Do you have any experience using LINQ?

**Yes/No**

How strong are you working with LINQ?

**Beginner/Intermediate/Excellent**

**Basic Questions**

1. LINQ Stands “**Language Integrated Query”**
2. **What is LINQ**? “Provides a common syntax for querying data similar to SQL from various types like **Objects, XML, Datasets and Entities**”. With just one namespace: “System.Linq” all the capabilities we can achieve.
3. **Why do we go for LINQ**? -
   1. Problem with existing approach:
      1. **Scenario:** In c# application, task is to work with objects and xmls to perform read or write operations but we need to learn how to write queries for these two entity types. In future, again they asked us to work with datasets or any then needs to learn how to write queries.
      2. **Solution**: using this matured LINQ provides a common syntax to query on all these types: Objects, XML , Datasets and Entities.
      3. **Advantage:** Single syntax is better than learning multiple type syntax.
4. Write a small linq program
   1. Input: Int
   2. Type of Entity: Lists
   3. Output: one or more numbers to return
   4. Program Name: Print the numbers where value is less than 3.
   5. Mode: Any (Query/Lambda)
   6. **Program**

**// See https://aka.ms/new-console-template for more information**

**Console.WriteLine("Existing .NET Features - Without LINQ");**

**List<int> numbers = new List<int>{1,2,3,4,5,6};**

**foreach(var number in numbers)**

**{**

**if(number<3)**

**{**

**Console.WriteLine(number);**

**}**

**}**

**Console.WriteLine("Using Linq - Query Syntax");**

**// Step 1: Getting data source**

**List<int> numbersLinq = new List<int>{1,2,3,4,5,6};**

**// Step 2: Write Query - The below one is not executed its just hold the query.**

**var query = from number in numbersLinq**

**where number < 3**

**select number;**

**// Step 3: Execute the query**

**foreach(var num in query)**

**Console.WriteLine(num);**

**Using LinqPad**

// Print numbers whose value is less than 3

var numbers = new List<int>{1,2,3,4,5};

// Query Syntax

(from num in numbers

where num < 3

select num).Dump();

// Method Syntax with Lambda Expression

numbers.Where(num => num <3).Dump();

1. **To work with LINQ what exactly we require?**
   1. Data source - Either it may be Lists, database, xml
   2. Write a Linq Query - It may be Query Syntax or Lamda
   3. Execute the Query
2. **How do you define Lazy Execution?** - Based on the above program if there is no foreach statement then query is not going to trigger.
3. How do you define the LINQ Query syntax with basic example and define the purpose of the each operator.

var query = from number in numbersLinq // Part 1: Data Source: numbersLinq

where number < 3 // Part 2: Filter - Where => It might be one or more filters

select number; // Part 3: Return the results - Select => Either single value, object,

1. Why LINQ?
   1. Its a strong type so its compiled at compilation time
   2. Used to transform the data in different format
   3. LAZY?Defferred execution gives boost in performance
   4. More support in LINQ from Microsoft
2. **Sample Program - Print Even numbers**

Console.WriteLine("Print Even Numbers Using Linq - Query Syntax");

// Step 1: Getting data source

List<int> numbersLinq = new List<int>{1,2,3,4,5,6};

// Step 2: Write Query - The below one is not executed its just hold the query.

var queryEven = from number in numbersLinq // Part 1: Data Source: numbersLinq

where number % 2 == 0 // Part 2: Filter - Where => It might be one or more filters

select number; // Part 3: Return the results - Select => Either single value, object,

// Step 3: Execute the query

foreach(var num in queryEven)

Console.WriteLine(num);

1. LINQ Supports how many ways to write queries
   1. **Query Syntax** also there are other names like “Query Comprehension Syntax or Query Expression Syntax”
      1. Query Syntax follows SQL Declarative syntax approach
      2. Query Syntax always ends with Select or group statements
      3. Query Syntax is easy to read and write.
   2. **Method Syntax** is another way of writing LINQ Queries and these syntax uses Extension methods
      1. **Sample Code:**

Console.WriteLine("Using Linq - Method Syntax");

// Step 1: Getting data source

List<int> numbersLinqM = new List<int>{1,2,3,4,5,6};

// Step 2: Write Query - The one below is not executed, it's just holding the query.

var queryM = numbersLinqM.Where(GetLessThan3);

// Step 3: Execute the query

foreach(var num in queryM)

Console.WriteLine(num);

// Predicate function

static bool GetLessThan3(int num)

{

return num<3;

}

* + 1. Using Lambda Expression

// Another way is to write Lambda Expression

var queryM = numbersLinqM.Where(num => num <3);

// => means goes to - num goes to num < 3

// Lambda expression is also called as nameless functions.

* + 1. This is bit difficult to read and write
    2. Method syntax can be chained with one or more extension methods
  1. **Difference between Query vs Method Syntax**

| **Feature** | **Query Syntax** | **Method Syntax** |
| --- | --- | --- |
| List of LINQ operators support | **Limited** | **Supports all** |
| ‘select’ clause in syntax | **Mandatory** | **Optional** |
| Easy of use | **Simple to read and write** | **Bit difficult** |
| Linq Operators | **All in lowercase** | **Starts with uppercase** |
| Compile time action | **Converted to method syntax** | **Not applicable** |

**Note: Method syntax is super set to Query Syntax**

* 1. **Which syntax to use?**
     1. From microsoft point of view and their official statement, use method syntax when there is no way to achieve using query syntax.
  2. **Define a few LINQ Operators in general you used in applications?**
     1. **Filtering, Sorting, Projection,Set,....**

1. Sample program - to print square of each input

// Program to print square of each number

Console.WriteLine("Square using Method Syntax");

var queryMSquare = numbersLinqM.Select(x=> x \* x);

foreach(int num in queryMSquare)

{

Console.WriteLine(num);

}

Console.WriteLine("Square using Query Syntax");

var queryQSquare = from number in numbersLinq

select number;

// or

var queryQSquare = from number in numbersLinq

select number \* number;

foreach(int num in queryQSquare)

{

Console.WriteLine(num \* num);

// or

Console.WriteLine(num);

}

**OfType in LINQ Questions:**

1. OfType can use it in Query Syntax? **Ans:No, Its support only in Method Syntax**
2. Purpose of this ofType? Ans: Filtering specific type of data

**Ex:**

// ofType is also used to filter the records but specify the type

// ofType supports only with Method Syntax

/\* Ex: Student might be EnggStudent or MedicalStudent

If user wants only Engg Student then filter EnggStudents

Code: IEnumerable<EnggStudents> enggStudents = students.OfType<EnggStudents>();

\*/

**Sample Code for Where and OfType operators**

List<Student> students = new List<Student>();

Initialize();

// Initialize data for student model

void Initialize()

{

students.Add(new EnggStudents(1,"Nani","M"));

students.Add(new MedicalStudents(2,"Potti","V"));

// students.Add(new Student(1,"Nani","M"));

// students.Add(new Student(2,"Potti","V"));

}

// Query syntax

var query = from student in students

where student.StudentId == 1

select student;

foreach(var student in query)

Console.WriteLine("Student Name: "+ student.FirstName);

// ofType is also used to filter the records but specify the type

// ofType supports only with Method Syntax

/\* Ex: Student might be EnggStudent or MedicalStudent

If user wants only Engg Student then filter EnggStudents

Code: IEnumerable<EnggStudents> enggStudents = students.OfType<EnggStudents>();

\*/

Console.WriteLine("OfType - Method syntax");

var medicalStudents = students.OfType<MedicalStudents>();

foreach(var item in medicalStudents)

{

Console.WriteLine("Student firstName: " + item.FirstName);

}

**Select Operator** can define in three ways

1. **Define the type** ex: select student
2. **Define the properties** ex: select (student.FirstName, student.LastName)
3. **Define anonymous achieves using new keyword** ex: select new ( Id= student.studentId, Name = student.FirstName + student.LastName)

**Method Syntax:**

students.Select(student => new { Id= student.studentId, Name = student.FirstName + student.LastName});

**Note:** change from explict type to var when we define select keyword with anonymous and properties.

**SelectMany Operator**

void LinqSelectMany()

{

// Query syntax

Console.WriteLine("SelectMany - Query syntax");

var query = from student in students // Iterate each student

from a in student.FirstName.ToArray() // each student first name

select a;

foreach(var student in query)

Console.WriteLine(student);

Console.WriteLine("SelectMany - Method syntax");

var selectManyQuery = students.SelectMany(student => student.FirstName.ToArray(), (students, a) => (a));

foreach(var item in selectManyQuery)

{

Console.WriteLine(item);

}

}

**Question:** To reverse the collection using LINQ how do you achieve this?

**Ex code:** <Collection>.Reverse(), this is only available in Method Syntax

**Question:** Difference between GroupJoin vs GroupBy?

**GroupJoin** works on multiple collections

**GroupBy** works on single collection.

**Question:** Difference between GroupBy vs ToLookup?

**GroupBy** works based on key and read data using foreach.

**ToLookUp** works based on index and read data using for and pass this index to query variable.

**Question:** Is Element Operators supports in query syntax?

**Ans:** No, its supports only in Method Syntax.

**Question:** List of Element operators supports in Method Syntax?

**Ans:** ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault,Single, SingleOrDefault.

**ElementAt** operator uses to fetch the result from collection based on Index

**Ex:** Student student = students.ElementAt(0); => Returns single student information where index is 0(i.e., starting row from collection).

**Ex with Failure scenario:** Collection contains 2 records but user trying to fetch the record for 4th index. In this case what is going to be result?

**Ans:** Index out of range exception

**Question:**To handle this unhandled exception how you are going to fix the issue?

**Ans:**Use ElementAtOrDefault operator which gives **null** and checks if object is not null then only we read the data or else skip this logic.

**First()** operator - Fetches the first row from the collection. If incase no match found then it throws an exception “**Sequence contains no elements”.**

**Ex:** collection contains Nani,Potti but you are trying to find Ramya in that case you see the above error.

To fix this unhandled error then use **FirstOrDefault()** operator.

**Single()** operator throws an exception “**Sequence contains more than one element”** when user wants to fetch records matched with more than one with the same value.

**Note:** works only if collections contains unique records

**Question: what all Quantifier operators and purpose of these?**

**Ans:** All => Matches all the values in the condition then true else false

**Ex:** students.All(student => student.FirstName.Length > 5) => return false because one of collection value contains 4.

Any => Match atleast one record then true else false

**Ex:** students.Any(student => student.FirstName.Length > 5) => returns true

Contains => checks whether element is position at particular index

**Ex:** students.Contains(students.ElementAt(0)) => returns true if element is there at 0th position.

**Question:** What are available and support aggregate operators in Method Syntax?

**Ans:**Average, Count, Sum, Min and Max, LongCount

**Note:** These Aggregate operators are not available in **Query Syntax**.

**Ex:** students.Min(student => student.Id)

**Ex:** students.Max(student => student.Id)

**Ex:** students.Average(student => student.Id)

**Ex:** students.Count()

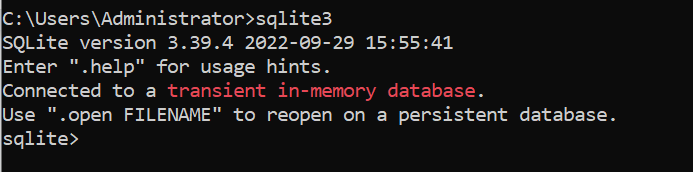
**Ex:** students.Sum(student => student.Id)

**LINQ To Entities** meant for reading and writing queries against Entity Framework models using c#.

**Steps to perform LINQ to Entities:**

1. **Create a database**
   1. **Ex: we are using sqlite**
      1. Download sqllite3 and add the sqlite3 path in environment variable.
      2. Check in command prompt >sqlite3

**Output:**

****

* + 1. Create a database

1. **Create a models**
   1. First add this namespace: **Microsoft.EntityFrameworkCore.Sqlite**
   2. Similar way which ever database we are using in our project we should add db package.
   3. Create models
   4. Create dbcontextfile and configure database file path
2. **Write queries to perform read and write operations.**
   1. Create a class/program file to read data from database.

**LINQ To XML** mainly we use to work with XML and manipulate these in inmemory, also converts into different structure of the file.

1. There are four main classes
   1. XElement
   2. XDocument
   3. XAttributes
   4. XName

**Reference Issues:**

// Issue or error: An expression tree may not contain a tuple literal #16995

// Reason is we added like this "(student.StudentId, student.FirstName)"

// Solution is to fix the above issue with "new {student.StudentId, student.FirstName}"

**LINQ to JSON:**

To work with JSON files using LINQ, there are several classes available. To know which class to use please find the following table which illustrate,

| **Symbol** | **Json Class** |
| --- | --- |
| {} | JObject |
| [] | JArray |
| KeyValuePair | JProperty |
| values | JValue |
| any | JToken |

**Question:**Could you please create a JSON manually using above classes

**Sample Input:**

{

“StudentId”:20,

“FirstName”:”Potti”,

“LastName”:”Nani”,

“Course”:{

“CourseId”:1,

“CourseName”:”Engineer”

}

}

**Namespace:** Newtonsoft.Json

**Code:**

using Newtonsoft.Json.Linq;

// See https://aka.ms/new-console-template for more information

Console.WriteLine("Hello, World!");

/\*

Sample Input 1:

{

“StudentId”:20,

“FirstName”:”Potti”,

“LastName”:”Nani”,

“Course”:{

“CourseId”:1,

“CourseName”:”Engineer”

}

}

\*/

JObject student = new JObject();

student .Add("StudentId",20);

student .Add("FirstName","Potti");

student .Add("LastName","Nani");

JObject course= new JObject();

course.Add("CourseId",1);

course.Add("CourseName","Engineer");

student.Add("Course",course);

// Console.WriteLine(student);

/\*

Sample Input 2:

{

"Students":{

"Student":{

“StudentId”:20,

“FirstName”:”Potti”,

“LastName”:”Nani”

}

}

}

\*/

JObject root = new JObject(

new JProperty("Students",

new JObject(

new JProperty("Student", new JObject(

new JProperty("StudentId",20),

new JProperty("FirstName","Potti"),

new JProperty("LastName","LastName")

))

)));

// Console.WriteLine(root);

/\*

Sample Input 3:

{

"Students":[

{

"StudentId":1,

"FirstName":"Potti",

"LastName":"V"

},

{

"StudentId":2,

"FirstName":"Nani",

"LastName":"M"

}

]

}

\*/

// Approach 1 but not with Linq

JObject rootWithOutLinq = new JObject(

new JProperty("Students",

new JArray(new JObject(

new JProperty("StudentId",1),

new JProperty("FirstName","Potti"),

new JProperty("LastName","V")

),

new JObject(

new JProperty("StudentId",2),

new JProperty("FirstName","Nani"),

new JProperty("LastName","M")

))

));

Console.WriteLine(rootWithOutLinq);

List<Student> students = new List<Student>()

{

new Student{StudentId =1 , FirstName = "Potti",LastName="V"},

new Student{StudentId =2 , FirstName = "Nani",LastName="M"}

};

// Approach 2 with Linq but assume that students collection should be create

JObject rootLinq = new JObject(

new JProperty("Students",

new JArray(students.Select(student=> new JObject(

new JProperty("StudentId", student.StudentId),

new JProperty("FirstName", student.FirstName),

new JProperty("LastName", student.LastName)

)))

));

Console.WriteLine(rootLinq);

public class Student{

public int StudentId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

}

**Question:** How to convert object to JSON

// Convert object to JSON

Student stu = new Student();

stu.StudentId = 1;

stu.FirstName="Naniv";

stu.LastName="Potti";

JObject objToJson = JObject.FromObject(stu);

Console.WriteLine(objToJson);

**Question:** How to parse String json?

// ParsingJSON from string

string jsonStr = @"{

'StudentId': 2,

'FirstName': 'Ramyaaa',

'LastName': 'Sree'

}";

JObject parseString = JObject.Parse(jsonStr);

Console.WriteLine(parseString);

**Question:** How to parse JSON from a file

// Parse JSON from a file

JToken token;

using(StreamReader sr = new StreamReader("Sample.json"))

{

token = JToken.ReadFrom(new JsonTextReader(sr));

}

Console.WriteLine(token);

**Question:** How to query JSON using property name?

// Read property data using jobject

Student stu1 = new Student();

stu1.StudentId = 1;

stu1.FirstName="Nanip";

stu1.LastName="Potti";

JObject objToJsonProp = JObject.FromObject(stu1);

Console.WriteLine(objToJsonProp["FirstName"]);

**Question:** How to query JSON using collection index?

// Querying from Collection Data

JArray studentsArray = (JArray)token["Students"];

Console.WriteLine(studentsArray.Count);

foreach(var item in studentsArray)

{

Console.WriteLine("StudentId: " + item["StudentId"] +" and FirstName "+ item["FirstName"]);

}

**Question:** How to query JSON using LINQ?

// Querying using LINQ

var query = from s in token["Students"]

select s["FirstName"];

foreach(var stuu in query)

{

Console.WriteLine(stuu);

}

**Question:** How to query JSON using SelectToken?

int studentId = (int)token.SelectToken("Students[1].StudentId");

Console.WriteLine("Student at 1st index ID: "+ studentId);

// Get collection

var queryToken = token.SelectToken("Students[1]");

foreach(var item in queryToken)

{

Console.WriteLine(item);

}

**Important “dotnet core” command prompt commands**

> dotnet new console

> dotnet new classlib

> dotnet add reference ..\Folder\.csproj

> dotnet build

> dotnet run

> dotnet add package <packagname>