C# LINQ ilə bağlı 50 Müsahibə Sualları və Cavabları

1. LINQ nədir?

Cavab: LINQ (Language Integrated Query) C# proqramlaşdırma dilində məlumat sorğularını yerinə yetirmək üçün istifadə olunan bir texnologiyadır. LINQ vasitəsilə kolleksiyalar, verilənlər bazaları, XML və digər məlumat mənbələri üzərində sorğular yazıla bilər. LINQ sorğuları SQL-ə bənzər sintaksisdən istifadə edir və C# dilinə inteqrasiya olunmuşdur.

2. LINQ-nun hansı növləri mövcuddur?

Cavab: LINQ-nun əsas növləri bunlardır:

- **LINQ to Objects**: Kolleksiyalar (List, Array və s.) üzərində sorğular.
- **LINQ to SQL**: Verilənlər bazaları ilə işləmək üçün (əsasən SQL Server).
- **LINQ to XML**: XML sənədləri ilə işləmək.
- **LINQ to DataSet**: ADO.NET DataSet-ləri ilə işləmək.
- **LINQ to Entities**: Entity Framework ilə işləmək.

3. LINQ sorğularının iki sintaksis növü hansılardır?

Cavab: LINQ sorğuları iki formada yazıla bilər:

- **Query Sintaksisi**: SQL-ə bənzər sintaksis, daha oxunaqlıdır.
- **Metod Sintaksisi**: Lambda ifadələri və metod çağırışları ilə yazılır.

4. LINQ Query Sintaksisi ilə Metod Sintaksisi arasındakı fərq nədir?

Cavab:

Query Sintaksisi: Daha oxunaqlı və SQL-ə bənzər formada yazılır. Məsələn:
var result = from x in numbers where x > 5 select x;

-

Metod Sintaksisi: Lambda ifadələri ilə metod çağırışları şəklində yazılır. Məsələn:
var result = numbers.Where(x => x > 5).Select(x => x);

-

5. LINQ-nun əsas üstünlükləri nələrdir?

Cavab:

- Kodun oxunaqlılığını artırır.
- Fərqli məlumat mənbələri ilə eyni sintaksisdə işləməyə imkan verir.
- Tip təhlükəsizliyini təmin edir.
- Kompilyator tərəfindən səhvlər erkən aşkarlanır.

6. LINQ-nun əsas operatorları hansılardır?

Cavab: LINQ operatorları aşağıdakı kateqoriyalara bölünür:

- **Filtrləmə**: `Where`, `OfType`
- **Sıralama**: `OrderBy`, `OrderByDescending`, `ThenBy`
- **Qruplaşdırma**: `GroupBy`
- **Birləşdirmə**: `Join`, `GroupJoin`
- **Seçmə**: `Select`, `SelectMany`
- **Aqreqasiya**: `Count`, `Sum`, `Min`, `Max`, `Average`

7. `Where` operatorunun funksiyası nədir?

Cavab: `Where` operatoru kolleksiyadan müəyyən şərtə uyğun elementləri filtrləmək üçün istifadə olunur. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
var result = numbers.Where(n => n % 2 == 0); // Cüt ədədləri seçir: 2, 4
```

8. `Select` operatoru nə iş görür?

Cavab: `Select` operatoru kolleksiyanın hər elementini yeni bir formaya çevirmək üçün istifadə olunur. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
var squared = numbers.Select(n => n * n); // 1, 4, 9
```

9. `SelectMany` ilə `Select` arasındakı fərq nədir?

Cavab:

- `Select` hər element üçün bir nəticə qaytarır.

- `SelectMany` isə hər element üçün bir kolleksiya qaytarır və bu kolleksiyaları birləşdirərək tək bir kolleksiya yaradır. Məsələn:

```
var lists = new List<List<int>> { new List<int> { 1, 2 }, new List<int> { 3, 4 } };
var flatList = lists.SelectMany(x => x); // 1, 2, 3, 4
```

10. `OrderBy` və `OrderByDescending` operatorları necə işləyir?

Cavab:

- `OrderBy` kolleksiyanı artan sıra ilə sıralayır.
- `OrderByDescending` isə azalan sıra ilə sıralayır. Məsələn:

```
var numbers = new List<int> { 3, 1, 4, 2 };
var sorted = numbers.OrderBy(n => n); // 1, 2, 3, 4
var sortedDesc = numbers.OrderByDescending(n => n); // 4, 3, 2, 1
```

11. `ThenBy` nədir və nə üçün istifadə olunur?

Cavab: `ThenBy` əlavə sıralama şərtləri tətbiq etmək üçün istifadə olunur. İlk olaraq `OrderBy` ilə sıralama aparılır, sonra `ThenBy` ilə ikinci dərəcəli sıralama edilir. Məsələn:

```
var students = new List<(string Name, int Age)>
{
    ("Ali", 20), ("Veli", 20), ("Ayşe", 19)
};
var result = students.OrderBy(s => s.Age).ThenBy(s => s.Name);
// Nəticə: Ayşe (19), Ali (20), Veli (20)
```

12. `GroupBy` operatoru nə iş görür?

Cavab: `GroupBy` kolleksiyanı müəyyən bir açara görə qruplara bölür. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
var groups = numbers.GroupBy(n => n % 2);
// Nəticə: 0 (cüt): 2, 4; 1 (tək): 1, 3, 5
```

13. `Join` operatoru nədir?

Cavab: `Join` iki kolleksiyanı ümumi açar əsasında birləşdirir. Məsələn:

```
var students = new List<(int Id, string Name)> { (1, "Ali"), (2, "Veli") };
var grades = new List<(int Id, int Grade)> { (1, 90), (2, 85) };
var result = students.Join(grades,
    s => s.Id,
    g => g.Id,
    (s, g) => (s.Name, g.Grade));
// Nəticə: (Ali, 90), (Veli, 85)
```

14. `GroupJoin` operatoru nədir?

Cavab: `GroupJoin` iki kolleksiyanı birləşdirir, lakin ikinci kolleksiyanın elementlərini qruplar şəklində saxlayır. Məsələn:

```
var departments = new List<(int Id, string Name)> { (1, "IT"), (2, "HR") };
var employees = new List<(int DepId, string Name)> { (1, "Ali"), (1, "Veli"), (2, "Ayşe") };
var result = departments.GroupJoin(employees,
    d => d.Id,
    e => e.DepId,
    (d, e) => (d.Name, Employees: e.Select(x => x.Name)));
// Nəticə: (IT, [Ali, Veli]), (HR, [Ayşe])
```

15. `Any` operatoru nə iş görür?

Cavab: `Any` kolleksiyada müəyyən şərtə uyğun elementin olub-olmadığını yoxlayır. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
bool hasEven = numbers.Any(n => n % 2 == 0); // true (2 cütdür)
```

16. `All` operatoru nədir?

Cavab: `All` kolleksiyadakı bütün elementlərin müəyyən şərtə uyğun olub-olmadığını yoxlayır. Məsələn:

```
var numbers = new List<int> { 2, 4, 6 };
bool allEven = numbers.All(n => n % 2 == 0); // true
```

17. `Count` və `LongCount` operatorları nədir?

Cavab:

- `Count` kolleksiyadakı elementlərin sayını qaytarır (32-bit integer).
- `LongCount` böyük kolleksiyalar üçün istifadə olunur və 64-bit integer qaytarır. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
int count = numbers.Count(); // 3
```

18. `Sum`, `Min`, `Max`, `Average` operatorları nə iş görür?

Cavab:

- `Sum`: Kolleksiyadakı ədədlərin cəmini hesablayır.
- `Min`: Ən kiçik dəyəri tapır.
- `Max`: Ən böyük dəyəri tapır.
- `Average`: Orta dəyəri hesablayır. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4 };
var sum = numbers.Sum(); // 10
var min = numbers.Min(); // 1
var max = numbers.Max(); // 4
var avg = numbers.Average(); // 2.5
```

19. `First` və `FirstOrDefault` operatorları arasındakı fərq nədir?

Cavab:

- `First`: Şərtə uyğun ilk elementi qaytarır, əgər element yoxdursa istisna (exception) atır.
- `FirstOrDefault`: Şərtə uyğun ilk elementi qaytarır, əgər element yoxdursa null və ya default dəyər qaytarır. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
var first = numbers.First(n => n > 1); // 2
var firstOrDefault = numbers.FirstOrDefault(n => n > 5); // 0
```

20. `Single` və `SingleOrDefault` operatorları nədir?

Cavab:

- `Single`: Şərtə uyğun tam olaraq bir elementi qaytarır, əgər bir və ya çox element varsa istisna atır.
- `SingleOrDefault`: Şərtə uyğun bir elementi qaytarır, əgər element yoxdursa null/default qaytarır, birdən çox element varsa istisna atır.

21. `Take` və `Skip` operatorları nə iş görür?

Cavab:

- `Take`: Kolleksiyadan müəyyən sayda elementi alır.
- `Skip`: Müəyyən sayda elementi keçir. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
var take = numbers.Take(2); // 1, 2
var skip = numbers.Skip(2); // 3, 4, 5
```

22. `TakeWhile` və `SkipWhile` operatorları nədir?

Cavab:

- `TakeWhile`: Şərt doğru olduğu müddətcə elementləri alır.
- `SkipWhile`: Şərt doğru olduğu müddətcə elementləri keçir. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
var takeWhile = numbers.TakeWhile(n => n < 3); // 1, 2
var skipWhile = numbers.SkipWhile(n => n < 3); // 3, 4, 5
```

23. LINQ-da `IEnumerable` və `IQueryable` arasındakı fərq nədir?

Cavab:

- `IEnumerable`: Yaddaşda olan kolleksiyalar üzərində işləyir, bütün məlumatlar yaddaşa yüklənir.
- `IQueryable`: Verilənlər bazası kimi uzaq mənbələrdə sorğuları optimallaşdırır və yalnız lazım olan məlumatları çəkir.

24. LINQ-da `Deferred Execution` (Təxirə salınmış icra) nədir?

Cavab: LINQ sorğuları dərhal icra olunmur, yalnız nəticəyə ehtiyac olduqda (məsələn, `ToList()`, `Count()` və s.) icra edilir. Buna təxirə salınmış icra deyilir. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
var query = numbers.Where(n => n > 1); // Sorğu burada icra olunmur
var result = query.ToList(); // İcra burada baş verir
```

25. LINQ-da `Immediate Execution` nədir?

Cavab: Bəzi LINQ operatorları (`Count`, `Sum`, `ToList`, `ToArray` və s.) sorğunu dərhal icra edir. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
var count = numbers.Count(); // Dərhal icra olunur
```

26. LINQ sorğularını optimallaşdırmaq üçün hansı üsullardan istifadə olunur?

Cavab:

- Lazım olmayan məlumatları çəkməmək üçün `Select` ilə yalnız lazımi sahələri seçmək.
- `Take` və `Skip` ilə məlumat həcmini məhdudlaşdırmaq.
- Verilənlər bazası sorğularında `IQueryable` istifadə etmək.
- Təkrar sorğuları kəşləmək (`ToList` və ya `ToArray` ilə).

27. LINQ-da `Distinct` operatoru nədir?

Cavab: `Distinct` kolleksiyadan təkrarlanan elementləri çıxarır. Məsələn:

```
var numbers = new List<int> { 1, 2, 2, 3 };
var distinct = numbers.Distinct(); // 1, 2, 3
```

28. LINQ-da `Concat` və `Union` operatorları nədir?

Cavab:

- `Concat`: İki kolleksiyanı birləşdirir, təkrarlanan elementləri saxlayır.
- `Union`: İki kolleksiyanı birləşdirir, təkrarlanan elementləri çıxarır. Məsələn:

```
var list1 = new List<int> { 1, 2 };
```

```
var list2 = new List<int> { 2, 3 };
var concat = list1.Concat(list2); // 1, 2, 2, 3
var union = list1.Union(list2); // 1, 2, 3
```

29. LINQ-da `Except` operatoru nədir?

Cavab: `Except` bir kolleksiyadan digər kolleksiyada olan elementləri çıxarır. Məsələn:

```
var list1 = new List<int> { 1, 2, 3 };
var list2 = new List<int> { 2, 3, 4 };
var except = list1.Except(list2); // 1
```

30. LINQ-da `Intersect` operatoru nədir?

Cavab: `Intersect` iki kolleksiyanın kəsişməsini (ümumi elementlərini) qaytarır. Məsələn:

```
var list1 = new List<int> { 1, 2, 3 };
var list2 = new List<int> { 2, 3, 4 };
var intersect = list1.Intersect(list2); // 2, 3
```

31. LINQ-da `Zip` operatoru nədir?

Cavab: `Zip` iki kolleksiyanın elementlərini cüt-cüt birləşdirir. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
var words = new List<string> { "bir", "iki", "üç" };
var zipped = numbers.Zip(words, (n, w) => $"{n}: {w}");
// Nəticə: "1: bir", "2: iki", "3: üç"
```

32. LINQ-da anonim tiplər nədir?

Cavab: LINQ sorğularında `Select` ilə yeni obyektlər yaradılarkən anonim tiplər istifadə olunur. Məsələn:

```
var students = new List<(string Name, int Age)> { ("Ali", 20), ("Veli", 22) };
var result = students.Select(s => new { s.Name, AgeGroup = s.Age > 21 ? "Böyük" : "Kiçik"
});
```

33. LINQ-da `let` açar sözü nə üçün istifadə olunur?

Cavab: `let` Query Sintaksisində ara dəyişənlər yaratmaq üçün istifadə olunur. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
var query = from n in numbers
        let square = n * n
        where square > 2
        select square;
// Nəticə: 4, 9
```

34. LINQ-da `into` açar sözü nədir?

Cavab: `into` sorğunun nəticəsini yeni bir dəyişənə saxlamaq və ya qruplaşdırma nəticələri ilə işləmək üçün istifadə olunur. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4 };
var query = from n in numbers
        group n by n % 2 into g
        select new { Key = g.Key, Count = g.Count() };
// Nəticə: { Key = 0, Count = 2 }, { Key = 1, Count = 2 }
```

35. LINQ-da null dəyərlərlə işləmək necə aparılır?

Cavab: Null dəyərlərlə işləyərkən `?.` (null-conditional) operatoru və ya `FirstOrDefault`, `SingleOrDefault` kimi metodlar istifadə olunur. Məsələn:

```
var list = new List<string> { "Ali", null, "Veli" };
var nonNull = list.Where(x => x != null).ToList();
```

36. LINQ-da performans problemləri ilə bağlı hansı səhvlərə yol verilir?

Cavab:

- Böyük kolleksiyalarda təkrar sorğuların icra olunması.
- Lazımsız məlumatların yaddaşa yüklənməsi.
- Verilənlər bazası sorğularında `IQueryable` əvəzinə `IEnumerable` istifadəsi.

37. LINQ ilə SQL injection riski varmı?

Cavab: LINQ to SQL və ya LINQ to Entities istifadə edərkən SQL injection riski yoxdur, çünki LINQ sorğuları parametrized formada çevrilir. Lakin xam SQL sorğuları yazılarsa, risk ola bilər.

38. LINQ-da `AsEnumerable` və `AsQueryable` metodları nədir?

Cavab:

- `AsEnumerable`: `IQueryable`-ni `IEnumerable`-ə çevirir, yəni sorğu yaddaşda icra olunur.
- `AsQueryable`: Kolleksiyanı `IQueryable` kimi təqdim edir, verilənlər bazası ilə işləməyə imkan verir.

39. LINQ-da `Aggregate` operatoru nədir?

Cavab: `Aggregate` kolleksiyanın elementləri üzərində ardıcıl olaraq bir funksiya tətbiq edir. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4 };
var product = numbers.Aggregate((a, b) => a * b); // 1 * 2 * 3 * 4 = 24
```

40. LINQ-da `Reverse` operatoru nədir?

Cavab: `Reverse` kolleksiyanın elementlərinin sırasını tərsinə çevirir. Məsələn:

```
var numbers = new List<int> { 1, 2, 3 };
var reversed = numbers.Reverse(); // 3, 2, 1
```

41. LINQ-da `SequenceEqual` nədir?

Cavab: `SequenceEqual` iki kolleksiyanın eyni elementlərə və sıraya malik olub-olmadığını yoxlayır. Məsələn:

```
var list1 = new List<int> { 1, 2, 3 };
var list2 = new List<int> { 1, 2, 3 };
bool equal = list1.SequenceEqual(list2); // true
```

42. LINQ-da dinamik sorğular necə yazılır?

Cavab: Dinamik sorğular üçün `System.Linq.Dynamic.Core` kimi xarici kitabxanalar istifadə olunur və ya lambda ifadələri ilə dinamik şərtlər qurulur. Məsələn:

```
var numbers = new List<int> { 1, 2, 3, 4 };
var condition = 2;
var result = numbers.Where(n => n > condition);
```

43. LINQ ilə paralel sorğular necə icra olunur?

Cavab: Paralel sorğular üçün `AsParallel()` metodu istifadə olunur. Bu, PLINQ (Parallel LINQ) ilə çoxnüvəli emalı aktivləşdirir. Məsələn:

```
var numbers = Enumerable.Range(1, 1000);
var result = numbers.AsParallel().Where(n => n % 2 == 0).ToList();
```

44. LINQ-da `OfType` operatoru nədir?

Cavab: `OfType` kolleksiyadan yalnız müəyyən tipdə olan elementləri seçir. Məsələn:

```
var list = new List<object> { 1, "Ali", 2, true };
var numbers = list.OfType<int>(); // 1, 2
```

45. LINQ-da `Cast` operatoru nədir?

Cavab: `Cast` kolleksiyanın elementlərini müəyyən bir tipə çevirir. Məsələn:

```
var list = new List<object> { 1, 2, 3 };
var numbers = list.Cast<int>(); // 1, 2, 3
```

46. LINQ-da `DefaultIfEmpty` operatoru nədir?

Cavab: `DefaultIfEmpty` kolleksiya boşdursa, default dəyəri qaytarır. Məsələn:

```
var list = new List<int>();
var result = list.DefaultIfEmpty(0); // 0
```

47. LINQ ilə XML-dən məlumat çəkmək necə aparılır?

Cavab: LINQ to XML (`System.Xml.Linq`) istifadə olunur. Məsələn:

XElement xml =
XElement.Parse("<Students><Student><Name>Ali</Name></Student></Students>");
var names = xml.Elements("Student").Select(s => s.Element("Name").Value); // Ali

48. LINQ-da `ElementAt` və `ElementAtOrDefault` nədir?

Cavab:

- `ElementAt`: Kolleksiyadan müəyyən indeksdəki elementi qaytarır, əgər indeks yoxdursa istisna atır.
- `ElementAtOrDefault`: İndeksdəki elementi qaytarır, əgər indeks yoxdursa default dəyər qaytarır.

49. LINQ-da `Range` və `Repeat` metodları nədir?

Cavab:

`Range`: Müəyyən diapazonda ədədlər yaradır. Məsələn:
var numbers = Enumerable.Range(1, 5); // 1, 2, 3, 4, 5

-

`Repeat`: Müəyyən elementi müəyyən sayda təkrarlayır. Məsələn:
var repeated = Enumerable.Repeat("A", 3); // A, A, A

-

50. LINQ sorğularında səhvləri necə idarə etmək olar?

Cavab:

- `try-catch` blokları ilə istisnaları idarə etmək.
- `FirstOrDefault`, `SingleOrDefault` kimi metodlarla null yoxlamaları aparmaq.
- Sorğu nəticələrini kəşləməklə təkrar icradan qaçmaq.