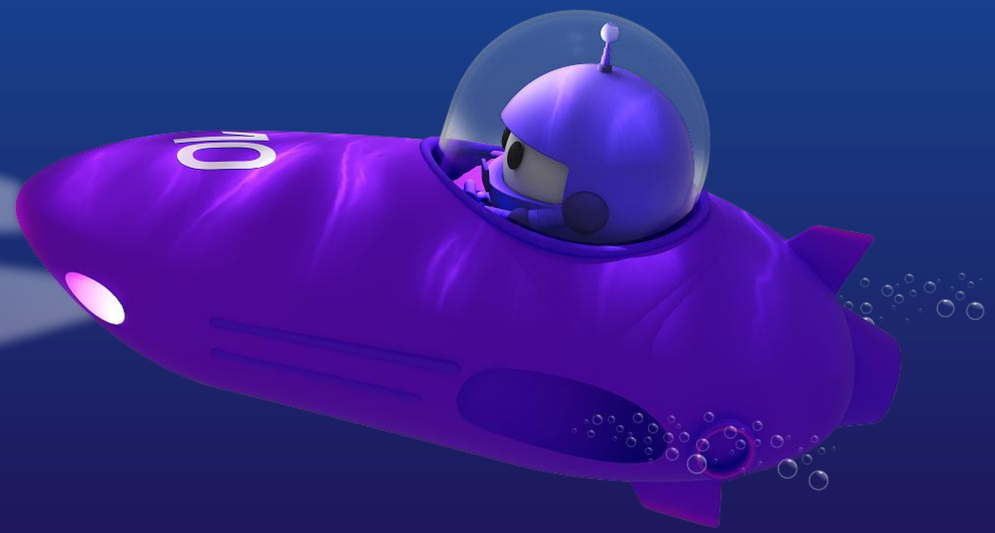
EF Core 10

.NET

Deep Dive into new Features

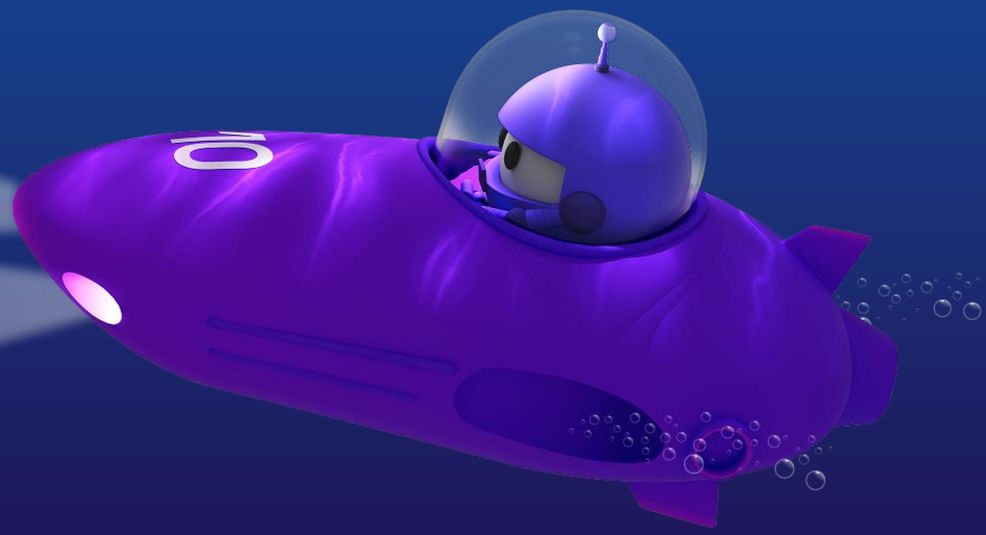


.NET

Sanjar Akhmedov

Software Engineer

Silk Road Professionals

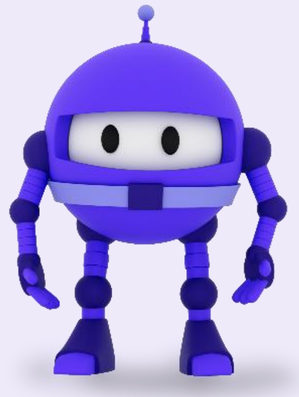


/Akhmedov_Sanjar /Akhmedov_Sanjar /AkhmedovSanjar

Agenda

- Introduction – What is EF Core?
- Complex Types
- JSON Type Support
- Vector Search
- LeftJoin & RightJoin
- Named Query Filters

What is Entity Framework Core?



C# Classes
.NET

EF Core
(ORM for .NET)

Database Tables
Maps classes to tables

No manual SQL

LINQ queries

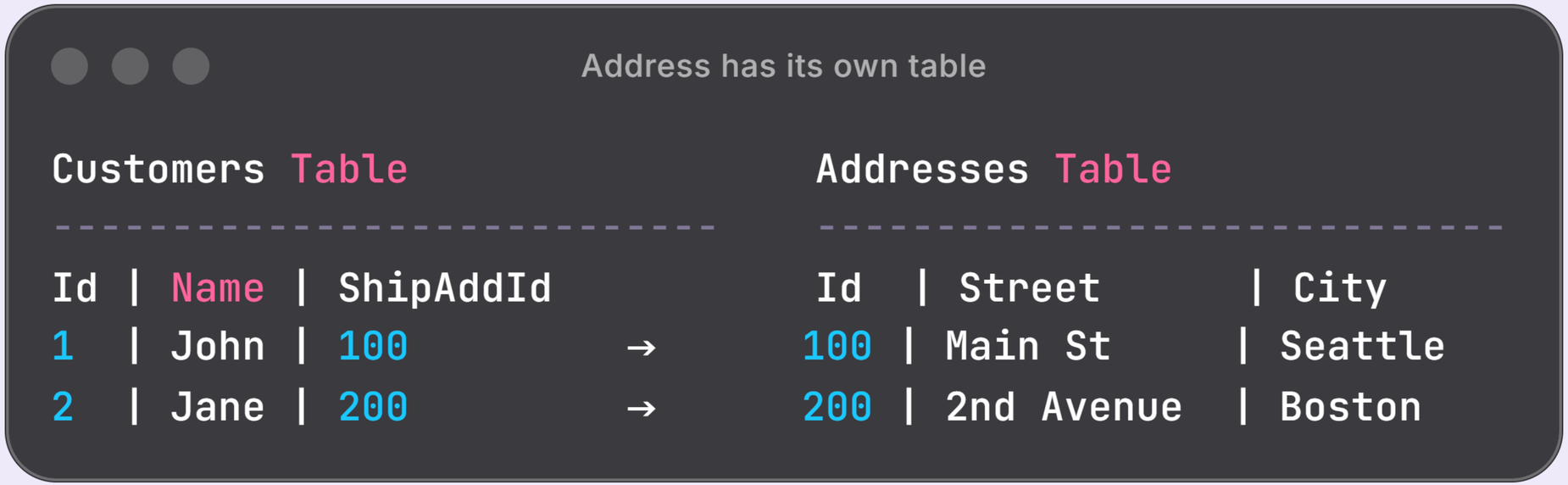
What are Complex Types?

Ways to Store Related Data (Ex: Customer and Address)

Address has its own table

```
public class Address {
    public int Id { get; set; }           // ← Has its own ID
    public string Street { get; set; }
    public string City { get; set; }
}

public class Customer {
    public int ShippingAddressId { get; set; }  // ← Links to Address
    public Address ShippingAddress { get; set; }
}
```



Address becomes extra columns in Customer table

```
// Address becomes columns
public record Address(string Street, string City, string ZipCode);

public class Customer {
    public Address ShippingAddress { get; set; }  // ← No separate table!
}

// Configuration
entity.ComplexProperty(c => c.ShippingAddress);
```

Address becomes extra columns in Customer table

```
Customers Table
-------------------------------------------------------------------------
Id | Name | ShippingAddress_Street | ShippingAddress_City | ShippingAddress_ZipCode
1  | John | 123 Main St            | Seattle              | 98101
2  | Jane | 456 Oak Ave            | Portland             | 97205
```

JSON Type Support

Address stored as JSON in one column

```
// Address stored as JSON
public record Address(string Street, string City, string ZipCode);

public class Customer {
    public Address ShippingAddress { get; set; }  // ← Stored as JSON
}

// Configuration
entity.ComplexProperty(c => c.ShippingAddress, b => b.ToJson());
```

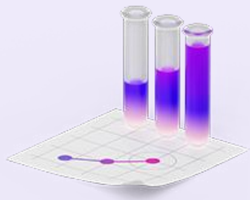
Address stored as JSON in one column

```
Customers Table
-----------------------------------------------------------------
Id | Name | ShippingAddress (JSON)
-----------------------------------------------------------------
1  | John | {"Street":"123 Main","City":"Seattle","Zip":"98101"}
2  | Jane | {"Street":"456 Oak","City":"Portland","Zip":"97201"}
```

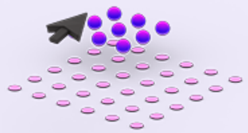
Performance Benchmark
by Filtering - Address.City with 100,000 Customers

Query Performance Comparison (Address.City)

Storage Method	Mean (ms)	StdDev (ms)	Allocated
Association (Separate Table / JOIN)	37.870	0.2845	4.03 MB
Complex Type (JSON Column)	19.834	0.2090	3.53 MB
Complex Type (Splitted Columns)	6.346	0.0926	3.50 MB



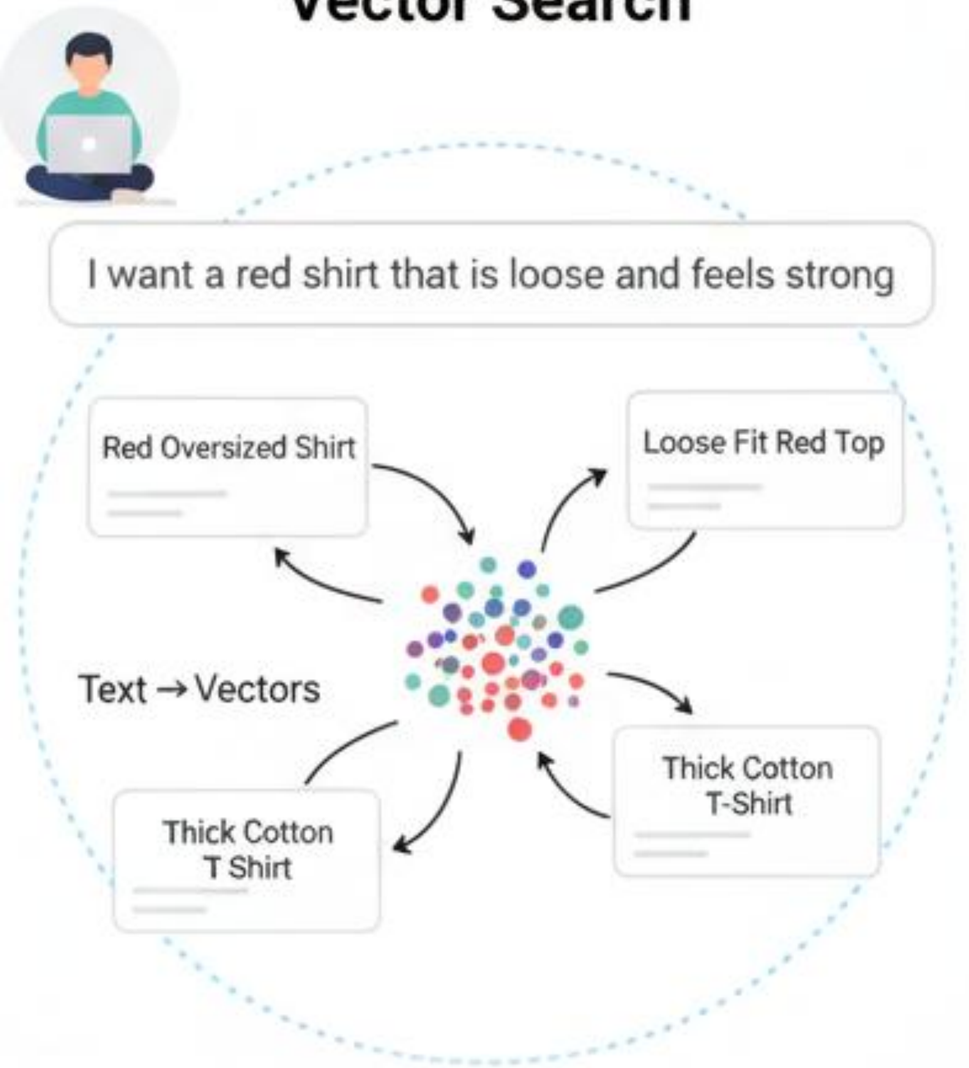
Vector Search Support



Keyword Search

Exact word match only

Vector Search



Finds related meaning

EF Core Product Model with Vector Search

```cs
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Category { get; set; }
    public string Description { get; set; }

    [Column(TypeName = "vector(1536)")]
    public SqlVector<float> SearchVector { get; set; }
}
```

Products table with Vector Search (Name, Category, Desc)

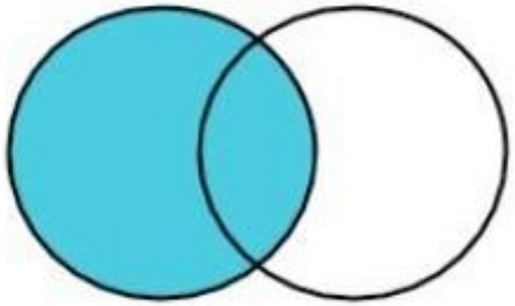
Product Name	Category	Description	Vector Search
Wireless Charging Pad	Accessories	Fast charging pad for smartphones	{0.12, 0.88, ...}
USB-C Charger 65W	Chargers	High power charger for laptops	{0.67, 0.21, ...}
Magnetic Phone Stand	Accessories	Desk stand with magnetic grip	{0.14, 0.81, ...}
Wireless Power Mat	Accessories	Slim mat for cable-free phone charg	{0.11, 0.86, ...}

Products table with Vector Search (Name, Category, Desc)

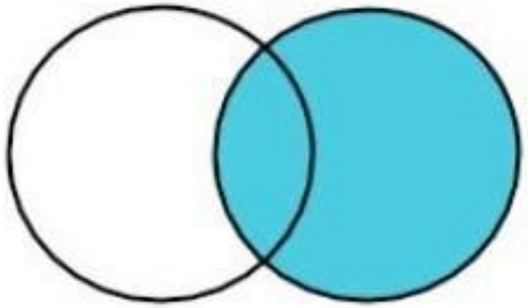
```
User Text (Ex:"Fast charging pad for smartphones")
   |
   v
Embedding Model
   | {0.13, 0.87, 0.46, ...}
   v
Query Vector
   |
   v
Compare Query Vector
   |
   v
Cosine / Distance Similarity
   |
   v
Search Results:
```

Product Name	Description	Similarity
Wireless Charging Pad	Fast charging pad for smartphones	0.97 ✓
Wireless Power Mat	Slim mat for cable-free phone charge	0.94 ✓
Magnetic Phone Stand	Desk stand with magnetic grip	0.61 ✓

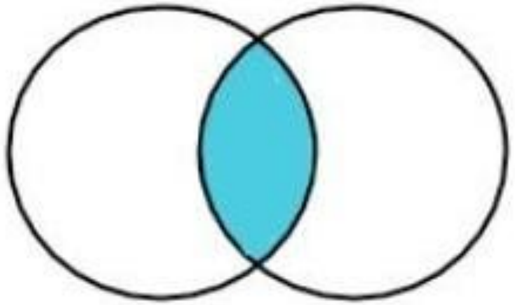
LeftJoin and RightJoin operators



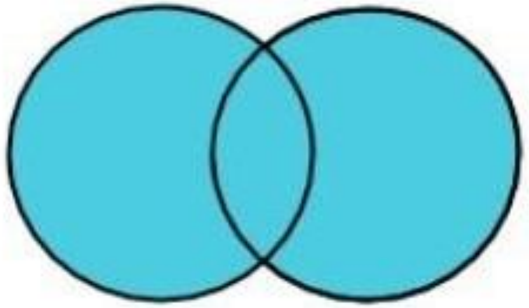
Left Join



Right Join

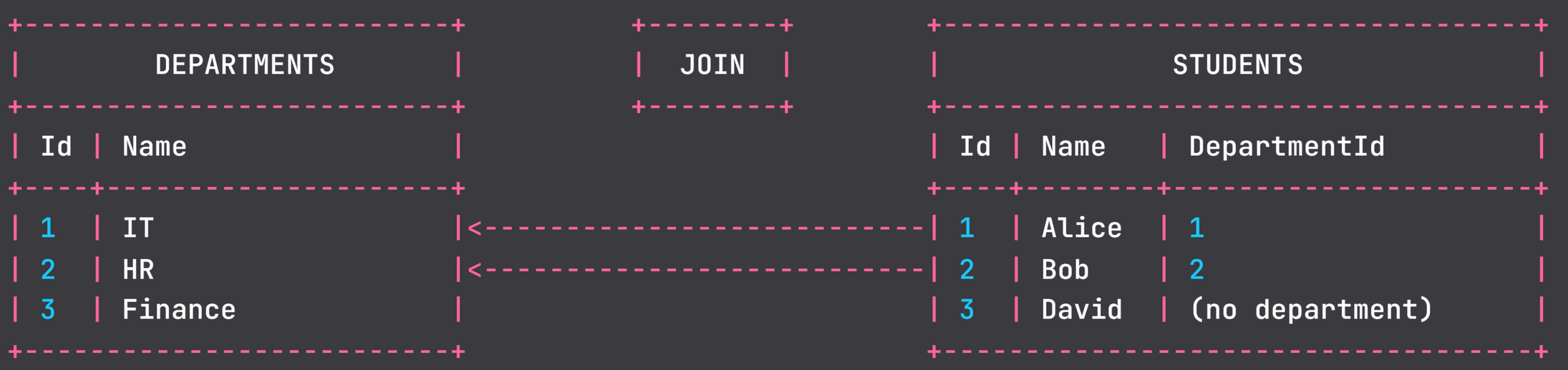


Inner Join



Full Outer Join

Database Joins



JOINED RESULT

StudentName	DepartmentName
Alice	IT
Bob	HR
David	(no department)

LeftJoin: GroupJoin + SelectMany + DefaultIfEmpty

```
var departments = new[]
{
    new Department { Name = "Computer Science" },
    new Department { Name = "Mathematics" },
    new Department { Name = "Physics" }
};
var students = new[]
{
    new Student { Name = "Alice", DepartmentId = departments[0].Id },
    new Student { Name = "Bob", DepartmentId = departments[0].Id },
    new Student { Name = "Charlie", DepartmentId = null },
    new Student { Name = "Diana", DepartmentId = departments[1].Id }
};

var leftJoin = await context.Students
    .GroupJoin(
        context.Departments,
        s => s.DepartmentId,
        d => d.Id,
        (student, departments) => new { student, departments })
    .SelectMany(
        x => x.departments.DefaultIfEmpty(),
        (x, department) => new
        {
            StudentName = x.student.Name,
            DepartmentName = department != null ? department.Name : "None"
        })
    .ToListAsync();
```

NEW WAY in EF CORE 10: LeftJoin & RightJoin

```
var leftJoin = await context.Students
    .LeftJoin(context.Departments, s => s.DepartmentId, d => d.Id,
        (s, d) => new
        {
            StudentName = s.Name,
            DepartmentName = d != null ? d.Name : "None"
        })
    .ToListAsync();

var rightJoin = await context.Departments
    .RightJoin(context.Students, d => d.Id, s => s.DepartmentId,
        (d, s) => new
        {
            DepartmentName = d != null ? d.Name : "No Department",
            StudentName = s.Name
        })
    .ToListAsync();
```

Left Join			Right Join	
Student Name	Department Name		Department Name	Student Name
Alice	Computer Science		Computer Science	Alice
Bob	Computer Science		Computer Science	Bob
Charlie	None		Mathematics	Diana
Diana	Mathematics		No Department	Charlie

Named Query Filters

What Are Named Query Filters?

Global filters that automatically apply WHERE clauses to **every query** for an entity, with the ability to selectively disable specific filters by name.

Named Query Filters

Before EF Core 10:

```
// Single unnamed filter - all or nothing
entity.HasQueryFilter(a => !a.IsDeleted && a.TenantId == currentTenant);

// Problem: Can't disable just soft delete, must disable everything
var allRecords = context.Accounts.IgnoreQueryFilters(); // ← Ignores BOTH
```

After EF Core 10:

```
// Multiple named filters - selective control
entity.HasQueryFilter("SoftDelete", a => !a.IsDeleted);
entity.HasQueryFilter("Tenant", a => a.TenantId == currentTenant);

// Can disable individually. Keep tenant filter!
var deletedRecords = context.Accounts.IgnoreQueryFilters(["SoftDelete"]);
```

Thank you

Demo repository