Armen Melkumyan • 1st
Technical / Solutions Architect
1mo •

Debunking a Common EF Core Myth: Equals() vs. == in LINQ Queries

Recently, I came across a post suggesting that using .Equals() in EF Core LINQ queries causes inefficient execution—fetching all records into memory before applying the filter. According to that post, only == ensures correct SQL translation.

❌ This is not entirely correct—and it's time to debunk the myth!
EF Core is smart enough to translate Equals() to SQL just like ==, as long as the comparison remains simple.

Testing It Ourselves
To verify, I wrote a simple .NET console app with SQLite in-memory DB and logged the SQL translations for:
1) db.Dishes.Where(d => d.Name.Equals(dishName))
2) db.Dishes.Where(d => d.Name == dishName)

Attached images show the actual EF-generated SQL queries.
Both methods produce the same SQL:
SELECT "d"."Id", "d"."Name" FROM "Dishes" AS "d" WHERE "d"."Name" = 'Pizza'
Clearly, EF does not load all records into memory before filtering.

Key Ideas
Both .Equals() and == translate correctly to SQL in EF Core.
Potential Issue: .Equals() can throw an exception if dishName is null.

Best Practice: Use == for readability and null safety.
When Does .Equals() Cause Issues?
In older EF6 (not EF Core), .Equals() might not always translate well.
If using complex expressions, EF may struggle to convert .Equals() to SQL.

If dishName is null, .Equals(dishName) throws a NullReferenceException.

Final Thoughts

- Don’t take everything at face value—always verify with .ToQueryString().

- Understanding how EF Core works internally helps you write better-performing queries.

#EntityFramework #DotNet #Performance #EFCore #Csharp #SoftwareDevelopment #BestPractices