الـ Navigational Properties في Entity Framework

إيه هي الـ Navigational Properties؟

لو بتشتغل على Entity Framework (EF) في .NET، أكيد سمعت عن الـ Navigational Properties. دول هما الخصائص اللي بتسمحلك تنتقل بين الكيانات (Entities) المرتبطة في قاعدة البيانات. يعني، بدل ما تعمل Joins يدوية زي في SQL العادي، EF بيخليك تكتب كود أبسط وأقرب للـ OOP.

في البداية، Entity Framework هو ORM (Object-Relational Mapping) بيربط بين كودك الـ C# وقاعدة البيانات. والـ Navigational Properties هي جزء أساسي من ده، زي روابط افتراضية (Virtual Properties) في الكلاسات اللي بتمثل الجداول.

مثال بسيط: لو عندك كلاس Student وكلاس Course، وكل طالب ممكن يكون له كورسات كتير (One-to-Many)، هتكون الـ Navigational Property زي ICollection<Course> Courses في كلاس Student. ده بيخليك تقدر تقول student.Courses وتوصل لكل الكورسات اللي تابعة للطالب ده.

1. إزاي بتعمل الـ Navigational Properties؟

الـ Navigational Properties مش بس خصائص عادية؛ هي بتعتمد على العلاقات (Relationships) اللي بتحددها في الـ Model. في EF، بتقدر تعرفها بطريقتين:

- **Convention**: الـ EF بيفهم العلاقات تلقائيًا لو سميت الخصائص صح (زي StudentId كـ Foreign Key).
- **Fluent API**: لو عايز تخصيص أكتر، تستخدم OnModelCreating في الـ DbContext عشان تحدد الـ HasOne/HasMany/WithOne/WithMany.

مثال كود في C#:

```
public class Student {
    public int Id { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Course> Courses { get; set; } //
Navigational Property
}

public class Course {
    public int Id { get; set; }
    public string Title { get; set; }
    public int StudentId { get; set; } // Foreign Key
    public virtual Student Student { get; set; } // Navigational
Property العكسية
}
```

هنا، Courses هي Navigational Property للوصول من الطالب للكورسات، وStudent للوصول العكسي.

الكلمة virtual مهمة هنا عشان تفعل الـ Lazy Loading (تحميل البيانات لاحقًا لما تحتاجها).

2. أهمية وجود الـ Navigational Properties في الـ Queries

الـ Navigational Properties هي السر في كتابة Queries فعالة وسهلة في EF.
بدونها، هتضطر تعمل كل حاجة يدويًا، وده بيأثر على الأداء والكود.

- 🔹 **لو موجودة (الفوائد):**

- **تسهيل الـ LINQ Queries**: تقدر تكتب

context.Students.Include(s => s.Courses).Where(s => s.Name == "Ahmed")
عشان تحمل الطالب والكورسات معاه في استعلام واحد (Eager Loading). ده يمنع مشكلة N+1 Queries (يعني استعلام رئيسي + استعلام لكل عنصر مرتبط).

- **Lazy Loading**: لو مش عايز تحمل كل حاجة مرة واحدة، EF بيحمل البيانات تلقائيًا لما تقول student.Courses.Count() . ده بيوفر موارد لو البيانات كبيرة.
- **Explicit Loading**: تقدر تحكم في التحميل يدويًا بـ context.Entry(student).Collection(s => s.Courses).Load()
- **تحسين الأداء**: بتقلل عدد الـ Round Trips لقاعدة البيانات، وبتخليك تستخدم Projection (زي Select) عشان تجيب بس اللي تحتاجه.
- **دعم العلاقات المعقدة**: زي Many-to-Many، حيث EF بيعمل جدول وسيط تلقائيًا ويخليك تنقل بسهولة.

مثال Query مع Navigational Properties:

```
var students = context.Students
    .Include(s => s.Courses) // Eager Loading
    .ToList();
foreach (var student in students) {
    Console.WriteLine(student.Name + " has " + student.Courses.Count
+ " courses.");
}
```

بدونها، هتضطر تعمل Join يدوي زي from s in Students join c in Courses on s.Id equals c.StudentId، وده أطول وأقل مرونة.

🔹 **لو مش موجودة (العيوب):**

- **Queries يدوية معقدة**: هتعتمد على Joins خام في LINQ أو SQL، وده بيخلي الكود أصعب في الصيانة وأكثر عرضة للأخطاء.
- **مشاكل أداء**: بدون Navigational، مش هتقدر تستفيد من Lazy/Eager Loading، فممكن يحصل N+1 Problem (استعلامات كتير صغيرة بدل واحد كبير).

- **فقدان المرونة**: مش هتقدر تستخدم ميزات EF زي الـ Change Tracking أو الـ Cascade Delete بشكل كامل، لأن العلاقات مش محددة بوضوح.
- **زيادة التعقيد في الكود**: لو عندك علاقات كتير، هتضطر تكتب كود إضافي عشان تربط البيانات بعد الجلب، زي استخدام Loops أو Mapping يدوي.

في الختام، وجودها بيحسن الكفاءة والقراءة، بس لو مش موجودة، EF هيعمل زي ADO.NET القديم - فعال بس مش مريح.

3. نصائح مهمة للتعامل مع الـ Navigational Properties

✔️ **فعل Lazy Loading بحذر**: في EF Core، تحتاج تضيف UseLazyLoadingProxies() في الـ DbContext، بس انتبه لأنه ممكن يسبب Queries إضافية لو مش محتاجها.

✔️ **استخدم Eager Loading للأداء**: دايمًا استخدم Include أو ThenInclude للعلاقات المتداخلة عشان تقلل الـ Database Hits.

✔️ **تجنب Circular References**: لو في علاقات دائرية (زي Student يشير لـ Course وبالعكس)، استخدم Ignore في Fluent API عشان تتجنب مشاكل في الـ Serialization.

✔️ **اختبر الأداء**: استخدم أدوات زي EF Profiler عشان تشوف عدد الـ Queries وتحسنها.

✔️ **في EF Core vs EF6**: في EF Core، الـ Navigational Properties أقوى مع دعم أفضل لـ Value Objects و Owned Types.