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.

context.employees kısmında veritabanında ki employees tablosuna erişilir
GroupBy(e=> e.department) ile departmanlarına göre gruplandırılır

*B şıkkı olmaz çünkü,
GroupBy bellekte değil SQL de çalışıyor

*C şıkkı olmaz çünkü Average ve Sum SQL tarafında hesaplanır

*D şıkkı olmaz çünkü MaxSalary SQL de hesaplanır

Dolayısıyla doğru şık A EF, GroupBy ve diğer agregasyonları SQL e çevirebilir.

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

2.cevap

Enumerable.Repeat("Hi", 3)
bu metot hi değerini 3 kez tekrarlar

string.Join("-", [...])
bu metot da verilen değerleri birleştirir ve aralarına "-" koyar.

dolayısıyla cevap B şıkkı olur Hİ-Hİ-Hİ

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

3.cevap

*A şıkkı YANLIŞ çünkü,
-Is prime gibi özel C# fonksiyonları SQL e çevrilemez
-AsEnumerable() sonrası işlemler artık SQL de çalışmaz

*C şıkkı yanlış çünkü,
-ilk where sql de çalışıyor sadece 2. where bellekte yani genelleme hatalı

*D şıkkı yanlış çünkü,
-AsEnumerable()sorguyu hızlandırmaz aksine perfomansı düşürebilir
-Sonrasında yapılan işlemler artık sql tarafınad değil C# tarafında yapılır.

Dolayısıyla doğru şık B İlk Where SQL'de, ikinci Where belleğe alındıktan sonra çalışır.

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.

4.cevap
-AsSplitQuery Deartmens ve Employees tek joın sorgusu yerine ayrı ayrı sql sorguları olarak çalıştırır
- AsNoTracking EF Core bu değişiklikleri takip etmez.
- .Where(d => d.Employees.Count > 5) bu kod satırı 5'ten fazla çalışanı olan departmanları filtreler.

* A şıkkı yanlış çünkü,
- AsSplitQuery olduğu için veriler tek sorguyla değil 2 sorguyla gelir.
-AsNoTracking nedeniyle değişiklikler takip edilmez.

*C şıkkı yanlış çünkü,
- AsNoTracking değişiklik izleme yapmaz

*D şıkkı yanlış çünkü,
-AsSplitQuery olduğu için tek sorguda değil 2 sorguda getirilir.

Dolayısıyla doğru cevap B şıkkı.

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"

5.cevap
-string.format metodu belirtilen biçime göre string birleştirme işlemi yapar.
-{0}: ilk parametre olan "Hello" değerini temsil eder. {1}: ikinci parametre olan "World" değerini temsil eder.

*A şıkkı yanlış çünkü,
- bu literal sting olduğu haliyle yazılırdı string.format çalımşıyor gibi davranıyor

*B şıkkı yanlış çünkü
-{0} {1} sırasıyla kullanılmış olsaydı doğru olurdu.

*D şıkkı yanlış çünkü,
-boşluk bile yok. Eğer "{0}{1}" olsaydı bu olurdu.

Dolayısıyla doğru cevap B şıkkı.

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

6.cevap

*A şıkkı yanlış çünkü,
-Enumerable metotları IEnumerable üzerinde çalışır, IQueryable değil.
*C şıkkı yanlış çünkü,
-Enumerable metotları bellekte çalışır, veritabanına SQL göndermez.
-SQL'e sadece Queryable metotları dönüştürebilir.
*D şıkkı yanlış çünkü,
-Queryable, IQueryable<T> olan her şeyde çalışabilir. Sadece string değil, int, Product, Order, vs.

Dolayısıyla doğru cevap B şıkkı.

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

-Where(p => p.Age > 30) yaşı 30dan büyük olanları listeye alır
-Select(p => p.Name) listeye alınanların sadece adını listeye ekler
-OrderByDescending(n => n) liste tersten sıralanır

*A şıkkı neden yanlış
-Ali,Mehmet artan sıralama olsa olurdu yani OrderBy ama soruda OrderByDesvending yazıldığı için olmaz

*B şıkkı neden yanlış
-Ayşenin yaşı 25 olduğu için where filtresine takılır

*D şıkkı neden yanlış
-sadece Ali değil mehmette olduğundan

Dolayısıyla cevap B şıkkı.

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-

8.cevap
-Where(n => n % 2 == 0) burda çift sayılar listeleniyor
-Select(n => n * n) burda listelenen sayıların karesi alınıyor
-ToList().ForEach(n => sb.Append(n + "-")) her sayı StringBuilder a "-" olarak ekleniyor
*4-
*16-
*36-
-TrimEnd('-') ile sonda ki - kaldırılır.

ve sonuç: 4-16-32 olurç
---
dolayısıyla cevap A şıkkı.

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()

9.cevap
-ilk olarak listeyi steralize ederiz
-bir json string oluşturulur
-daha json stringi deserialize ederiz
-son olarak tekrar listeye çevririz

*B şıkkı yanlış çünkü,
-listeyi deserialize etmek için zaten bir json string olması gerek

*C şıkkı yanlış çünkü,
-önce json string nası oluşacak liste olmadan json oluşturalamaz

*D şıkkı yanlış çünkü,
-bu ifade zaten anlamsız ToString sadece yazdırır.

Dolayısıyla cevap A şıkkı.

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

10.cevap
*.Select(p => new { p.Id, p.Name, p.Price })

-bu kod parçası anonymous type oluşturur. yani product sınıfı döndürülmez, sadece veriler kopyalanır.

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

-bu kod bloğu ise anonim nesne üzerinde ki name değerini değiştirir.
-ancak anonim nesneler zaten EF tarafından izlenmiyor.
-ayrıca AsNoTracking kullanıldığı için context hiçbir entity'yi takip etmiyordur.

*context.ChangeTracker.Entries().Count();
-EF core sadece izlenen entityleri ChangeTracker ile listeler.
-bu örnekte izlenen hiç bir entity yoktur çünkü AsNoTracking kullanılmıştır ve Select ile anonymous object oluşturulmuştur.

*B şıkkı yanlış çünkü,
-sadece 1 entity izlense bile 1 olurdu ama bu örnekte hiç izlenmiyor.

*C şıkkı yanlış çünkü,
-AsNoTracking yüzünden EF hiçbiriyle ilgilenmez.

*D şıkkı yanlış çünkü,
bu kod geçerli ve hata vermez

Dolayısıyla doğru cevap A şıkkı.

D) EF Core hata fırlatır

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

11.cevap

*A şıkkı yanlış çünkü,
-AsSplitQuery doğru her include için ayrı sorgu olarak çalışır
-ama skip/take her sorguya uygulanmaz yalnızca ana tabloya uygulanır.

*C şıkkı yanlış çünkü,
-Sadece ana tabloya Departments uygulanır.
-ilişkili tablolara Employees ve Project uygulanmaz.

*D şıkkı yanlış çünkü,
-AsSplitQuery çoklu sorgu ile çalışır.
-performansı düşürmez hatta çok ilişkili tablolarda arttırabilir.
---
dolayısıyla cevap B şıkkı.

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();
}
```

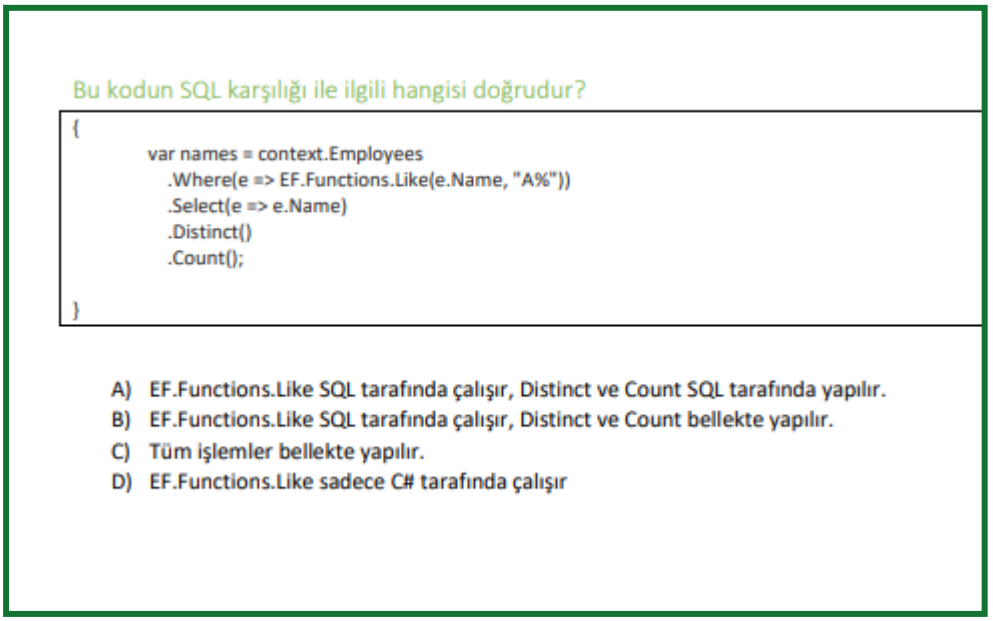
A) Sadece siparişi olan müşteriler listelenir.
B) Siparişi olmayan müşteriler de listelenir, OrderId null olur.
C) Sadece siparişi olmayan müşteriler listelenir.
D) GroupJoin SQL tarafında çalışmaz, tüm veriler belleğe alınır

12.cevap

*A şıkkı yanlış çünkü,
-DefaultEmpty sayesinde siparişi olmayan müşterilerde listelenir

*C şıkkı yanlış çünkü,
-siparişi olan ve olmayan tüm müşteriler listelenir

*D şıkkı yanlış çünkü
-EF gibi orm lerde groupjoin genellikle sql e çevrilir ve veritabanı tarafında çlışır bu nedenle performans açısından kritik olan yerlerde genellikle doğru sorguya çevrilir.
---
dolayısıyla doğru cevap B şıkkı.



13.cevap

*B şıkkı yanlış çünkü,
-distinct ve count da sql e çevrilebilir bu ifade sql tarafında çalışır belleğe alınmaz

*C şıkkı yanlış çünkü,
-bu kodun hiçbir kısmı bellekte yapılmak zorunda değildir ef bu işlemleri sql e çevirir ve veritabanında çalıştırır

*D şıkkı yanlış çünkü,
-EF.Functions.Like zaten SQL LIKE ifadesine karşılık gelir Amaç SQL tarafında çalışmasıdır.
---
dolayısıyla doğru cevap A şıkkı.

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.

14.cevap

*B şıkkı yanlış çünkü
-Include gerekmez çünkü Select içinde Customer.Name açıkça istenmiş. EF Core bunu otomatik olarak JOIN ile çeker.

*C şıkkı yanlış çünkü,
-Include burada çalışmaz çünkü Select yapıldığında Include yok sayılır.

*D şıkkı yanlış çünkü,
-Bu ifadede sıradan ziyade öncelik değil, hangisinin etkili olduğu önemlidir
---
dolayısıyla doğru cevap A şıkkı.

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

15.cevap
*A şıkkı yanlış çünkü,
Join SQL'de yapılır, ama Name.Length > 5 bellekte yapılır.

*C şıkkı yanlış çünkü,
-AsEnumerable() nedeniyle Where kısmı SQL dışına çıkar.

*D şıkkı yanlış çünkü,
-Join işlemi, AsEnumerable()'den önce olduğundan SQL tarafında gerçekleşir.
---
dolayısıyla doğru cevap A şıkkı.