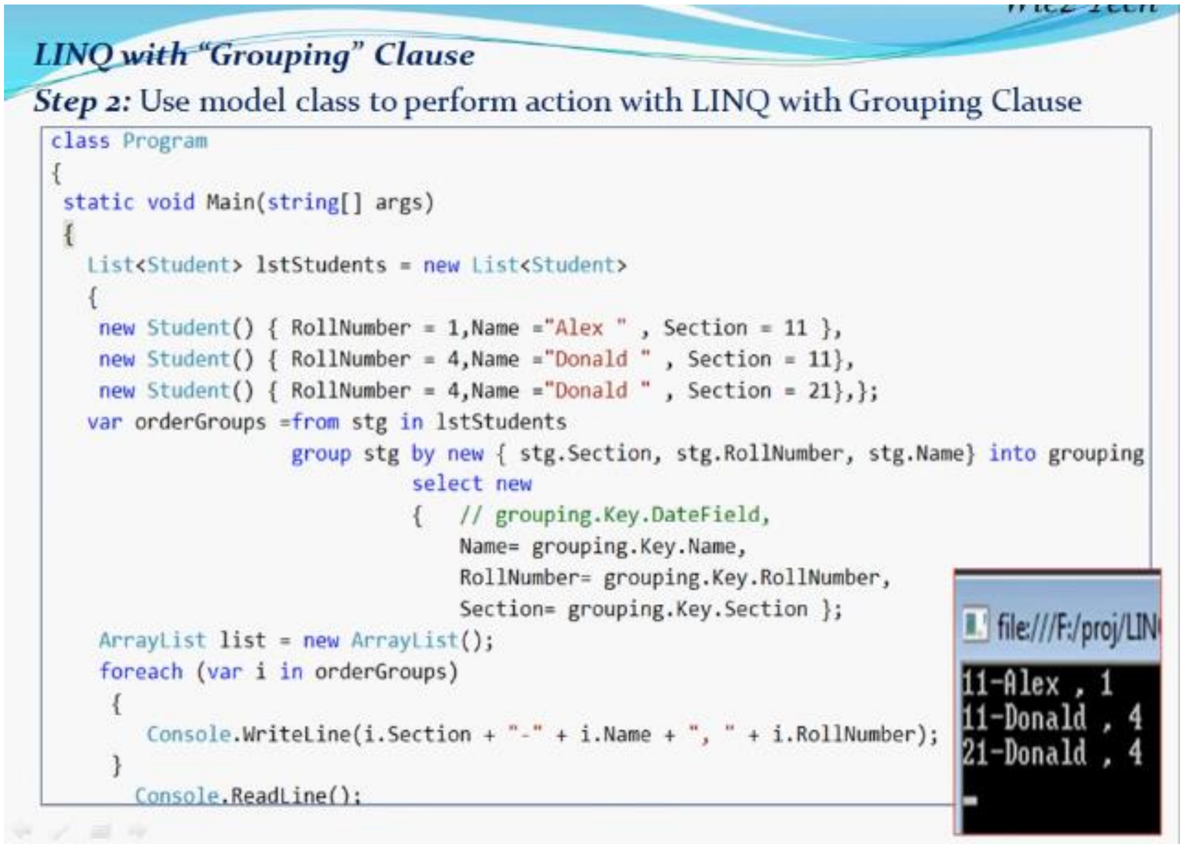
```
class Program
{
    static void Main(string[] args)
    {
        List<Student> lstStudents = new List<Student>
   {
    new Student() { RollNumber = 1,Name ="Alex " , Section = 1 },
    new Student() { RollNumber = 3,Name ="Samba " , Section = 13 },
    new Student() { RollNumber = 4,Name ="Donald " , Section = 2},
    };
        IEnumerable<CustomStudent> custstd = from r in lstStudents
                                             select new CustomStudent
                                             {
                                                 CustomName = r.Name,
                                                 CustomRollNumber = r.RollNumber
                                             };
        foreach (CustomStudent a in custstd)          CustomStudent.CustomStudent()
        { Console.WriteLine(a.CustomName + ", " + a.CustomRollNumber); }
        Console.ReadLine();
    }
```



Example 10 (LINQ to Objects): LINQ with Grouping Clause

Step 1: First create a Model class "student.cs"

```
public class Student
{
    public int RollNumber { get; set; }
    public int Section { get; set; }
    public string Name { get; set; }
}
```



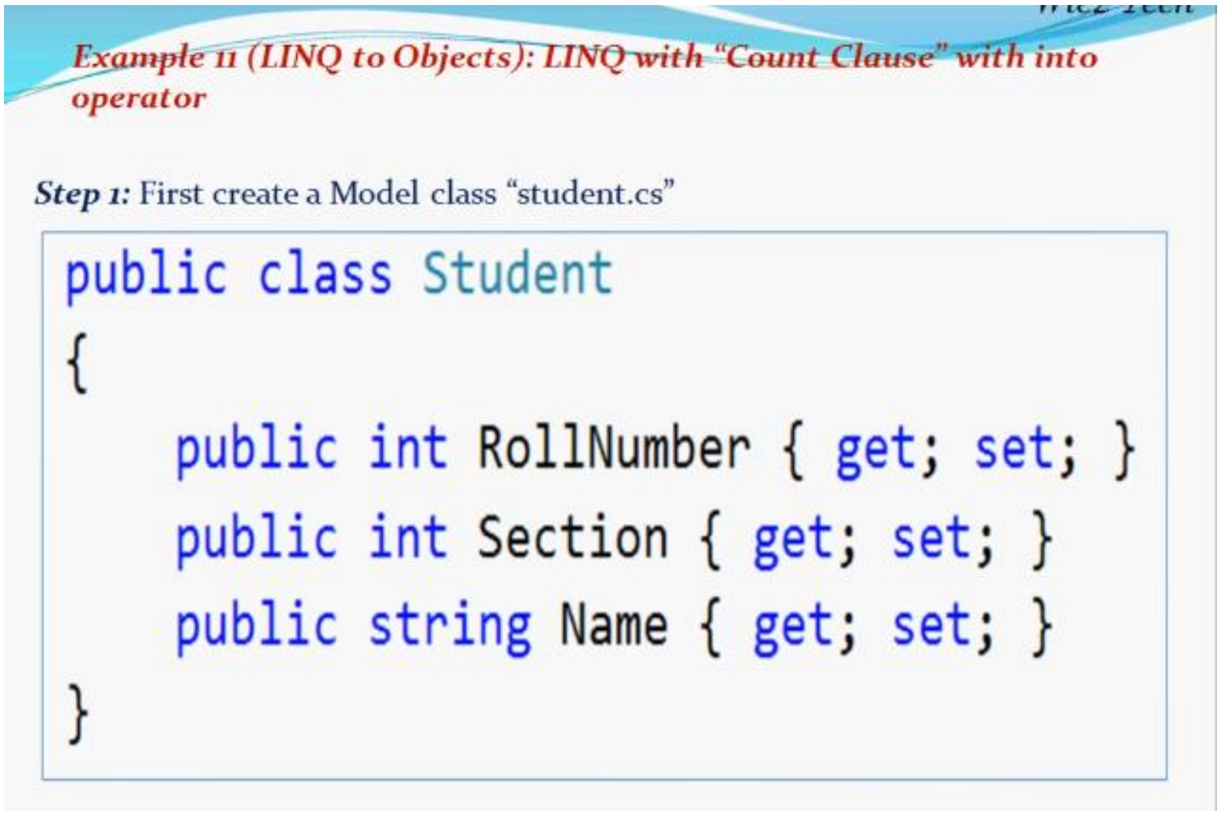
LINQ with "Grouping" Clause

Step 2: Use model class to perform action with LINQ with Grouping Clause

```
class Program
{
 static void Main(string[] args)
 {
   List<Student> lstStudents = new List<Student>
   {
    new Student() { RollNumber = 1,Name ="Alex " , Section = 11 },
    new Student() { RollNumber = 4,Name ="Donald " , Section = 11},
    new Student() { RollNumber = 4,Name ="Donald " , Section = 21},};
   var orderGroups =from stg in lstStudents
                    group stg by new { stg.Section, stg.RollNumber, stg.Name} into grouping
                              select new
                              {   // grouping.Key.DateField,
                                  Name= grouping.Key.Name,
                                  RollNumber= grouping.Key.RollNumber,
                                  Section= grouping.Key.Section };
    ArrayList list = new ArrayList();
    foreach (var i in orderGroups)
     {
        Console.WriteLine(i.Section + "-" + i.Name + ", " + i.RollNumber);
     }
      Console.ReadLine();
```

file:///F:/proj/LIN

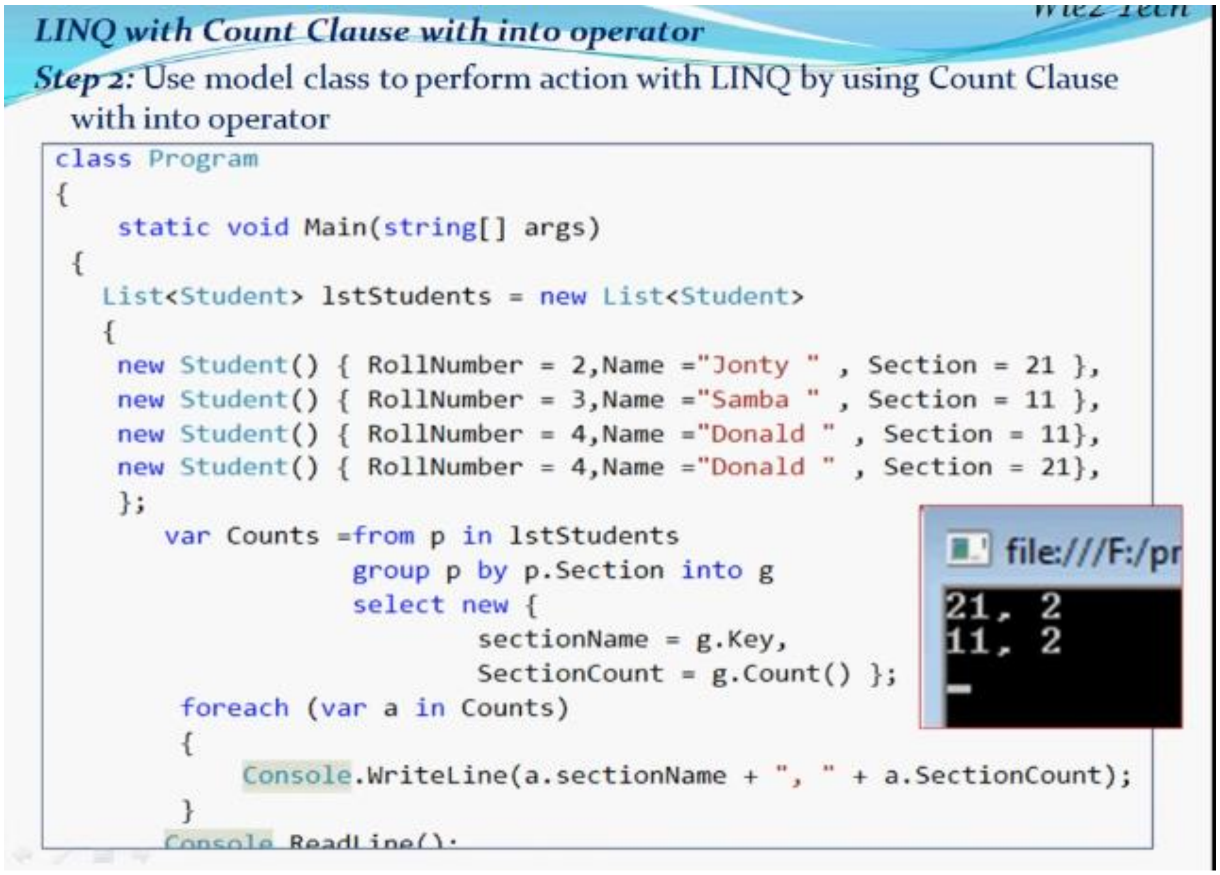
```
11-Alex , 1
11-Donald , 4
21-Donald , 4
```



Example 11 (LINQ to Objects): LINQ with "Count Clause" with into operator

Step 1: First create a Model class "student.cs"

```
public class Student
{
    public int RollNumber { get; set; }
    public int Section { get; set; }
    public string Name { get; set; }
}
```



Wiez Tech

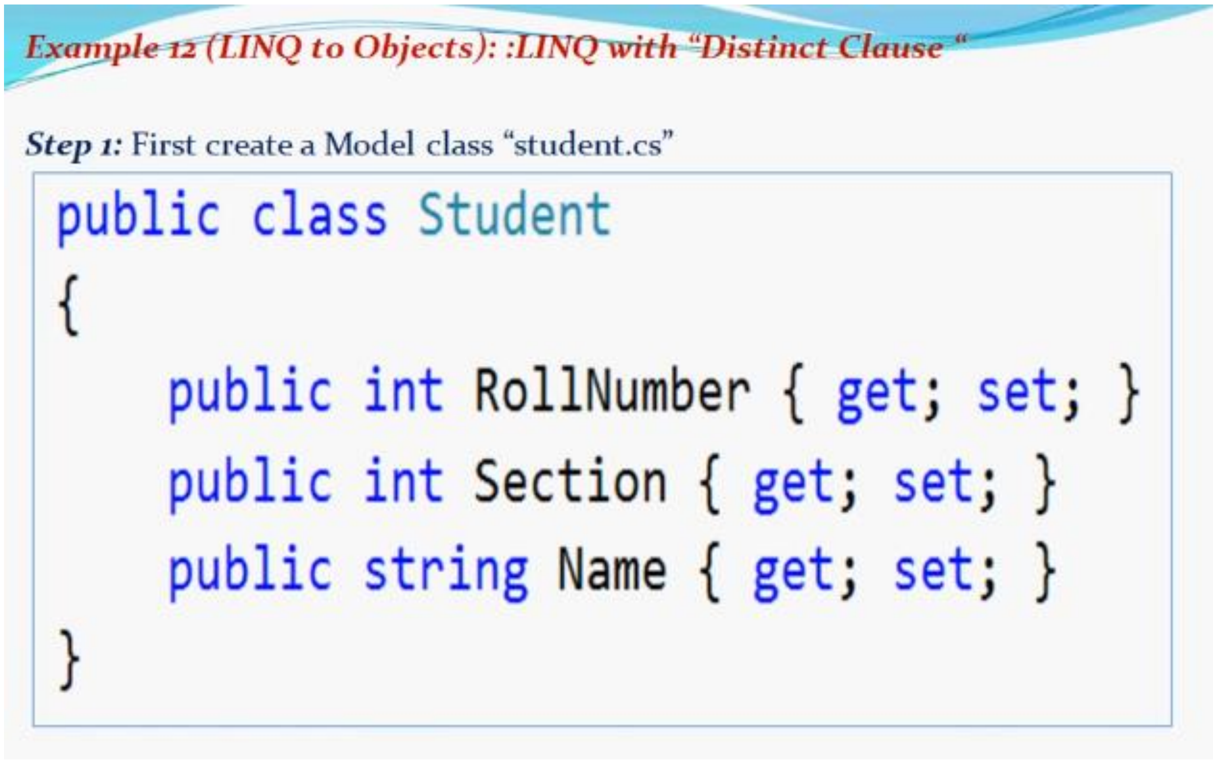
LINQ with Count Clause with into operator

Step 2: Use model class to perform action with LINQ by using Count Clause with into operator

```
class Program
{
    static void Main(string[] args)
 {
   List<Student> lstStudents = new List<Student>
   {
    new Student() { RollNumber = 2,Name ="Jonty " , Section = 21 },
    new Student() { RollNumber = 3,Name ="Samba " , Section = 11 },
    new Student() { RollNumber = 4,Name ="Donald " , Section = 11},
    new Student() { RollNumber = 4,Name ="Donald " , Section = 21},
    };
       var Counts =from p in lstStudents
                   group p by p.Section into g
                   select new {
                           sectionName = g.Key,
                           SectionCount = g.Count() };
        foreach (var a in Counts)
        {
            Console.WriteLine(a.sectionName + ", " + a.SectionCount);
        }
       Console.ReadLine();
```

file:///F:/pr

```
21, 2
11, 2
_
```



Example 12 (LINQ to Objects): :LINQ with "Distinct Clause "

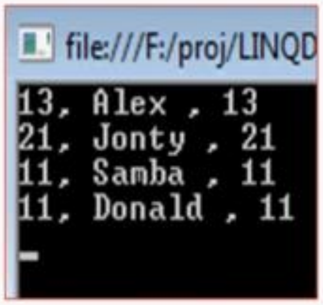
Step 1: First create a Model class "student.cs"

```
public class Student
{
    public int RollNumber { get; set; }
    public int Section { get; set; }
    public string Name { get; set; }
}
```

LINQ with Distinct Clause

Step 2: Use model class to perform action with LINQ by using Distinct Clause

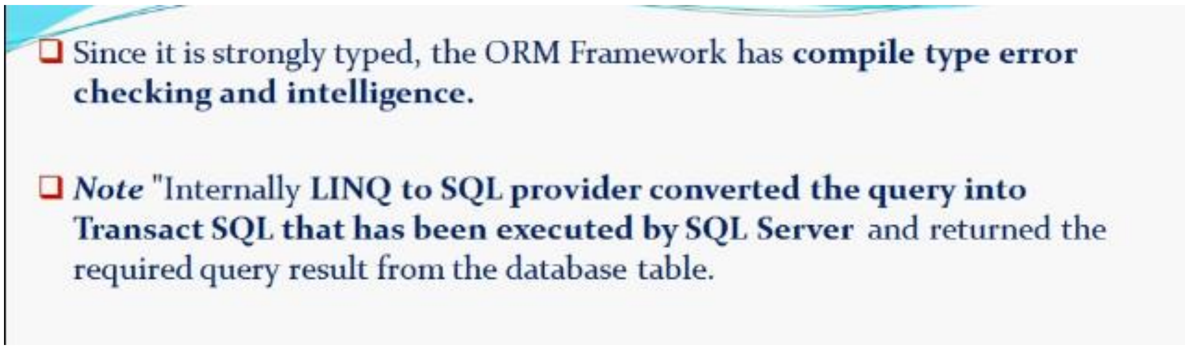
```
class Program
{
   static void Main(string[] args)
 {
  List<Student> lstStudents = new List<Student>
  {
   new Student() { RollNumber = 1,Name ="Alex " , Section = 13 },
   new Student() { RollNumber = 2,Name ="Jonty " , Section = 21 },
   new Student() { RollNumber = 3,Name ="Samba " , Section = 11 },
   new Student() { RollNumber = 4,Name ="Donald " , Section = 11},
   new Student() { RollNumber = 4,Name ="Donald " , Section = 11},
   };

  var distinctSection = (from p in lstStudents select new { p.Section, p.RollNumber, p.Name }).Distinct();
      foreach (var a in distinctSection)
      {
          Console.WriteLine(a.Section + ", " + a.Name+ ", " + a.Section);
      }
     Console.ReadLine();

}
```



```
13, Alex , 13
21, Jonty , 21
11, Samba , 11
11, Donald , 11
```

What is LINQ to SQL ?

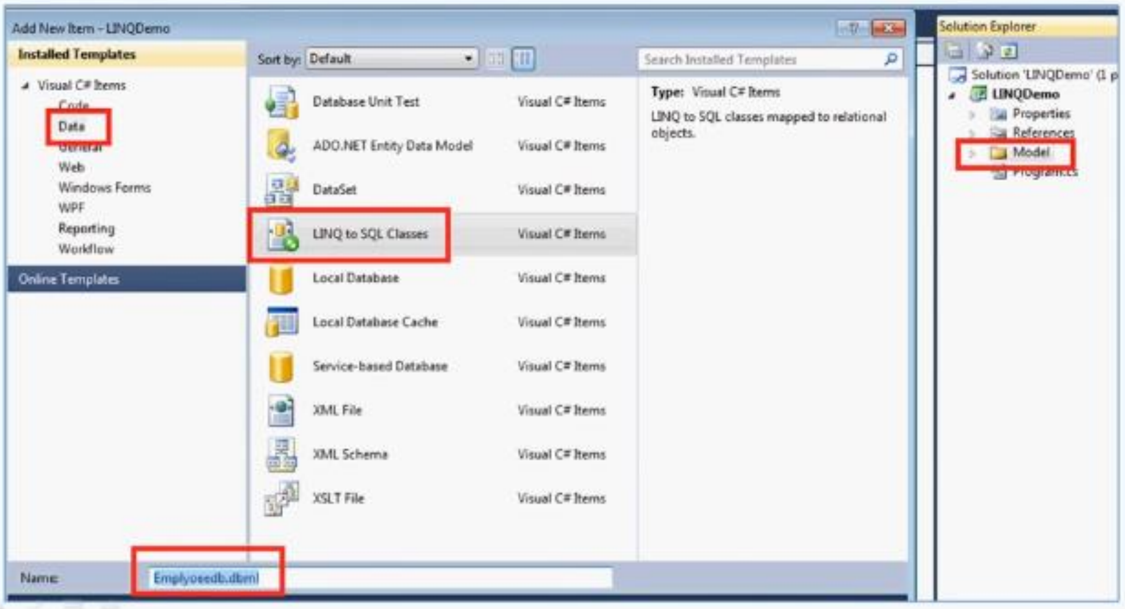
- It is specifically **designed to work with only SQL Server database.**
- It is a ORM framework **for converting LINQ queries into Transact SQL** that can be supported by SQL Server.
- Basically it will **create a strongly typed .NET class based on the database table** that are further used for the queries.
- Using LINQ technology to access SQL databases is similar to accessing an in-memory collection.
- LINQ to SQL **support transactions, views and Stored Procedures** and will work only with SQL Server databases.



- Since it is strongly typed, the ORM Framework has **compile type error checking and intelligence.**
- *Note* "Internally **LINQ to SQL provider converted the query into Transact SQL that has been executed by SQL Server** and returned the required query result from the database table.

How to Start working with LINQ to SQL. Explain with Example ?

Step 1 : Create a LINQ to SQL classes(Emplyoeedb.dbml) template in Model Folder

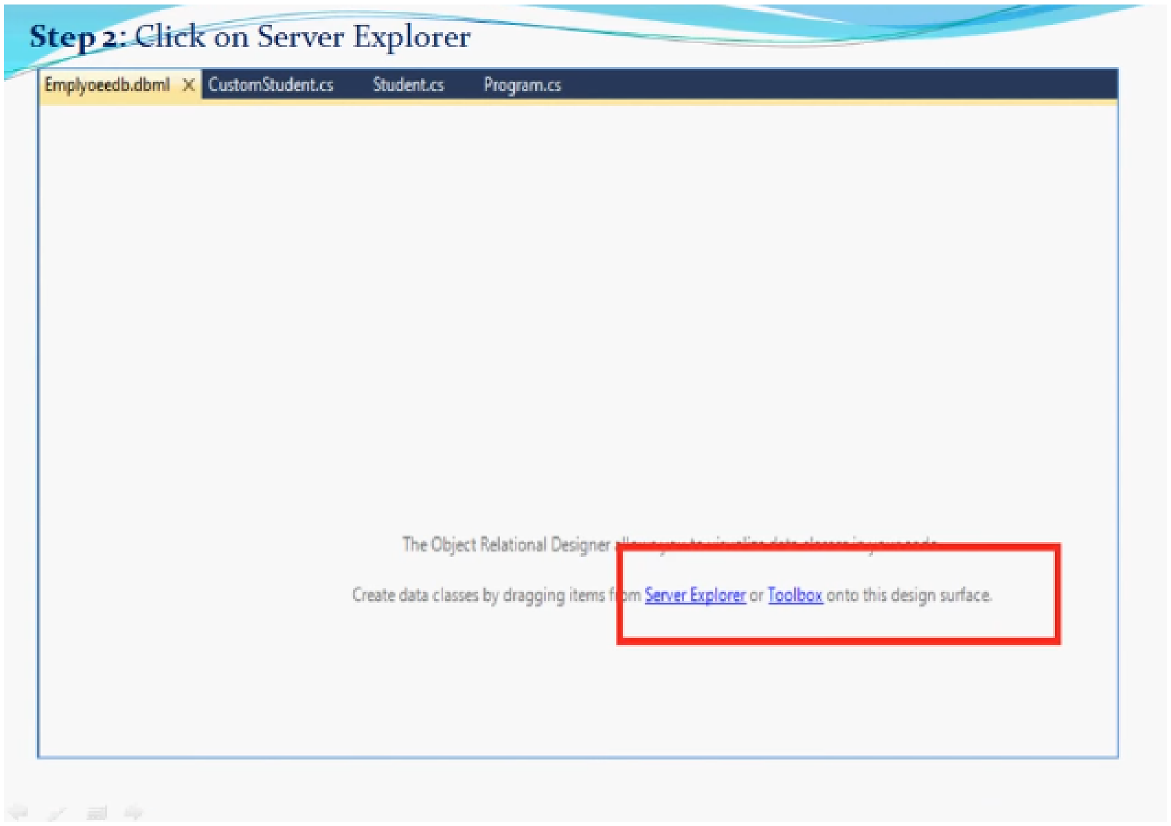

Add New Item - LINQDemo
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Sort by: Default
Search Installed Templates
Database Unit Test — Visual C# Items
ADO.NET Entity Data Model — Visual C# Items
DataSet — Visual C# Items
LINQ to SQL Classes — Visual C# Items
Local Database — Visual C# Items
Local Database Cache — Visual C# Items
Service-based Database — Visual C# Items
XML File — Visual C# Items
XML Schema — Visual C# Items
XSLT File — Visual C# Items
Solution Explorer
Solution 'LINQDemo' (1 p
LINQDemo
Properties
References
Model
Program.cs

Type: Visual C# Items

LINQ to SQL classes mapped to relational objects.

Name: Emplyoeedb.dbml

Step 2: Click on Server Explorer

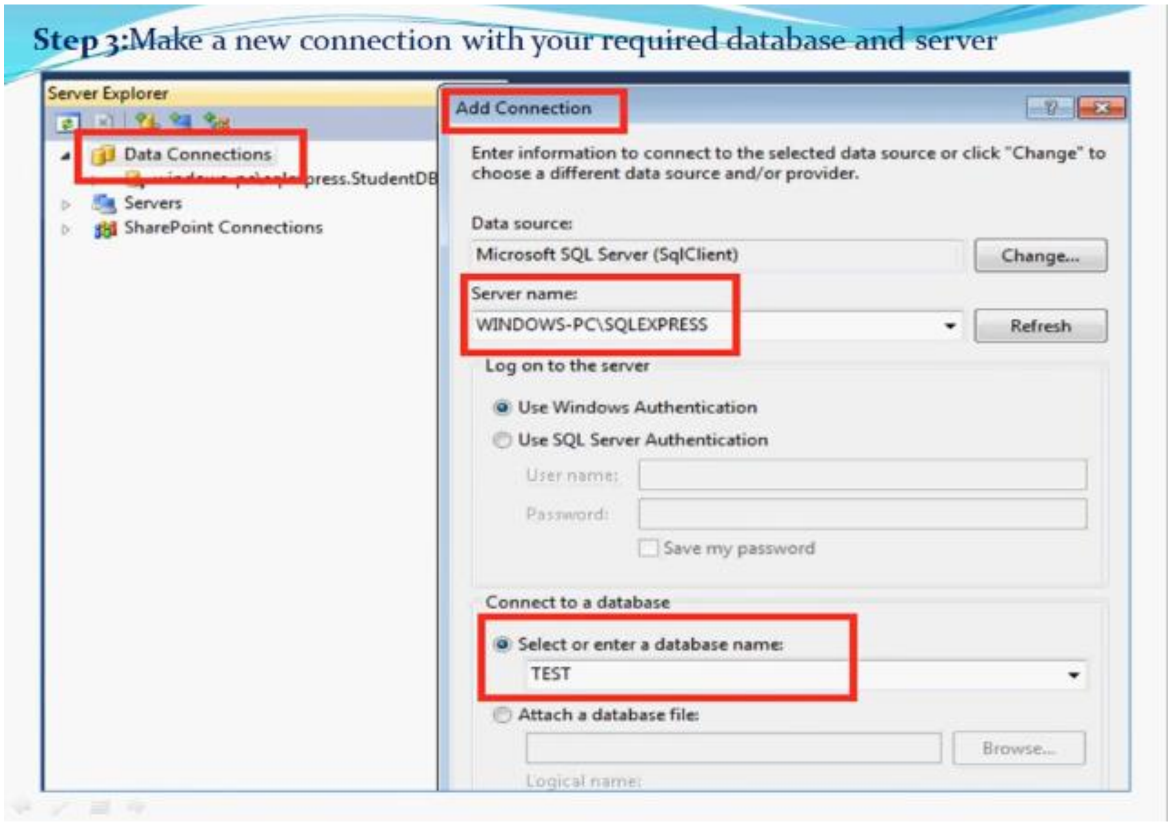


Emplyoeedb.dbml × CustomStudent.cs Student.cs Program.cs

The Object Relational Designer allows you to visualize data classes in your code.

Create data classes by dragging items from Server Explorer or Toolbox onto this design surface.

Step 3: Make a new connection with your required database and server



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient)

Server name:
WINDOWS-PC\SQLEXPRESS

Log on to the server

[✓ Use Windows Authentication]
Use SQL Server Authentication

User name:
Password:
Save my password

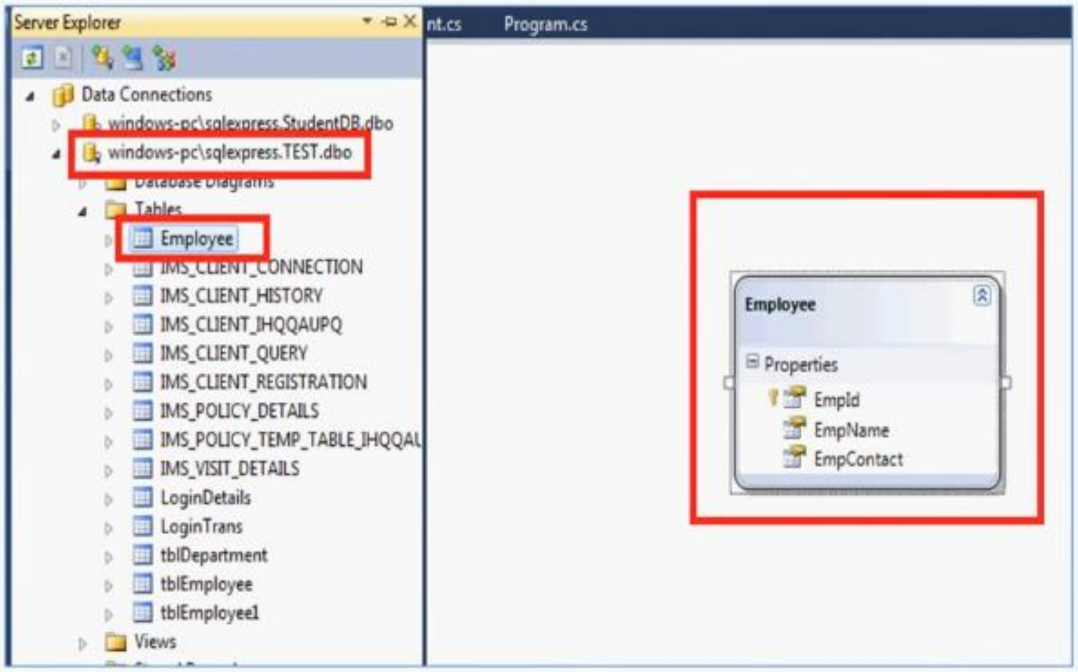
Connect to a database

[✓ Select or enter a database name:]
TEST

Attach a database file:

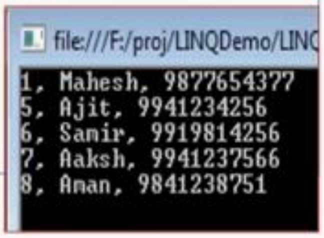
Logical name:

Step 4:Select a required table from Database and build the application

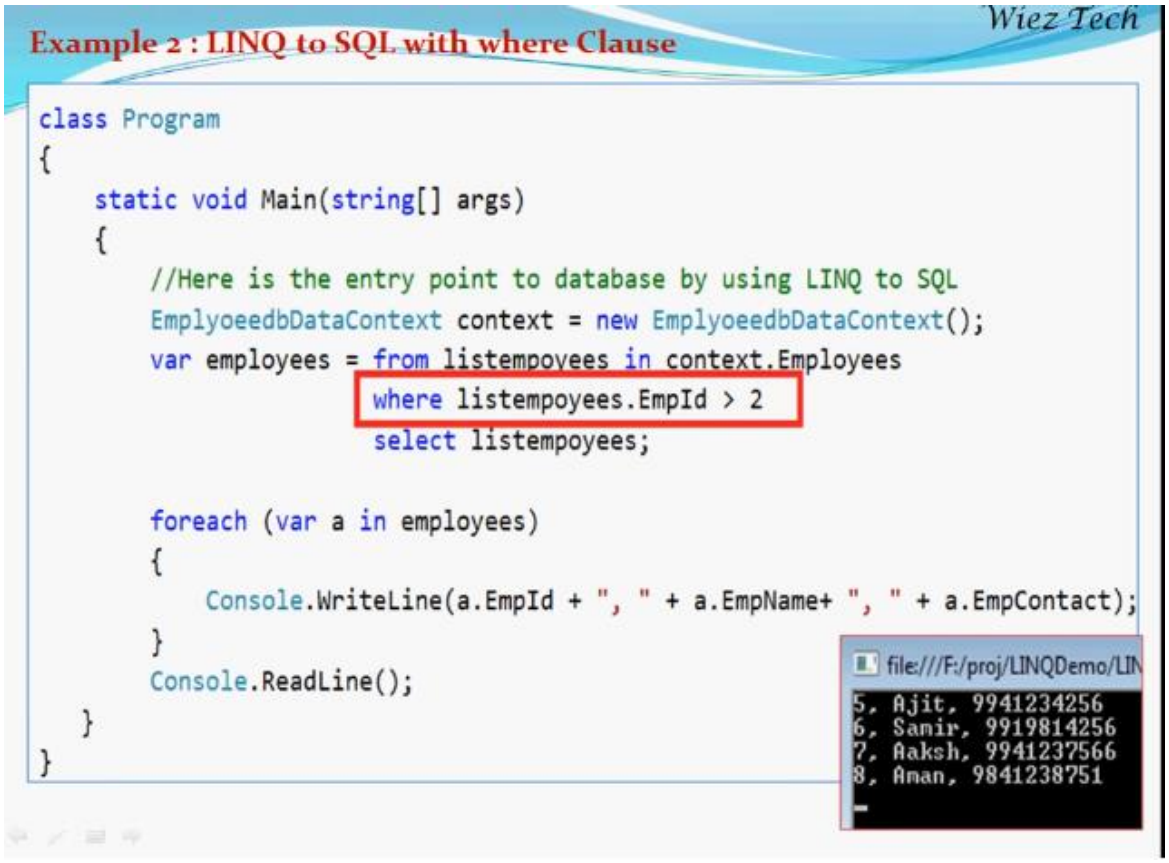


Step 5:Retieve all Employee Details from database by using LINQ to SQL

```
class Program
{
    static void Main(string[] args)
    {
        //Here is the entry point to database by using LINQ to SQL
        EmplyoeedbDataContext context = new EmplyoeedbDataContext();
        var employees = from listempoyees in context.Employees
                        select listempoyees;
        foreach (var a in employees)
        {
            Console.WriteLine(a.EmpId + ", " + a.EmpName + ", " + a.EmpContact);
        }
        Console.ReadLine();
    }
}
```



```
1, Mahesh, 9877654377
5, Ajit, 9941234256
6, Samir, 9919814256
7, Aaksh, 9941237566
8, Aman, 9841238751
```



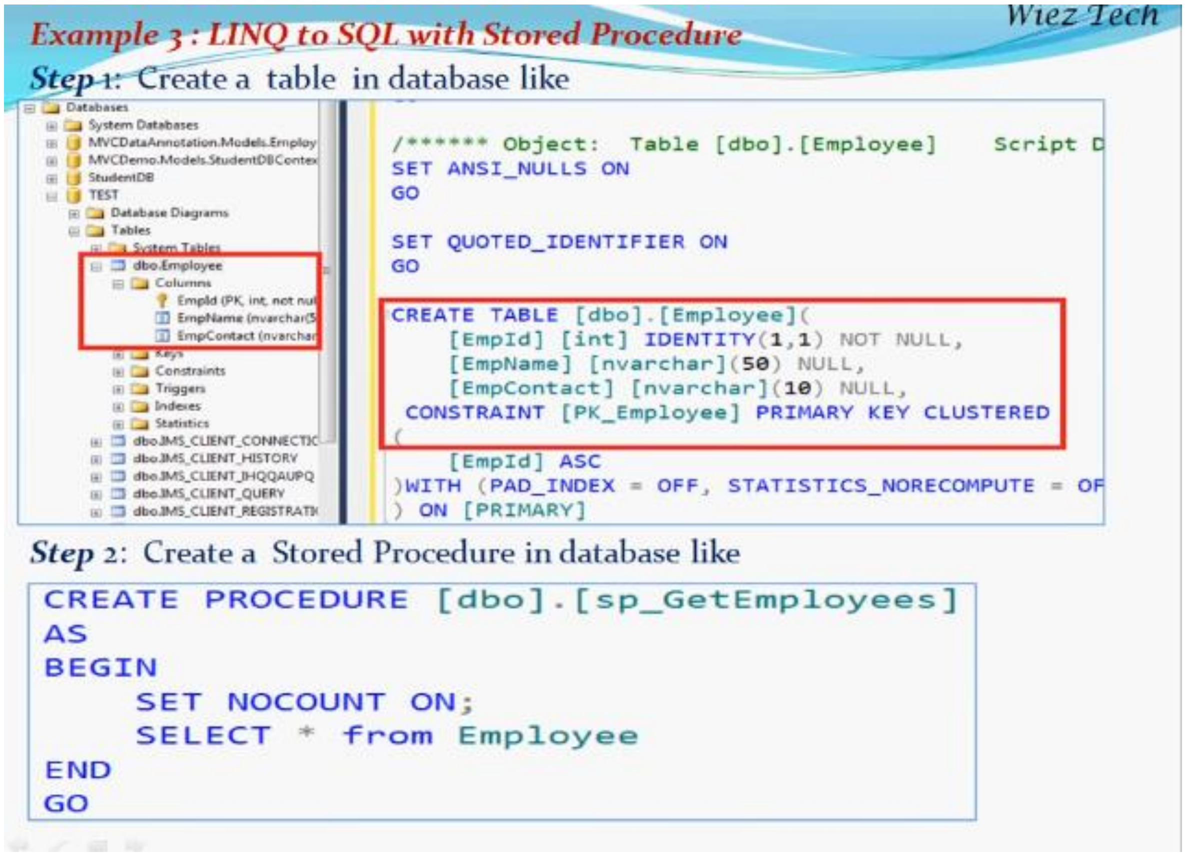
Wiez Tech

Example 2 : LINQ to SQL with where Clause

```
class Program
{
    static void Main(string[] args)
    {
        //Here is the entry point to database by using LINQ to SQL
        EmplyoeedbDataContext context = new EmplyoeedbDataContext();
        var employees = from listempoyees in context.Employees
                        where listempoyees.EmpId > 2
                        select listempoyees;

        foreach (var a in employees)
        {
            Console.WriteLine(a.EmpId + ", " + a.EmpName+ ", " + a.EmpContact);
        }
        Console.ReadLine();
    }
}
```

file:///F:/proj/LINQDemo/LIN

```
5, Ajit, 9941234256
6, Samir, 9919814256
7, Aaksh, 9941237566
8, Aman, 9841238751
```



Wiez Tech

Example 3 : LINQ to SQL with Stored Procedure

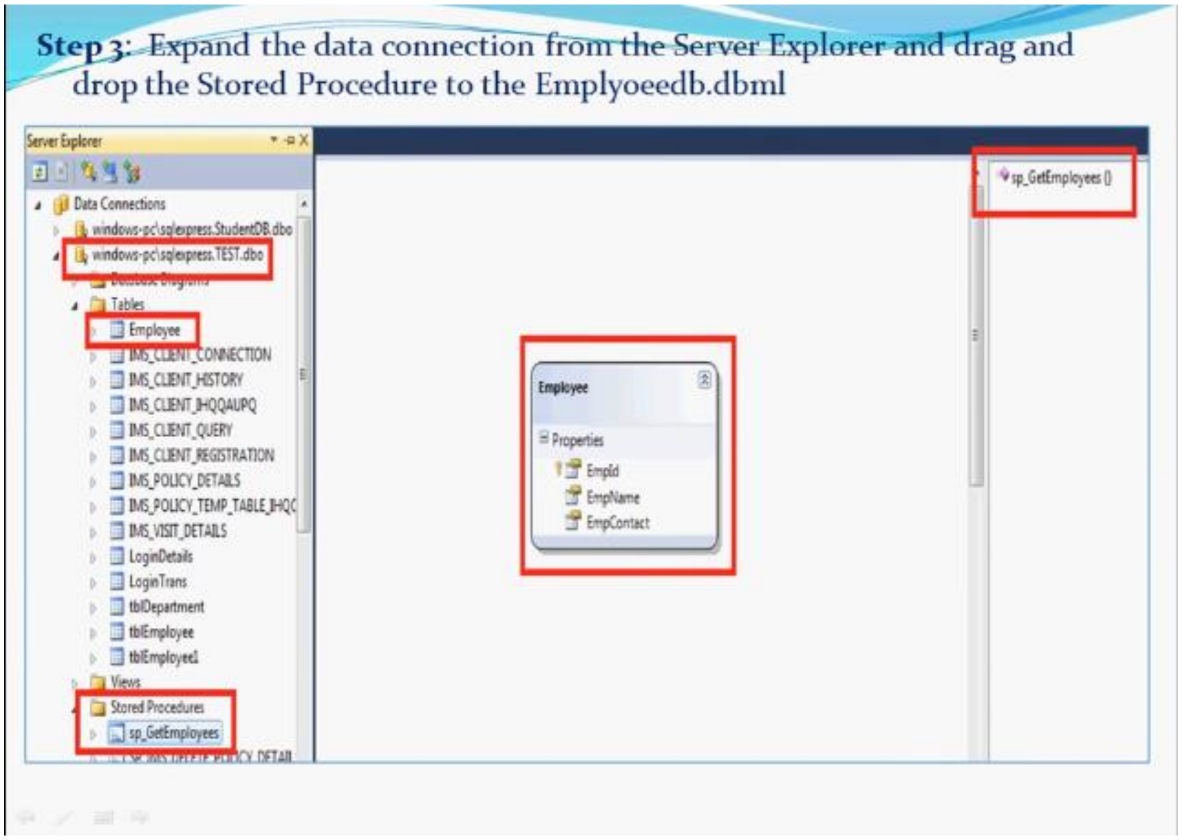
***Step* 1:** Create a table in database like

```
Databases
  System Databases
  MVCDataAnnotation.Models.Employ
  MVCDemo.Models.StudentDBContex
  StudentDB
  TEST
    Database Diagrams
    Tables
      System Tables
      dbo.Employee
        Columns
          EmpId (PK, int, not nul
          EmpName (nvarchar(5
          EmpContact (nvarchar
        Keys
        Constraints
        Triggers
        Indexes
        Statistics
      dbo.JMS_CLIENT_CONNECTIC
      dbo.JMS_CLIENT_HISTORY
      dbo.JMS_CLIENT_IHQQAUPQ
      dbo.JMS_CLIENT_QUERY
      dbo.JMS_CLIENT_REGISTRATI
```

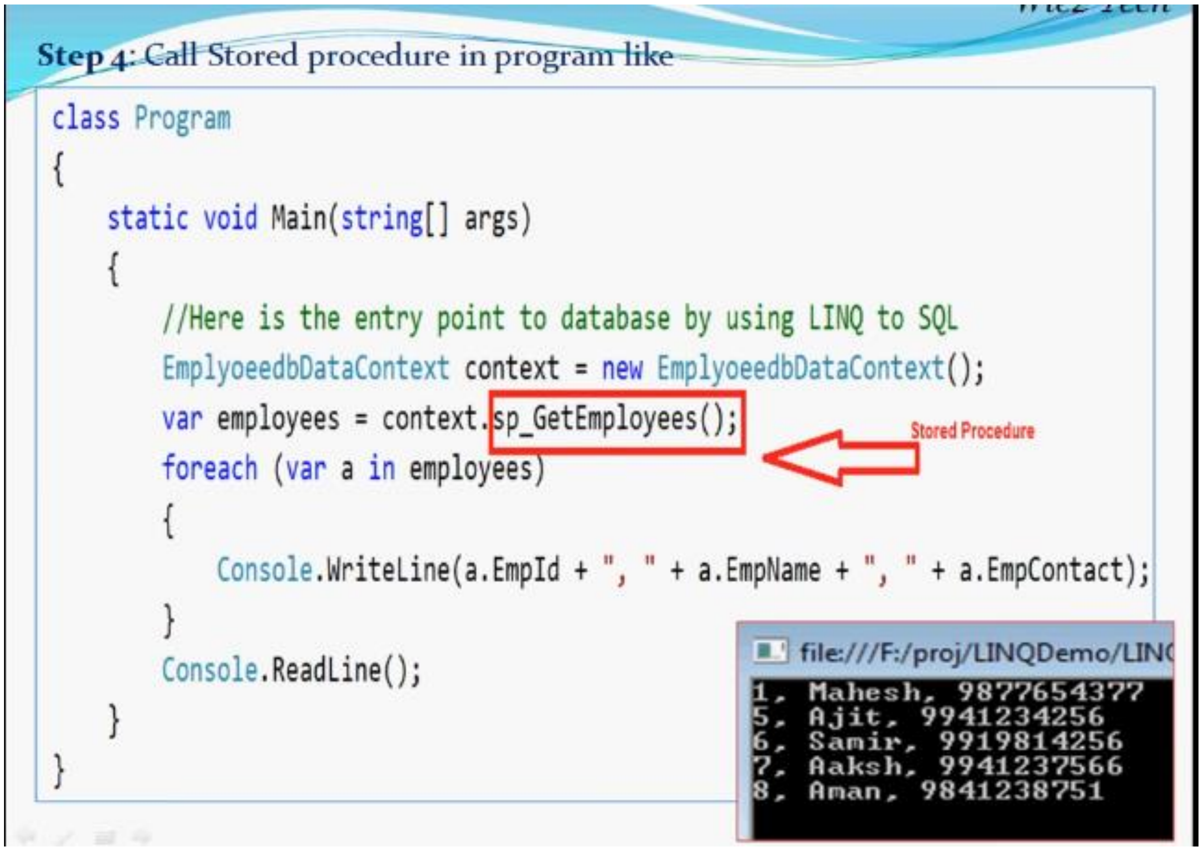
```
/****** Object:  Table [dbo].[Employee]    Script D
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Employee](
    [EmpId] [int] IDENTITY(1,1) NOT NULL,
    [EmpName] [nvarchar](50) NULL,
    [EmpContact] [nvarchar](10) NULL,
 CONSTRAINT [PK_Employee] PRIMARY KEY CLUSTERED
(
    [EmpId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OF
) ON [PRIMARY]
```

***Step* 2:** Create a Stored Procedure in database like

```
CREATE PROCEDURE [dbo].[sp_GetEmployees]
AS
BEGIN
    SET NOCOUNT ON;
    SELECT * from Employee
END
GO
```



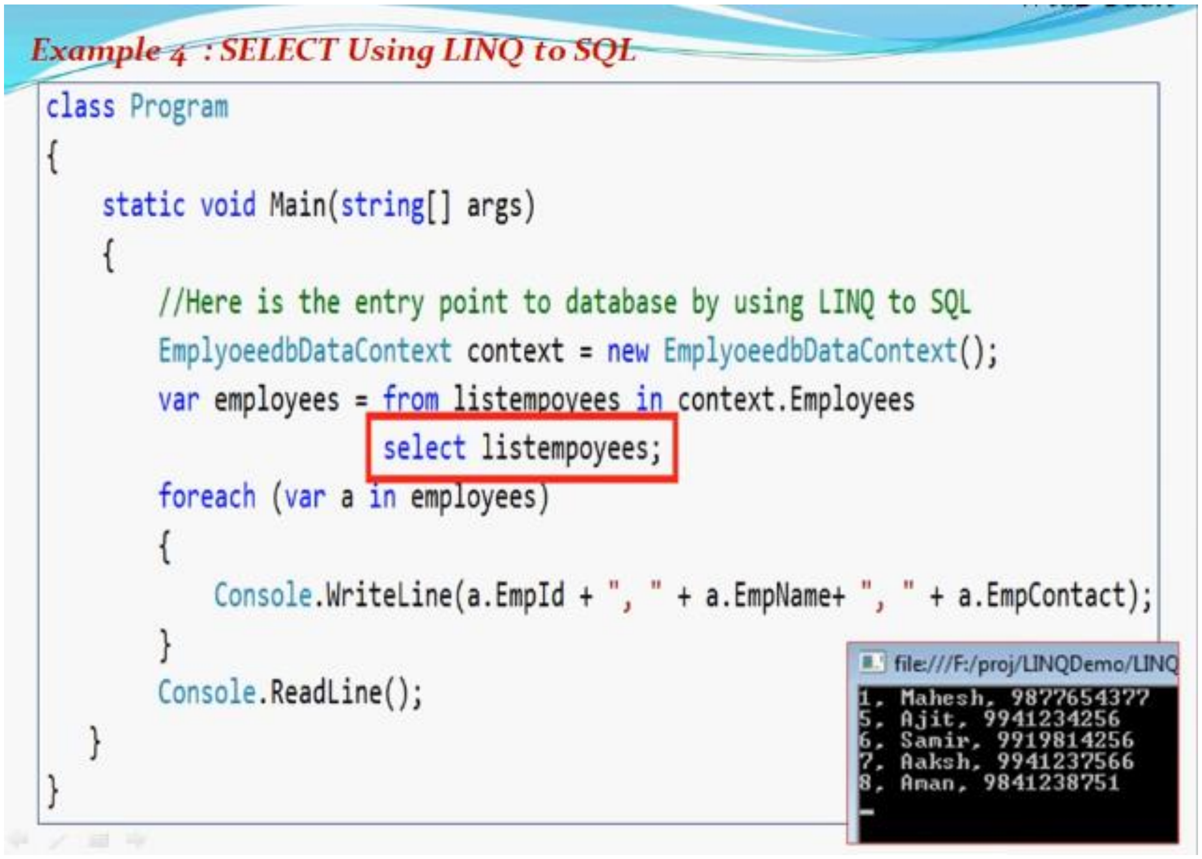
Step 3: Expand the data connection from the Server Explorer and drag and drop the Stored Procedure to the Emplyoeedb.dbml



Step 4: Call Stored procedure in program like

```
class Program
{
    static void Main(string[] args)
    {
        //Here is the entry point to database by using LINQ to SQL
        EmplyoeedbDataContext context = new EmplyoeedbDataContext();
        var employees = context.sp_GetEmployees();
        foreach (var a in employees)
        {
            Console.WriteLine(a.EmpId + ", " + a.EmpName + ", " + a.EmpContact);
        }
        Console.ReadLine();
    }
}
```

```
1, Mahesh, 9877654377
5, Ajit, 9941234256
6, Samir, 9919814256
7, Aaksh, 9941237566
8, Aman, 9841238751
```

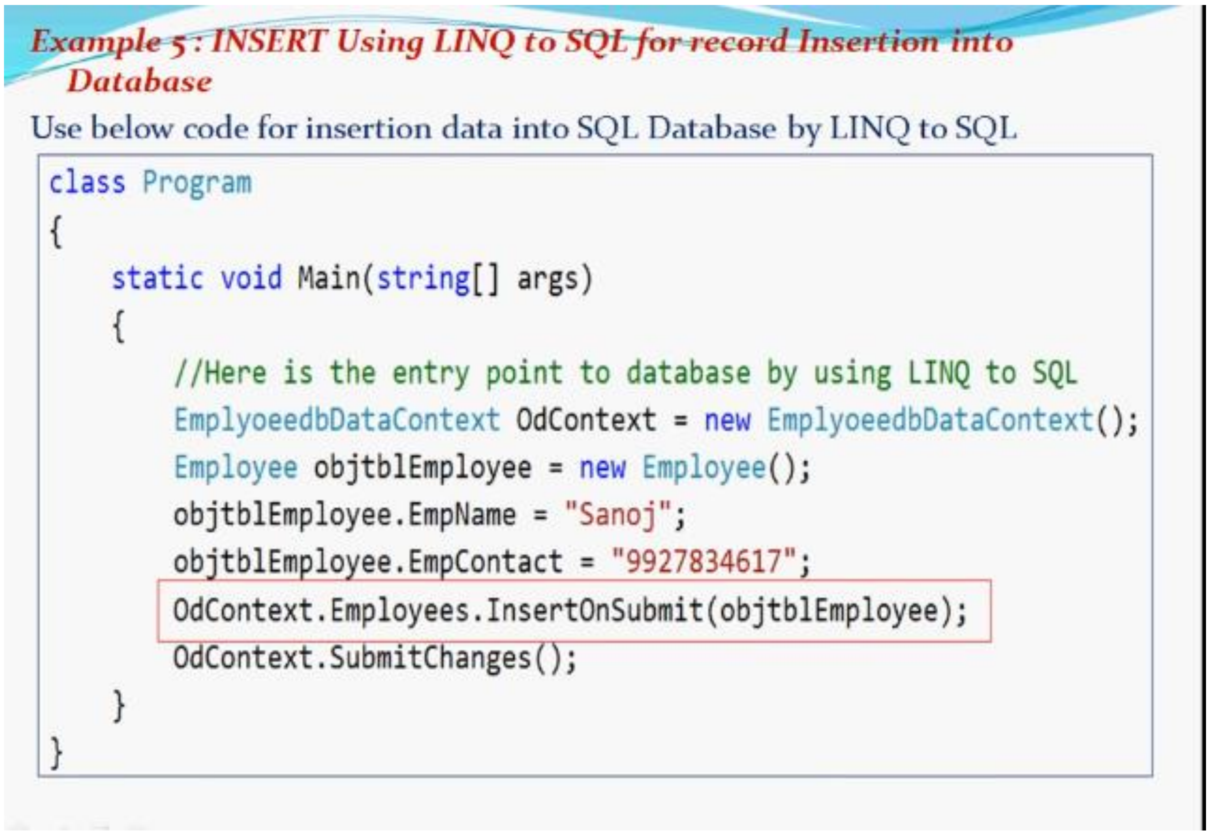


Example 4 : SELECT Using LINQ to SQL

```
class Program
{
    static void Main(string[] args)
    {
        //Here is the entry point to database by using LINQ to SQL
        EmplyoeedbDataContext context = new EmplyoeedbDataContext();
        var employees = from listempoyees in context.Employees
                        select listempoyees;
        foreach (var a in employees)
        {
            Console.WriteLine(a.EmpId + ", " + a.EmpName+ ", " + a.EmpContact);
        }
        Console.ReadLine();
    }
}
```

file:///F:/proj/LINQDemo/LINQ

```
1, Mahesh, 9877654377
5, Ajit, 9941234256
6, Samir, 9919814256
7, Aaksh, 9941237566
8, Aman, 9841238751
_
```

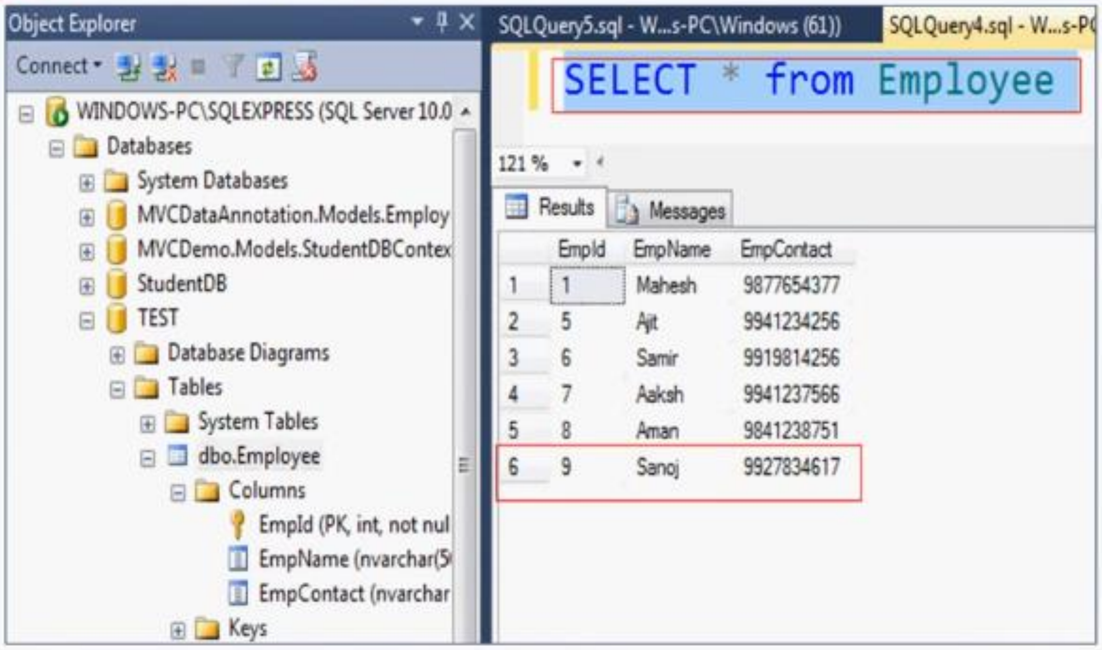


Example 5 : INSERT Using LINQ to SQL for record Insertion into Database

Use below code for insertion data into SQL Database by LINQ to SQL

```
class Program
{
    static void Main(string[] args)
    {
        //Here is the entry point to database by using LINQ to SQL
        EmplyoeedbDataContext OdContext = new EmplyoeedbDataContext();
        Employee objtblEmployee = new Employee();
        objtblEmployee.EmpName = "Sanoj";
        objtblEmployee.EmpContact = "9927834617";
        OdContext.Employees.InsertOnSubmit(objtblEmployee);
        OdContext.SubmitChanges();
    }
}
```

Run the Below query and get the new inserted data into database



```
SELECT * from Employee
```

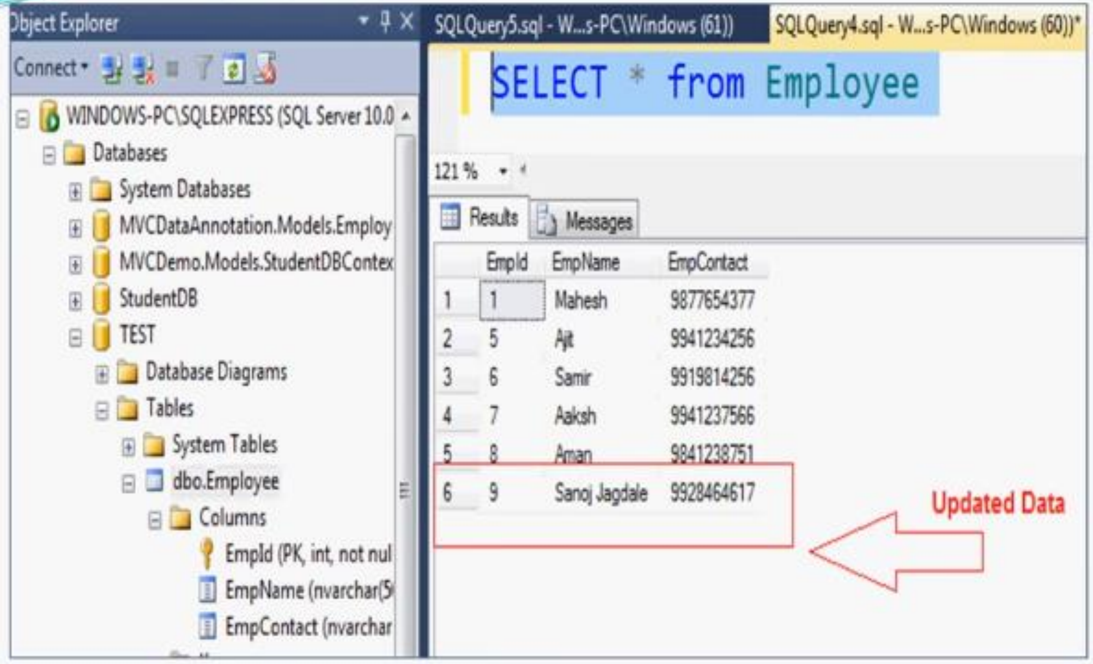
	EmpId	EmpName	EmpContact
1	1	Mahesh	9877654377
2	5	Ajit	9941234256
3	6	Samir	9919814256
4	7	Aaksh	9941237566
5	8	Aman	9841238751
6	9	Sanoj	9927834617

Example 6 : UPDATE Using LINQ to SQL for record updating into Database

Use below code for updating data into SQL Database by LINQ to SQL

```
class Program
{
    static void Main(string[] args)
    {
        //Here is the entry point to database by using LINQ to SQL
        EmplyoeedbDataContext OdContext = new EmplyoeedbDataContext();
        Employee objtblEmployee = OdContext.Employees
            .Single(m => m.EmpId == 9);
        objtblEmployee.EmpName = "Sanoj Jagdale";
        objtblEmployee.EmpContact = "9928464617";
        OdContext.SubmitChanges();
    }
}
```

Run the Below query and get the updated data into database

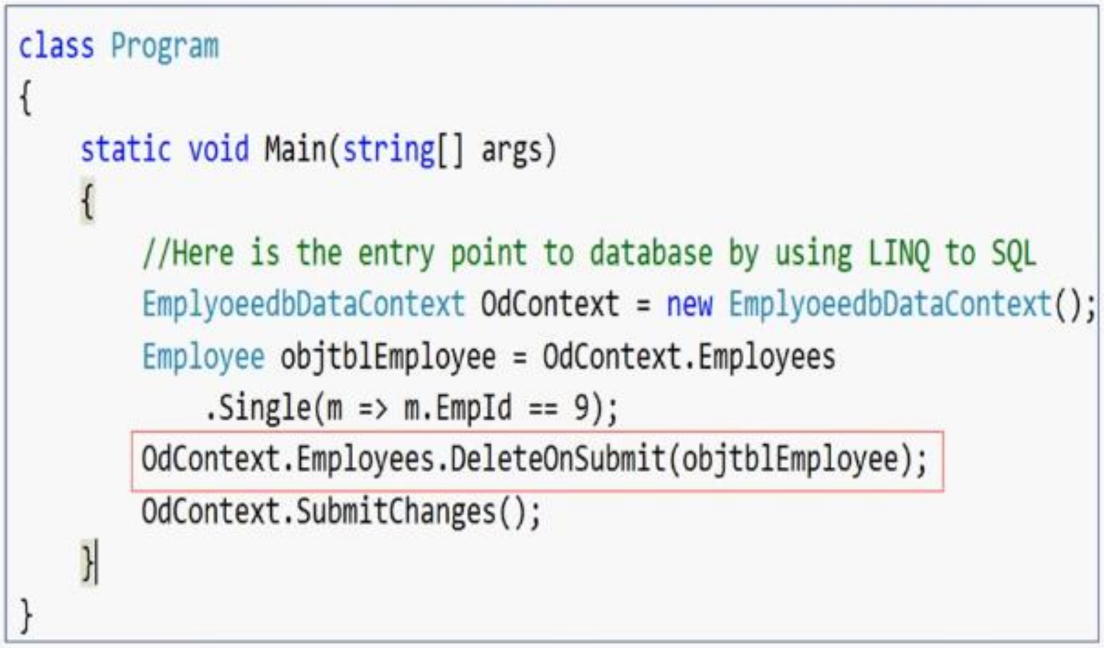


```sql
SELECT * from Employee
```

	EmpId	EmpName	EmpContact
1	1	Mahesh	9877654377
2	5	Ajit	9941234256
3	6	Samir	9919814256
4	7	Aaksh	9941237566
5	8	Aman	9841238751
6	9	Sanoj Jagdale	9928464617

Example 7: DELETE Using LINQ to SQL

Use below code for deletion data into SQL Database by LINQ to SQL



```csharp
class Program
{
    static void Main(string[] args)
    {
        //Here is the entry point to database by using LINQ to SQL
        EmplyoeedbDataContext OdContext = new EmplyoeedbDataContext();
        Employee objtblEmployee = OdContext.Employees
            .Single(m => m.EmpId == 9);
        OdContext.Employees.DeleteOnSubmit(objtblEmployee);
        OdContext.SubmitChanges();
    }
}
```