LINQ

מבוא:

כמפתחים רוב עיסוקנו הוא ביצוע מניפולציות על מידע.
מידע יכול להגיע ממגוון מקורות: מסדי נתונים, קבצי XML, קבצים וכו'.
לעיתים קרובות, באפליקציה אחת נדרש מידע המגיע ממספר סוגים של מקורות נתונים, עובדה זו מקשה עלינו המפתחים בשל הצורך ללמוד ולהכיר "שפות" שונות וטכניקות שונות, הקידוד הנדרש לביצוע פעולה על נתונים המגיעים ממסד נתונים שונה מצורת הקידוד כאשר המידע מגיע מקובץ XML, ושונה מצורת הכתיבה הנדרשת לביצוע מניפולציה על נתונים הנמצאים באוסף.

LINQ (Language Integrated Query) מספקת לנו מודל ותחביר אחידים לטיפול במידע ללא קשר למקור הנתונים ממנו הוא מגיע, היא מאפשרת להגדיר שאילתות כחלק אינטגראלי משפת התכנות בה אנו משתמשים.
Linq מאפשרת ביצוע שאילתות חיפוש, סינון, סידור, צירוף, איחוד לקבוצות של נתונים וכו'
תחביר שפת השאילתות מובנית כחלק אינטגראלי בשפת C# 3.0.

Linq מחולקת למספר תחומים:

Linq to Object – יכולת לתשאל מבני נתונים הנמצאים בזיכרון. (בזה אנחנו נעסוק כעת)

Linq to SQL – ביצוע שאילתות על מידע המגיע מ-SQL.

Linq to Dataset – ביצוע שאילתות על מידע המגיע מ- Datasets או Datatables

Linq to Entities – בצוע שאילתות על מידע הנמצא בישויות מידע, טכניקה שנתמכת ב- .NET Framework 3.5.

Linq to XML – בצוע שאילתות הנמצאות בקבצי XML.

[מיקרוסופט רואה ב- Linq חשיבות רבה מאוד, חלק מהתכונות החדשות של שפת C#, כגון: var, Anonymous Type, Object Initialization, Lambda וכו' נוצרו על מנת לתמוך ב- Linq. ההיכרות עם התכונות הללו של השפה היא חיונית על מנת לתכנת עם שפת Linq. מפאת חוסר זמן אנו נאלץ ללמוד LINQ ללא הידע על כל אחד מהמרכבים הללו.]

מבנה תחבירי בסיסי

```
var positives =      from x in nums
                     where x > 0
                     select x + 2;
```

1. כל שאילתה מתחילה ב from
2. x או כל שם חוקי אחר של משתנה ב-#C, הוא משתנה המייצג כל אחד מהאלמנטים המוגדרים במקור המידע (nums בדוגמה זו).משתנה זה מוגדר על ידי var או בציון הטיפוס בצורה מפורשת , המשתנה הוא Strongly type ומכיוון שכך שגיאה בשאילתה תתגלה בזמן קומפילציה (לעומת שאילתת SQL רגילה הטעות תתגלה רק בזמן ריצה) וההשלמה האוטומטית של ה-IntelliSense זמינה גם בשאילתות. בדוגמה, x מוגדר על ידי var והוא מטיפוס int משום שכך מוגדר מקור המידע nums.
למרות זאת ניתן, אם נרצה בכך, להגדיר אותו בצורה מפורשת: from int x in nums.
בספרות המקצועית x מכונה identifier או range variable.
3. מקור המידע מופיע לאחר המילה in והוא חייב להיות מטיפוס הממש את IEnumerable או ממש את הגרסה הגנרית שלו IEnumerable<T> או ממש ממשק היורש ממשקים אלו.
4. where - יוצר תנאי לשאילתה (או פילטר אם נרצה – מסנן את הרשומות שמוחזרות), התנאי יכול להיות תנאי פשוט או תנאי מורכב תוך שימוש באופרטור AND (&&) או באופרטור OR (||), where הוא אופציונאלי.
השאילתה תחזיר רק את אותם אלמנטים שעומדים בתנאי שהוגדר באמצעות where.
5. select - מגדיר את הטיפוס של האלמנטים שיוחזרו מהשאילתה. הטיפוס <u>לא</u> חייב להיות זהה לטיפוס של האלמנטים במקור המידע.

[01_1 Linq basics]

```
//DATA SOURCE
int[] nums = { 1, -2, 3, -4 , 7, 10, -12};

//QUERY DEFINITION
var positives =      from x in nums
                     where x > 0
                     select x + 2;
//QUERY EXECUTION
foreach (var item in positives)
{
       Console.WriteLine(item);
}
```

בשלב ראשון הוגדר מבנה הנתונים, מבנה הנתונים חייב להיות סידרתי ומטיפוס הממשמש את IEnumerable.
בשלב שני הוגדרה השאילתה תוך שימוש בתחביר Linq , בשלב זה השאילתה מוגדרת אולם עדיין לא מתבצעת.
בשלב השלישי, השאילתה מתבצעת וערכיה המוחזרים מודפסים. `positives` חייב להיות מטיפוס הממשמש את הממשק IEnumerable.

לחילופין ניתן להגדיר את result בצורה יותר מפורשת:

```
IEnumerable<int> positives2 =      from x in nums
                                   where x > 0
                                  select x + 2;
```

פלט:

3

5

9

12

[01_2 Linq examples]

בדוגמה הבאה המזהה name הוא מטיפוס string ולכן ניתן להיעזר במתודות שלו לצורך בנית השאילתה, במידה וב- string לא הייתה מוגדרת המתודה StartsWith או לא הייתה המתודה Length הייתה מתקבלת שגיאת קומפילציה, זה אחד היתרונות של ניצול ה- Strong Type :

```
string[] names = {"beni", "dana", "zvi", "danny", "daniel", "avi", "danuba" ,
"dan"};

var short_danies =   from name in names
                     where name.StartsWith("dan") && name.Length <= 5
                      select name;

Console.WriteLine("short_danies:");
foreach (string item in short_danies)
{
       Console.WriteLine(item);
}
```

פלט:

short_danies:

dana

danny

dan

שאילתות Linq ניתן להריץ גם על אוסף אובייקטים.

```
class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public byte Grade { get; set; }
    public override string ToString()
    {
        return string.Format("id={0}, name={1}, grade={2}", Id,Name,Grade);
    }
}
```

...

```
        List<Student> myClass = new List<Student>
        {
            new Student{Id=1, Name="avi", Grade=80},
            new Student{Id=2, Name="dudu", Grade=50},
            new Student{Id=3, Name="zvi", Grade=90},
            new Student{Id=4, Name="beni", Grade=100},
        };

        var students = from student in myClass
                       where student.Grade > 60
                       select student;

        foreach (Student item in students)
        {
            Console.WriteLine(item);
        }
```

פלט:

```
id=1, name=avi, grade=80

id=3, name=zvi, grade=90

id=4, name=beni, grade=100

Press any key to continue . . .
```

בשאילתות Linq ניתן לשלב מחלקות, מתודות ומאפיינים המוגדרים ב- NET Framework. כמו למשל במתודה Math.Sqrt. שימו לב כי לא בכל המקרים האוסף שהוא תוצאת השאילתה הוא מאותו הטיפוס של האוסף שתושאל בשאילתה, Linq יודעת לבצע המרה (טרנספורמאציה) אוטומטית מטיפוס לטיפוס במקרה הצורך כפי שמציגה הדוגמה הבאה

```
int[] arr = {4,16,25,49 };
var nums = from num in arr
           select Math.Sqrt(num);

foreach (var item in nums)
{
    Console.Write(item.GetType() + ",\t ");
    Console.WriteLine(item);
}

Console.WriteLine("\n");
arr = new int[]{ 7, 17, 27, 49 };
nums = from num in arr
           select Math.Sqrt(num);

foreach (var item in nums)
{
    Console.Write(item.GetType() + ",\t ");
    Console.WriteLine(item);
}
```

פלט:

```
System.Double,   2

System.Double,   4

System.Double,   5

System.Double,   7


System.Double,   2.64575131106459

System.Double,   4.12310562561766

System.Double,   5.19615242270663

System.Double,   7

Press any key to continue . . .
```

מכיוון שהערך המוחזר של Math.Sqrt אינו מספר שלם אלא מטיפוס double האוסף שיווצר הוא של אלמנטים מטיפוס double.

Linq מאפשרת לתשאל ערכים ממבנה נתונים אחד ולהחזיר ערכים מתאימים ממבנה נתונים אחר

```
            int[] arr2 = {1,3,5,7 };
            string[] days = {"none","sunday" ,"monday" ,"tuesday" ,"wednesday"
,"thursday" ,"friday", "saturday" };

            var dayName = from x in arr2
                          select days[x];
            foreach (string item in dayName)
            {
                Console.WriteLine(item);
            }
```

פלט:

sunday

tuesday

thursday

saturday

Linq ו- Anonymous Types

Anonymous Types זו אחת התכונות החדשות שלבטח התווספו ל- C# 3.0 תוך מחשבה על הצרכים של Linq. בלא מעט מקרים, נגדיר טיפוס חדש אשר ידע להכיל את ערכי תוצאות השאילתה, ומכיוון שאין משמעות לטיפוס מעבר לצרכי השאילתה נגדירו כ-Anonymous Type

[01_3 Linq AnonymousType]

```
            int[] arr2 = { 1, 3, 5, 7 };
            string[] days = { "none", "sunday", "monday", "tuesday", "wednesday",
"thursday", "friday", "saturday" };

            var dayName = from x in arr2
                          select new
                              {
                                  dayNumber = x,
                                  day_name = days[x]
                              };

            foreach (var item in dayName)
            {
                Console.WriteLine(item);
            }
            Console.WriteLine();
            foreach (var item in dayName)
            {
                Console.WriteLine(item.dayNumber +": means " +  item.day_name);
            }
```

פלט:

```
{ dayNumber = 1, day_name = sunday }

{ dayNumber = 3, day_name = tuesday }

{ dayNumber = 5, day_name = thursday }

{ dayNumber = 7, day_name = saturday }


1: means sunday

3: means tuesday

5: means thursday

7: means saturday

Press any key to continue . . .
```

במחלקה מוגדרים מספר תכונות, לעיתים נהיה זקוקים לכולם אולם בלא מעט מקרים אנו נשתמש רק בחלקן לצורך פעילות מסוימות.
הדבר דומה לשליפת מידע ממסד נתונים, לעיתים נשלוף את כל השדות של טבלה מסוימת ובמקרים אחרים נשלוף רק את השדות הרלבנטיים למשימה שעומדת בפנינו.
באמצעות השימוש ב- Anonymous Types נוכל לברור רק את התכונות אותן אנו צריכים.

נניח כי המחלקה STUDENT היא זו:

```
        public int Id { get; set; }
        public string Name { get; set; }
        public byte Grade { get; set; }
        public DateTime Birthdate { get; set; }
        public byte Age { get; set; }
        public double Hight { get; set; }

        public override string ToString()
        {
            return string.Format("id={0}, name={1}, grade={2}", Id, Name, Grade);
        }

        static public List<Student> CreateClassOfStudents()
        {
            return new List<Student>
            {
                new Student{Id=1, Name="avi", Grade=80, Birthdate=new
DateTime(2000,1,27)},
                new Student{Id=2, Name="dudu", Grade=50, Birthdate=new
DateTime(1990,7,30)},
                new Student{Id=3, Name="zvi", Grade=90, Birthdate=new
DateTime(2005,5,5)},
                new Student{Id=4, Name="beni", Grade=100, Birthdate=new
DateTime(1992,11,22)},
            };
```

```
        }
```

אם נרצה לראות רק את שמותיהם וציונם של הסטודנטים שנולדו אחרי שנת 2000, ממויינים לפי הציונים בסדר הפוך (שימו לב לפקודה `orderby` ולאפשרות מיון ההפוכה `descending`), נוכל לעשות זאת כך:

```
            List<Student> ls = Student.CreateClassOfStudents();
            var students = from student in ls
                           where student.Birthdate > new DateTime(2000, 1, 1)
                            orderby student.Grade descending
                           select new
                           {
                               SName = student.Name,
                               SGrade = student.Grade
                           };

            //foreach (Student student in students)   //ERROR type missmatch!!
            foreach (var student in students)
            {
                Console.WriteLine("name={0}, grade={1}", student.SName,
student.SGrade);
            }
```

פלט:

```
name=zvi, grade=90

name=avi, grade=80

Press any key to continue.
```

כפי שניתן להתרשם, עיקר כוחן של ה- Anonymous Type הוא בביצוע שאילתות Linq.

[נסו לכתוב את השאילתה הזו ללא שימוש ב LINQ או כל שאר התוספות של C#3.0 אלה ע"י לולאות ומימוש של Icomparable– עכשיו אולי השתכנעתם ביעילות ובאלגנטיות של LINQ ☺]