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.

Bu sorguda GroupBy, Max, Average, Sum, Count gibi fonksiyonlar Entity Framework tarafından SQL ifadelerine çevrilip doğrudan veritabanında çalıştırılır. Yani gruplama ve agregasyon işlemleri belleğe çekilmeden, SQL tarafında yapılır.

Bu yüzden doğru **cevap A.**

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

Cevap: B) Hi-Hi-Hi

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

Cevap: B) İlk Where SQL'de çalışır; AsEnumerable() sonrası ikinci Where bellek tarafında (IsPrime C# metodu olduğu için).

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.

Cevap:B) Department ve Employee verileri ayrı SQL sorguları ile gelir (AsSplitQuery), ayrıca AsNoTracking nedeniyle değişiklik izleme yapılmaz.

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"

Cevap: C) World Hello

{1} → "World", {0} → "Hello"; arada boşluk var → World Hello

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

Cevap:B) Enumerable metodları IEnumerable üzerinde çalışır; Queryable metodları Expression Tree üretip sağlayıcıya (ör. EF) çevirtir.

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

Cevap:B) Mehmet, Ali

Mantık: Yaş > 30 ⇒ **Ali (35)** ve **Mehmet (40)** kalır.
İsimler **alfabetik olarak azalan** sıralanır ⇒ **Mehmet, Ali**

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-

Cevap: A) 4-16-36

Çiftleri al → karesini hesapla → "-" ile birleştir → TrimEnd('-') son tireyi siler.

System.Text.Json ve System.Collections.Generic kullanılarak bir listeyi JSON'a dönüştürmek ve ardından deseralize 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()

Cevap: A) Listeyi serialize et → JSON string oluşur → Deserialize → liste

JsonSerializer.Serialize(list) ile JSON'a çevir, sonra JsonSerializer.Deserialize<List<T>>(json) ile geri liste yap.

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

Cevap: A) 0

AsNoTracking() ile getirilenler izlenmez; ayrıca Select(new { ... }) zaten entity değil anonim tip döndürüyor. Bu yüzden ChangeTracker içinde kayıt olmaz → **0**.

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

Cevap:B) Skip/Take sadece ana tabloya uygulanır; ilişkililerde seçilen satırların tüm ilişkili kayıtları gelir.

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

Cevap: B) Siparişi olmayan müşteriler de listelenir; OrderId null olur.
Kısa mantık: GroupJoin + DefaultIfEmpty() ⇒ left outer join etkisi.

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

Cevap: A)

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.

Cevap:A) sadece seçili alanlar çekilir.

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

Cevap: B) Join SQL'de yapılır; AsEnumerable() sonrası Name.Length > 5 filtresi bellekte çalışır.