CRUD операции с базата данни

```
static void Main(string[] args)
{
   using (var db = new PhoneBookContext())
   {
      // Create
      var phone = new Phone { Name = "John", Family = "Doe",
                                          PhoneNumber = "1234567890", IsDeleted = false };
      db.Phones.Add(phone);
      db.SaveChanges();

      // Read
      var phones = db.Phones.ToList();
      foreach (var p in phones)
      {
         Console.WriteLine($"{p.Name} {p.Family} - {p.PhoneNumber}");
      }

      // Update
      phone.Name = "Jane";
      db.SaveChanges();

      // Delete
      db.Phones.Remove(phone);
      db.SaveChanges();
   }
}
```

Супер! А има ли някакви практики за използване на LINQ методи директно при комуникацията към базата данни, така че да не се извлича цялата таблица а само конкретни данни (за да не се създава излишен трафик). Опиши ми най-често използваните такива методи

Да, определено има добри практики за ефективно използване на LINQ с Entity Framework за оптимизиране на достъпа до данните и намаляване на ненужния трафик между приложението и базата данни. Ключът е да използваш LINQ методи, които формират SQL заявки, изпълнявани директно от базата данни. Това означава, че данните се филтрират на ниво база данни, а не в паметта на приложението. Ето някои често използвани LINQ методи и практики:

1. Използване на `Where` за филтриране

Филтрирай записите в базата данни с условие, което се изпълнява на страната на сървъра.

csharp Копиране на код

```csharp
var filteredPhones = db.Phones.Where(p => p.Family == "Doe").ToList();
```

2. Използване на `Select` за извличане на конкретни полета

Извличай само полетата, които са ти необходими, вместо цели обекти.

csharp Копиране на код

```csharp
var phoneNumbers = db.Phones.Where(p => p.Family == "Doe").Select(p => p.PhoneNumber).ToList();
```

3. Използване на `FirstOrDefault` или `SingleOrDefault`

Използвай тези методи, когато искаш да намериш един запис, като изразът се оценява на страната на сървъра.

csharp Копиране на код

```csharp
var phone = db.Phones.FirstOrDefault(p => p.Id == 5);
```

4. Ограничаване на броя записи с `Take`

Ограничи броя на върнатите записи, особено полезно при пагинация.

csharp Копиране на код

```csharp
var limitedPhones = db.Phones.Take(10).ToList();
```

5. Прескачане на записи с `Skip`

Прескочи определен брой записи, полезно за пагинация.

csharp Копиране на код

```csharp
var paginatedPhones = db.Phones.OrderBy(p => p.Id).Skip(20).Take(10).ToList();
```

6. Използване на `OrderBy` и `OrderByDescending`

Сортирай записите преди да ги извлечеш.

csharp Копиране на код

```csharp
var sortedPhones = db.Phones.OrderBy(p => p.Family).ToList();
```

7. Обединяване на методи

Често е ефективно да комбинираш няколко LINQ метода за постигане на сложни заявки.

csharp Копиране на код

```csharp
var specificPhones = db.Phones
                      .Where(p => p.Family == "Doe" && p.IsDeleted == false)
                      .OrderBy(p => p.Name)
                      .Select(p => new { p.Name, p.PhoneNumber })
                      .Take(5)
                      .ToList();
```

Използването на тези методи може значително да намали количеството данни, които се трансферират от базата данни към приложението, като същевременно осигурява гъвкавост при обработката на данни. Важно е да тестваш и профилираш заявките си, за да си сигурен, че те са оптимизирани за производителност.