Test Questions – Answers

Soru-1

Bu kodun SQL karşılığıyla ilgili doğru ifade nedir?

```
{
        var result = context.Employees
          .GroupBy(e => e.Department)
          .Select(g => new
          {
            Department = g.Key,
            MaxSalary = g.Max(e => e.Salary),
            AvgSalary = g.Average(e => e.Salary),
            TotalSalary = g.Sum(e => e.Salary),
            Count = g.Count()
          })
          .ToList();
}
```

A) GroupBy işlemi SQL tarafında yapılır.
B) GroupBy bellekte yapılır, tüm veriler önce çekilir.
C) Average ve Sum C# içinde hesaplanır.
D) MaxSalary C# içinde hesaplanır.

Çözüm:

context.Employees bir **DbSet<Employee>** olduğu için, LINQ sorgusu SQL'e çevrilmek üzere bir **expression tree** olarak hazırlanır. Bu aşamada sorgu henüz çalıştırılmaz ve veritabanına gitmez. Sorguyu tetikleyen .ToList() gibi bir method çağrısı yapıldığında Entity Framework, GroupBy, Max, Average, Sum ve Count işlemlerini SQL tarafına çevirir ve veritabanında çalıştırır. Bu sayede tüm gruplama ve hesaplamalar SQL'de yapılır, C# sadece sonuçları alır.

Şıkların değerlendirmesi:

- **A) GroupBy işlemi SQL tarafında yapılır. - Doğru**
  GroupBy ifadesi DbSet üzerinde çalıştığı için SQL'de GROUP BY olarak çevrilir ve veritabanında uygulanır. Sorgu sadece .ToList() gibi bir tetikleyici method çağrıldığında çalıştırılır.
- **B) GroupBy bellekte yapılır, tüm veriler önce çekilir. - Yanlış**
  Bu ifade yalnızca sorgu belleğe alınırsa doğru olurdu. Örneğin ToList() veya AsEnumerable() çağrısı GroupBy'dan önce olsaydı, tüm veriler önce C# tarafına çekilir ve gruplama bellekte yapılırdı. Mevcut kodda gruplama SQL'de yapılır.
- **C) Average ve Sum C# içinde hesaplanır. - Yanlış**
  Average ve Sum SQL tarafında AVG() ve SUM() fonksiyonlarına çevrilir. C# tarafında herhangi bir hesaplama yapılmaz; sadece sonuçlar alınır.
- **D) MaxSalary C# içinde hesaplanır. - Yanlış**
  Max fonksiyonu SQL'de MAX() olarak çalışır ve C# tarafında sadece SQL'den dönen değer okunur.

Sonuç:
Doğru cevap **A)** ✅

SORU-2

Aşağıdaki kodun çıktısı nedir?

```
{
        var result = string.Join("-", Enumerable.Repeat("Hi", 3));
        Console.WriteLine(result);
}
```

A) HiHiHi
B) Hi-Hi-Hi
C) Hi Hi Hi
D) Hi,Hi,Hi

Çözüm:

Enumerable.Repeat("Hi", 3)
Bu method, belirtilen öğeyi ("Hi") belirli sayıda (3) tekrar eden bir koleksiyon (IEnumerable<string>) üretir. Yani bu ifade sonucunda ["Hi", "Hi", "Hi"] gibi bir dizim elde edilir.

string.Join("-", ...)
string.Join methodu, verilen koleksiyondaki öğeleri bir ayırıcı (separator) ile birleştirir. Buradaki ayırıcı "-" olduğu için, "Hi" öğeleri arasına “-“ konur ve sonuç olarak "Hi-Hi-Hi" stringi oluşturulur.

Console.WriteLine(result)
Oluşturulan sonucu konsola yazdırır. Bu durumda konsolda çıktı olarak **Hi-Hi-Hi** görünür.

Sonuç:
Doğru cevap B) ✅

SORU-3

Bu kodda IsPrime metodu C# içinde yazılmış özel bir metot. Kodun çalışmasıyla ilgili doğru ifade nedir?

```
{
        var query = context.Orders
          .Where(o => o.TotalAmount > 1000)
          .AsEnumerable()
          .Where(o => IsPrime(o.Id))
          .ToList();
}
```

A) Tüm filtreler SQL tarafında çalışır, performans çok yüksektir.
B) İlk Where SQL'de, ikinci Where belleğe alındıktan sonra çalışır.
C) Tüm Where filtreleri bellekte çalışır.
D) AsEnumerable sorguyu hızlandırır, hepsi SQL tarafında çalış

Çözüm:

1. **context.Orders.Where(o => o.TotalAmount > 1000)**
    - context.Orders bir **DbSet<Order>** olduğu için LINQ sorgusu SQL'e çevrilebilir.
    - İlk Where SQL tarafına çevrilir ve veritabanında filtreleme yapılır (TotalAmount > 1000).
2. **.AsEnumerable()**
    - Bu method, **IQueryable**'ı **IEnumerable**'a çevirir.
    - Yani bundan sonra yapılan LINQ işlemleri artık **C# tarafında bellekte çalışır**.
    - SQL tarafında bundan sonrası işlenmez.
3. **.Where(o => IsPrime(o.Id))**
    - IsPrime metodu **C# içinde yazılmış özel bir fonksiyon**.
    - SQL bunu anlayamaz, bu yüzden bu filtre **veriler belleğe alındıktan sonra C# tarafında uygulanır**.
4. **.ToList()**

    Bu method, filtrelenmiş veriyi C# tarafında bir listeye çevirir ve sorguyu sonlandırır.

Şıkların değerlendirmesi:

- **Tüm filtreler SQL tarafında çalışır, performans çok yüksektir. ❌ Yanlış**
    - İkinci Where SQL tarafında çalışmaz, çünkü IsPrime SQL tarafından anlaşılamaz.
- **İlk Where SQL'de, ikinci Where belleğe alındıktan sonra çalışır. ✅ Doğru**
    - Tam olarak bu şekilde çalışır: İlk filtre SQL'de, ikinci filtre C# tarafında uygulanır.
- **Tüm Where filtreleri bellekte çalışır. ❌ Yanlış**
    - Sadece AsEnumerable() sonrası filtreler bellekte çalışır. İlk Where SQL'de çalışır.
- **AsEnumerable sorguyu hızlandırır, hepsi SQL tarafında çalışır. ❌ Yanlış**
    - AsEnumerable tam tersi; sorguyu C# tarafına getirir ve SQL'de çalışmayı durdurur.
- **Sonuç:**
  Doğru cevap: **B)** ✅

Soru-4

Kod çalıştırıldığında hangi durum/sonuç gerçekleşir?

```
{
    using (var context = new AppDbContext())
          {
             var departments = context.Departments
                .Include(d => d.Employees)
                .AsSplitQuery()
                .AsNoTracking()
                .Where(d => d.Employees.Count > 5)
                .ToList();
          }
}
```

A) Tüm Department kayıtları tek bir SQL sorgusu ile, JOIN kullanılarak getirilir. EF Core değişiklik izleme yapar.
B) Department ve Employee verileri iki ayrı SQL sorgusu ile getirilir, EF Core değişiklik izleme yapmaz.
C) Department ve Employee verileri ayrı sorgularla getirilir, ancak EF Core değişiklik izleme yapar.
D) Tüm veriler tek sorguda getirilir ve değişiklik izleme yapılmaz.

Çözüm:

1. **Include(d => d.Employees)**
   - Department kayıtlarıyla birlikte, her bir Department'ın Employees koleksiyonu da yüklenecek.
2. **AsSplitQuery()**
   - **EF Core'da Include kullanıldığında varsayılan davranış tek sorgu + JOIN olabilir.**
   - Ancak **AsSplitQuery eklenirse, her ilişki için ayrı SQL sorgusu çalıştırılır.**
   - **Yani önce Department'lar, sonra Employees için ayrı sorgular yapılır.**
3. **AsNoTracking()**
   - **EF Core'un Change Tracker (değişiklik izleme mekanizması) devre dışı bırakılır.**
   - **Yani çekilen entity'ler üzerinde yapılan değişiklikler DbContext tarafından takip edilmez.**
4. **Where(d => d.Employees.Count > 5)**
   - **Employees.Count** ilişkili tabloya göre bir **subquery** üretir.
   - Yani SQL tarafında, her Department'ın Employees sayısı kontrol edilerek filtre uygulanır.
5. **ToList()**
   - Sorgu çalıştırılır ve sonuçlar belleğe alınır.

Şıkların Değerlendirmesi:

- **A**) Tüm Department kayıtları tek bir SQL sorgusu ile, JOIN kullanılarak getirilir. EF Core değişiklik izleme yapar. ❌ **Yanlış**
    - **Çünkü AsSplitQuery var, JOIN kullanılmaz.**
    - **Ayrıca AsNoTracking var, yani değişiklik izleme yapılmaz.**
- **B) Department ve Employee verileri iki ayrı SQL sorgusu ile getirilir, EF Core değişiklik izleme yapmaz. ✅ Doğru**
    - **AsSplitQuery → iki ayrı sorgu.**
    - **AsNoTracking → değişiklik izleme yok.**
- **C)** Department ve Employee verileri ayrı sorgularla getirilir, ancak EF Core değişiklik izleme yapar. ❌ **Yanlış**
    - **AsSplitQuery kısmı doğru, ama AsNoTracking yüzünden EF Core değişiklik izlemez.**
- **D)** Tüm veriler tek sorguda getirilir ve değişiklik izleme yapılmaz. ❌ **Yanlış**
    - **Tek sorguda değil, iki ayrı sorguda çalışır.**
- **Doğru cevap: B) ✅**

SORU-5

Aşağıdaki kodun çıktısı nedir?

```
{
        var result = string.Format("{1} {0}", "Hello", "World");
        Console.WriteLine(result);
}
```

A) "{0} {1} "
B) "Hello World"
C) "World Hello"
D) "HelloWorld"

Çözüm:

- **string.Format methodu,** verilen format string içindeki placeholder ({0}, {1} gibi) alanlarını sırasıyla parametrelerle doldurur.
- Burada {0} ilk parametreyi, {1} ise ikinci parametreyi temsil eder.
- Kodda parametreler şu şekilde:
    - **{0} → "Hello"**
    - **{1} → "World"**
- **Format string "{1} {0}" olduğu için çıktı: "World Hello" olur.**

Sonuç:
Doğru cevap : C) ✅

SORU-6

Aşağıdakilerden hangisi System.Linq.Enumerable ve System.Linq.Queryable arasındaki farktır?

A) Enumerable metodları yalnızca IQueryable üzerinde çalışır
B) Enumerable metodları IEnumerable üzerinde çalışır, Queryable metodları Expression Tree ile sorgu üretir
C) Enumerable metodları SQL veritabanına sorgu gönderir
D) Queryable metodları yalnızca string koleksiyonları üzerinde çalışır

System.Linq.Enumerable

- IEnumerable<T> koleksiyonlar üzerinde çalışır.
- LINQ sorgularını in-memory (bellek içinde) çalıştırır.
- Yani veriler önce belleğe alınır, sonra filtreleme ve diğer işlemler yapılır.
- Genellikle List, Array gibi koleksiyonlarda kullanılır.

System.Linq.Queryable

- IQueryable<T> koleksiyonlar üzerinde çalışır.
- LINQ sorgularını Expression Tree şeklinde temsil eder ve uygun sağlayıcı (ör. Entity Framework) bunu SQL gibi başka sorgulara dönüştürür.
- Yani filtreleme/veri çekme doğrudan veri kaynağında (örneğin SQL Server) yapılabilir.

Özet:

- **Enumerable → IEnumerable + Bellekte çalışır**
- **Queryable → IQueryable + Expression Tree + SQL gibi veri kaynağına çevrilebilir.**

Şıkların Değerlendirmesi:

- A) Enumerable metodları yalnızca IQueryable üzerinde çalışır ✕ Yanlış
  - Tam tersi: Enumerable metodları IEnumerable üzerinde çalışır.
- B) Enumerable metodları IEnumerable üzerinde çalışır, Queryable metodları Expression Tree ile sorgu üretir ☑ Doğru
  - Enumerable → Bellekteki koleksiyonlar (Array, List, IEnumerable).
  - Queryable → IQueryable üzerinden çalışır ve Expression Tree oluşturur.
  - Bu sayede Queryable ile yazılan LINQ sorguları SQL gibi başka veri kaynaklarına çevrilebilir.
- C) Enumerable metodları SQL veritabanına sorgu gönderir ✕ Yanlış
  - Enumerable tamamen in-memory (bellek içi) çalışır. SQL sorgusu üretmez.
- D) Queryable metodları yalnızca string koleksiyonları üzerinde çalışır ✕ Yanlış
  - Queryable string ile sınırlı değil, IQueryable<T> olan tüm veri kaynaklarında çalışabilir (ör. Entity Framework DbSet).

Doğru cevap: B) ☑

SORU-7

Aşağıdaki kodun çıktısı nedir?

```
{
        var people = new List<Person>{
          new Person("Ali", 35),
          new Person("Ayşe", 25),
          new Person("Mehmet", 40)
        };
        var names = people.Where(p => p.Age > 30)
                  .Select(p => p.Name)
                  .OrderByDescending(n => n);

        Console.WriteLine(string.Join(",", names));
}
```

A) Ali,Mehmet
B) Mehmet,Ali
C) Ayşe,Ali,Mehmet
D) Ali

Çözüm:

1. Veri kaynağı:
people adında bir liste oluşturuluyor ve türü **List<Person>.** Listeye üç adet **Person** nesnesi ekleniyor:
• **Ali (35 yaşında)**
• **Ayşe (25 yaşında)**
• **Mehmet (40 yaşında)**

2. Where(p => p.Age > 30) filtresi:

- Yaşı 30'dan büyük olanlar alınır.
- Ali (35) ✅
- Ayşe (25) ❌
- Mehmet (40) ✅
- **Sonuç: [Ali, Mehmet]**

3. Select(p => p.Name) seçimi:

- Sadece isimler alınır.
- **Sonuç: [Ali, Mehmet]**

4. OrderByDescending(n => n) sıralaması:

- İsimler alfabetik olarak azalan (Z → A) sıralanır.
- **"Mehmet" > "Ali" olduğu için sıralama: [Mehmet, Ali]**

5. string.Join(",", names) çıktısı:

- Liste elemanları araya , konarak birleştirilir.

30 yaş üstü olanlar **Ali** ve **Mehmet** seçilir, isimler alınır ve tersten sıralanır → **Mehmet,Ali**.
Doğru cevap: B ✅

SORU-8

Aşağıdaki kodun çıktısı nedir?

```
{
        var numbers = new List<int>{1,2,3,4,5,6};
        var sb = new StringBuilder();
        numbers.Where(n => n % 2 == 0)
                .Select(n => n * n)
                .ToList()
                .ForEach(n => sb.Append(n + "-"));

        Console.WriteLine(sb.ToString().TrimEnd('-'));
}
```

A) 4-16-36
B) 2-4-6
C) 1-4-9-16-25-36
D) 4-16-36-

Çözüm:

StringBuilder

- Değiştirilebilir (mutable) string oluşturmak için kullanılan bir sınıftır.
- Normal string'ler gibi her değişiklikte yeni bir nesne oluşturmaz, bu sayede performans avantajı sağlar.

Append

- Mevcut StringBuilder içeriğine yeni bir değer ekler.
- Eklenen değer string, karakter veya başka bir veri türü olabilir.

1. Where(n => n % 2 == 0)

- Çift sayılar seçilir: 2, 4, 6

2. Select(n => n * n)

- Seçilen sayılar karesine yükseltilir: 4, 16, 36

3. ForEach(n => sb.Append(n + "-"))

- Her sayıyı StringBuilder'a ekler ve araya - koyar: "4-16-36-"

4. TrimEnd('-')

- Sonundaki fazla - karakteri kaldırılır → "4-16-36"

Doğru cevap: B ☑

SORU-9

System.Text.Json ve System.Collections.Generic kullanılarak bir listeyi JSON'a dönüştürmek ve ardından deserialize etmek için doğru işlem sırası nedir?

A) Listeyi serialize et → JSON string oluştur → Deserialize → liste
B) Listeyi deserialize et → JSON string oluştur → liste
C) JSON string oluştur → liste → serialize
D) JSON string parse → ToString()

Çözüm:

Serialize (Serileştirme):

- Bir nesneyi veya koleksiyonu düz bir veri formatına (ör. JSON, XML, byte dizisi) dönüştürme işlemidir.
- Amaç, veriyi dosya, veritabanı veya ağ üzerinden taşımak veya saklamaktır.

Deserialize (Deserileştirme):

- Serialize edilmiş veriyi tekrar orijinal nesne veya koleksiyon hâline getirme işlemidir.
- JSON string gibi taşınmış veri, tekrar C# nesnesi olarak kullanılabilir.

Doğru işlem mantığı:

1. **Listeyi JSON'a dönüştürmek (Serialize):**
    - System.Text.Json.JsonSerializer.Serialize() kullanılarak bir liste JSON string'e çevrilir.
2. **JSON string'i tekrar listeye dönüştürmek (Deserialize):**
    - System.Text.Json.JsonSerializer.Deserialize<T>() ile JSON string tekrar liste haline getirilir.

Şıkların değerlendirmesi:

- **A) Listeyi serialize et → JSON string oluştur → Deserialize → liste ✅ Doğru**
    - İşlem sırası doğru ve mantıklıdır.
- **B) Listeyi deserialize et → JSON string oluştur → liste ❌ Yanlış**
    - Önce deserialize yapılamaz, çünkü liste JSON string'e çevrilmeden önce mevcut değildir.
- **C) JSON string oluştur → liste → serialize ❌ Yanlış**
    - İşlem sırası karışık; önce liste olmalı, sonra JSON'a çevrilir.
- **D) JSON string parse → ToString() ❌ Yanlış**
    - Parse ve ToString() JSON'dan liste oluşturmak için yeterli değildir.

Doğru cevap: A) ✅

SORU-10

Aşağıdaki kodda trackedEntitites değeri kaç olur?

```
{
        var products = context.Products
           .AsNoTracking()
           .Where(p => p.Price > 100)
           .Select(p => new { p.Id, p.Name, p.Price })
           .ToList();

        products[0].Name = "Updated Name";

        var trackedEntities = context.ChangeTracker.Entries().Count();
}
```

A) 0
B) 1
C) Ürün sayısı kadar
D) EF Core hata fırlatır

Çözüm:

1. **AsNoTracking()**
   Bu metod kullanıldığı için EF Core, sorgulanan entity'leri Change Tracker içine eklemez. Yani context tarafında herhangi bir izleme yapılmaz.
2. **Select(p => new { ... })**
   Burada bir anonim tip oluşturuluyor. Bu tip EF Core entity'si değildir, sadece C# tarafında geçici olarak üretilmiş bir nesnedir.
3. **products[0].Name = "Updated Name"**
   Bu satırda sadece liste içindeki anonim nesnenin Name property'si değiştirilir. EF Core bu değişikliği takip etmez, çünkü nesne hem anonim tiptir hem de AsNoTracking devreye girmiştir.
4. **context.ChangeTracker.Entries().Count()**
   Bu ifade, EF Core'un şu an izlediği entity sayısını döndürür. Ancak hiçbir entity takip edilmediği için sonuç 0 olur.

Doğru cevap: A) ✅

SORU-11

Hangisi doğrudur?

```
{
        var departments = context.Departments
          .Include(d => d.Employees)
            .ThenInclude(e => e.Projects)
          .AsSplitQuery()
          .OrderBy(d => d.Name)
          .Skip(2)
          .Take(3)
          .ToList();
}
```

A) Her include ilişkisi ayrı sorgu olarak çalışır, Skip/Take her sorguya uygulanır.
B) Skip/Take sadece ana tabloya uygulanır, ilişkilerde tüm kayıtlar gelir.
C) Skip/Take hem ana tablo hem ilişkili tablolara uygulanır.
D) AsSplitQuery performansı düşürür, tek sorgu ile çalışır

Çözüm:

- Include & ThenInclude → Department → Employees → Projects ilişkilerini dahil eder.
- AsSplitQuery() → EF Core her include için ayrı SQL sorgusu üretir. Böylece karmaşık JOIN yerine birden fazla sorgu çalışır.
- Skip(2).Take(3) → Bu ifade sadece ana tabloya (Departments) uygulanır. Yani 2 departman atlanır, sonraki 3 departman seçilir.
- İlişkili tablolar (Employees, Projects) → Skip/Take bunlara uygulanmaz; seçilen departmanların tüm ilişkili kayıtları gelir.

Şıkların Değerlendirmesi:

A) Her include ilişkisi ayrı sorgu olarak çalışır, Skip/Take her sorguya uygulanır ✗

AsSplitQuery ayrı sorgu çalıştırır doğru ✓ ama Skip/Take sadece **ana tabloya** uygulanır, diğerlerine değil ✗.

B) Skip/Take sadece ana tabloya uygulanır, ilişkilerde tüm kayıtlar gelir ✓

Doğru ifade budur. EF Core sadece Departments için Skip/Take uygular, Employees ve Projects için gelen ilişkiler filtrelenmez.

C) Skip/Take hem ana tablo hem ilişkili tablolara uygulanır ✗

Yanlış, çünkü ilişkilerde EF Core tüm kayıtları getirir.

D) AsSplitQuery performansı düşürür, tek sorgu ile çalışır ✗

Yanlış, çünkü AsSplitQuery tek sorgu değil **çoklu sorgu** üretir. Ayrıca performans senaryoya göre değişir.

Doğru cevap: B) ✓

SORU-12

Bu kodun sonucu ile ilgili doğru ifade hangisidir?

```
{
        var query = context.Customers
          .GroupJoin(
            context.Orders,
            c => c.Id,
            o => o.CustomerId,
            (c, orders) => new { Customer = c, Orders = orders }
          )
          .SelectMany(co => co.Orders.DefaultIfEmpty(),
            (co, order) => new
            {
              CustomerName = co.Customer.Name,
              OrderId = order != null ? order.Id : (int?)null
            })
          .ToList();
}
```

A) Sadece siparişi olan müşteriler listelenir.
B) Siparişi olmayan müşteriler de listelenir, OrderId null olur.
C) Sadece siparişi olmayan müşteriler listelenir.
D) GroupJoin SQL tarafında çalışmaz, tüm veriler belleğe alınır

1. **GroupJoin**
   - Customers ile Orders tabloları c.Id == o.CustomerId üzerinden birleştiriliyor.
   - Her müşteri için ilgili siparişler Orders koleksiyonunda tutuluyor.
2. **SelectMany + DefaultIfEmpty()**
   - DefaultIfEmpty() sayesinde siparişi olmayan müşteriler için orders boş değil, null değer döndürüyor.
   - Yani bu yapı Left Outer Join davranışı sergiliyor.
3. **OrderId = order != null ? order.Id : (int?)null**
   - Eğer sipariş varsa OrderId gerçek değeri alır.
   - Sipariş yoksa OrderId null olur.

Şıkların Değerlendirmesi

- **A) Sadece siparişi olan müşteriler listelenir** ❌
  - Yanlış. Çünkü DefaultIfEmpty() sayesinde siparişi olmayanlar da gelir.
- **B) Siparişi olmayan müşteriler de listelenir, OrderId null olur** ✅
  - Doğru. Bu tam olarak Left Join davranışıdır.
- **C) Sadece siparişi olmayan müşteriler listelenir** ❌
  - Yanlış. Siparişi olanlar da listelenir.
- **D) GroupJoin SQL tarafında çalışmaz, tüm veriler belleğe alınır** ❌
  - Yanlış. EF Core bunu SQL tarafında sorguya dönüştürür (LEFT JOIN).

Doğru cevap: B) ✅

SORU-13

Bu kodun SQL karşılığı ile ilgili hangisi doğrudur?

```
{
        var names = context.Employees
          .Where(e => EF.Functions.Like(e.Name, "A%"))
          .Select(e => e.Name)
          .Distinct()
          .Count();

}
```

A) EF.Functions.Like SQL tarafında çalışır, Distinct ve Count SQL tarafında yapılır.
B) EF.Functions.Like SQL tarafında çalışır, Distinct ve Count bellekte yapılır.
C) Tüm işlemler bellekte yapılır.
D) EF.Functions.Like sadece C# tarafında çalışır

Çözüm:

1. EF.Functions.Like(e.Name, "A%")
    - Bu metot EF Core'un sağladığı SQL LIKE karşılığıdır.
    - SQL tarafına "WHERE Name LIKE 'A%'" şeklinde yansır.
2. Select(e => e.Name)
    - Yalnızca Name kolonunu seçer.
3. Distinct()
    - SQL tarafına "SELECT DISTINCT [Name]" olarak çevrilir.
4. Count()
    - SQL'de "COUNT(*)" olarak çalışır.

Yani tüm işlemler EF Core tarafından tek bir SQL sorgusu olarak gönderilir.

Şıkların değerlendirilmesi

- **A) EF.Functions.Like SQL tarafında çalışır, Distinct ve Count SQL tarafında yapılır** ✅
    - Evet, doğru olan budur. Hepsi SQL tarafında çalışır.
- **B) EF.Functions.Like SQL tarafında çalışır, Distinct ve Count bellekte yapılır** ❌
    - Yanlış, çünkü Distinct ve Count da SQL'e çevrilir.
- **C) Tüm işlemler bellekte yapılır** ❌
    - Yanlış, çünkü EF.Functions.Like ve diğerleri SQL'e çevriliyor.
- **D) EF.Functions.Like sadece C# tarafında çalışır** ❌
    - Yanlış, EF.Functions.Like doğrudan SQL LIKE fonksiyonuna karşılık gelir.

Doğru cevap: A) ✅

SORU-14

Hangisi doğrudur?

```
{
        var result = context.Orders
           .Include(o => o.Customer)
           .Select(o => new { o.Id, o.Customer.Name })
           .ToList();
}
```

A) Include bu senaryoda gereksizdir, EF Core sadece Select ile ilgili alanları çeker.
B) Include gereklidir, yoksa Customer.Name gelmez.
C) Include ile Customer tüm kolonları gelir, Select bunu filtreler.
D) Select Include'dan önce çalışır.

Çözüm:

EF Core'da Select ifadesi kullanıp sadece ihtiyacımız olan alanları seçtiğimizde, Include çalışmaz çünkü Include sadece tam navigasyon nesneleri yüklenecekse anlamlıdır. Projection (anonim tip, DTO vb.) yapıldığında Include göz ardı edilir.

A) Include bu senaryoda gereksizdir, EF Core sadece Select ile ilgili alanları çeker. ✅

B) Select ile projection yapıldığında Include çalışmaz, Customer.Name zaten SQL join ile gelir.

C) Include çalışmadığı için tüm kolonlar gelmez.

D) Select SQL'e çevrilir, Include göz ardı edilir, sıralama söz konusu değil.

Doğru cevap: A) ✅

SORU-15

Hangisi doğrudur?

```
{
        var query = context.Employees
          .Join(context.Departments,
            e => e.DepartmentId,
            d => d.Id,
            (e, d) => new { e, d })
          .AsEnumerable()
          .Where(x => x.e.Name.Length > 5)
          .ToList();
}
```

A) Join ve Length kontrolü SQL tarafında yapılır.
B) Join SQL'de yapılır, Name.Length kontrolü belleğe alındıktan sonra yapılır.
C) Tüm işlemler SQL tarafında yapılır.
D) Join bellekte yapılır

Çözüm:

1. Join

- Employees tablosu ile Departments tablosu SQL tarafında INNER JOIN yapılır.
- Burada EF Core SQL sorgusu üretir.

2. .AsEnumerable()

- Bu kısımdan sonra artık sorgu SQL tarafında çalışmaz.
- SQL'den gelen veriler belleğe (C# tarafına) alınır.

3. .Where(x => x.e.Name.Length > 5)

- Name.Length > 5 kontrolü bellekte yapılır.
- Yani filtreleme SQL tarafında değil, program tarafında gerçekleşir.

4 .ToList()

- Sonuç listesi bellekte oluşturulur.

Şıkların Değerlendirilmesi;

A) Join ve Length kontrolü SQL tarafında yapılır. ✗

- Join SQL'de yapılır → doğru.
- Ama Length > 5 bellekte yapılır → yanlış.

B) Join SQL'de yapılır, Name.Length kontrolü belleğe alındıktan sonra yapılır. ✓

- Aynen kodun çalışma mantığı bu. Doğru cevap.

C) Tüm işlemler SQL tarafında yapılır. ✗

- .AsEnumerable() olduğu için Length > 5 SQL'de çalışmaz.

D) Join bellekte yapılır. ✗

- Join, .AsEnumerable()'den önce yazıldığı için SQL tarafında yapılır.

Doğru cevap: B) ✓ **EMİR ADIYAMAN**