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.

1) Context objesini DbContext olarak varsayarsak eğer, bu işlem SQL tarafında gerçekleşeceği için cevap **A şıkkıdır**.

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) Enumerable.Repeat(“Hi”, 3) methoduyla beraber içerisinde üç tane “Hi” içeren bir koleksiyon oluşacaktır. Join methoduyla beraber her birinin arasında ‘-’ string ifadesi eklenecektir. Bu yüzden cevap **B şıkkıdır**.

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) DbContext'in objesi üzerinde LINQ operasyonu yaparaken IQueryable döner. Bu yüzden bu ifadelerin methodları Expression Tree üreterek SQL'de çalışır. Ama bu kod içerisinde AsEnumerable() kullanılarak ifade IEnumerable'a dönüştüğünden ikinci Where ifadesi bellekte çalışır. Bu yüzden cevap **B şıkkıdır**.

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) AsSplitQuery() sayesinde, SQL tarafında her iki tablo için ayrı sorgular atılarak performans sağlanır. AsNoTracking() sayesinde de Entity Framework izleme yapmaz yani gelen veri sadece ReadOnly gibi olur, bu da performans için olumlu yönde etki yapar. Bu yüzden cevap **B şıkkıdır**.

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) Süslü parantez içerisinde bulunan sayılar 'Format()' methodundaki birinci parametreden sonraki parametrelin sırasını ifade etmektedir. Bu yüzden cevap **C şıkkıdır.**

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 **B şıkkıdır.**

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

7) İlk Where ifadesiyle yaşları 30'dan büyük olan kişiler filtrelenir. Ardından gelen Select ifadesiyle Person sınıfın Name alanı projekte edilir, yani artık ifade String koleksiyonu halindedir. En son OrderByDescending ifadesiyle isimler azalan formatında alfabetik olarak sıralanır. Bu yüzden cevap **B şıkkıdır**.

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) İlk Where ifadesiyle çift sayılar filtrelenir ve Select ile filtrelenen sayılar kendisiyle çarpılarak yeni sayılar elde edilip List haline getirilir. ForEach ile birlikte oluşturulmuş olan StringBuilder içerisine Liste içerisindeki her bir elamana '-' ifadesi eklenir. En son konsola print edilirken en sondaki '-' ifadesi TrimEnd ile çıkartılır. Bu yüzden cevap **A şıkkıdır.**

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()

9) Önce JsonSerializer sınıfın Serialize() methodu kullanılar bir listeden JSON String oluşturulur. Tam tersi işlem ise Deserialize'dır. Bu yüzden cevap **A şıkkıdır**.

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) Product Entity'i için AsNoTracking kullanılmış. Bu yüzden bu entity üzerinde olacak herhangi bir değişiklik takip edilmeyecektir.
O yüzden context.ChangeTracker.Entries().Count() işleminin sonucu 0 çıkacaktır, cevap **A şıkkıdır.**

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) Skip ve Take işlemi ana tablo olan Departments'a uygulanır. SplitQuery sayesinde Employees ve Projects için ayrı sorgular oluşturulur. Filtrelenmiş Departmanslarla ilişkili olan verilir çekilir. Bu yüzden cevap **B şıkkıdır.**

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) GroupJoin ile birlikte Customers ile Orders tablosu customerId üzerinden join işlemi yapılır ve anonim tipte olmak üzere her bir Customer ve o Customer'ın ilişkili olduğu Orders listesi oluşturulur. SelectMany ile Customer ve Orders Listesini içeren objenin eğer Order listesi boş ise Empty olarak alınır ve her bir müşteri ve onların her bir siparişi liste haline çevirilir. Bu işlem yapılırken siparişi olmayan müşterlierlin orderId değeri null olur. Bu yüzden cevap **B şıkkıdır**.

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

13) Buradaki işlem bir veri tabanı context'i üzerinde yapılıyor ve tüm LINQ işlemleri Iqueryable üzerinde yapılacağı için Entity Framework tarafından SQL'e çevirelecektir. Ayrıca EF.Functions.Like methodu direkt Entity Framework tarafından SQL için var olan method, SQL'de ki LIKE'a denk geliyor. Bu yüzden cevap **A şıkkıdır.**

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) Select işleminde Orders Entity ile ilişkile olan Customer Entity'nin Name alanı çekiliyor. Böyle bir işlemde Select arka planda ilgili Customer objesini SQL tarafında Join yaparak alıyor. Include ile Orders objesinde bulunan tüm Customer sütunlarını çekmek performans açısından kötü olur. Bu yüzden Include gereksizdir, cevap **A şıkkıdır**.

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) AsEnumerable() kısmına kadar LINQ işlemi IQueryable üzerinde yapılıyor. Bu yüzden SQL'de yapılacaktır. Ondan sonraki Where filtresi ise IEnumrable üzerinde olacağı için bellektedir. Bu yüzden cevap **B şıkkıdır**.