Advanced Querying

Advanced Entity Framework Core

SoftUni Team

Technical Trainers

Software University

https://about.softuni.bg/

Have a Question?

sli.do

#csharp-db

Table of Contents

Executing Native SQL Queries

Parameterless and Parameterized

Executing Native SQL Queries

- Executing a **native SQL query** in EF Core directly

```
var query = "SELECT * FROM Employees";
var employees = db.Employees
  .FromSqlRaw(query)
  .ToArray();
```

- Limitations
  - **JOIN** statements **don't** get mapped to the entity class
  - **Required columns** must **always** be selected
  - **Target table** must be the same as the **DbSet**

Native SQL Queries with Parameters

- Native SQL queries can also be parameterized

```
var context = new SoftUniDbContext();
string nativeSQLQuery =
  "SELECT FirstName, LastName, JobTitle" +
  "FROM dbo.Employees WHERE JobTitle = {0}";

var employees = context.Employees.FromSqlRaw(
  nativeSQLQuery, "Marketing Specialist");

foreach (var employee in employees)
{
  Console.WriteLine(employee.FirstName);
}
```

Parameter placeholder

Parameter value

Interpolation in SQL Queries

- **FromSqlInterpolated** allows string interpolation syntax

```
var context = new SoftUniDbContext();
string jobTitle = "Marketing Specialist";
FormattableString nativeSQLQuery =
  $"SELECT * FROM dbo.Employees WHERE JobTitle = {jobTitle}";
var employees = context.Employees.FromSqlInterpolated(
  nativeSQLQuery);

foreach (var employee in employees)
{
  Console.WriteLine(employee.FirstName);
}
```

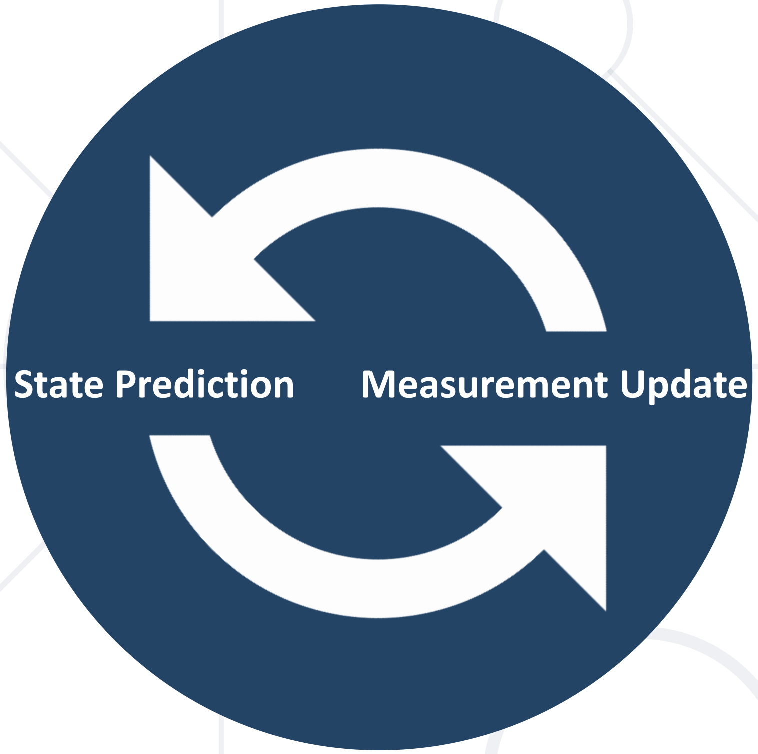
Interpolated parameter

Executing a Stored Procedure

- Stored Procedures can be executed via SQL

```sql
CREATE PROCEDURE UpdateSalary @param int
AS
UPDATE Employees SET Salary = Salary + @param;
```

```csharp
var salaryParameter = new SqlParameter("@salary", 5);
var query = "EXEC UpdateSalary @salary";
context.Database.ExecuteSqlRaw(query, salaryParameter);
```



Object State Tracking

Attached and Detached Objects

- In Entity Framework, objects can be
  - **Attached** to the object context (tracked object)
  - **Detached** from an object context (untracked object)
- Attached objects are tracked and managed by the **`DbContext`**
  - **`SaveChanges()`** persists all changes in DB
- Detached objects are not referenced by the **`DbContext`**
  - Behave like a normal objects, which are not related to EF
  - We can get detached objects using **`AsNoTracking()`**
  - No-tracking queries are quicker to execute

Tracking and No-tracking Queries

- **Tracking** queries

```
var employee = context.Employees
                      .FirstOrDefault(e => e.EmployeeId == 1);
employee.JobTitle = "Marketing Specialist";
context.SaveChanges();
```

Returns attached entry

- **No-tracking** queries

```
var employees = context.Employees
                       .AsNoTracking()
                       .ToList();
```

Returns detached read-only entity

Attached Objects

- When a query is executed inside a **`DbContext`**, the returned objects are **automatically attached** to it
- When a context is **destroyed**, all **objects** in it are automatically **detached**
  - e.g., in **Web applications** between requests
- You might later **attach** objects that have been previously **detached** to a **new context**

Detaching Objects

- When is an object detached?
  - When we get the object from a **DbContext** and then **Dispose** it

```
Employee GetEmployeeById(int id)
{
   using (var SoftUniDbContext = new SoftUniDbContext())
   {
     return SoftUniDbContext.Employees
       .First(e => e.EmployeeID == id);
   }
}
```

Returned employee is detached

  - Manually: by setting its **State** to **Detached**

Reattaching Objects

- When we want to update a detached object, we need to **reattach it** and then update it: change to **Attached** state

```
void UpdateName(Employee employee, string newName)
{
   using (var softUniDbContext = new SoftUniDbContext())
   {
      var entry = softUniDbContext.Entry(employee);
      entry.State = EntityState.Modified;
      employee.FirstName = newName;
      softUniDbContext.SaveChanges();
   }
}
```

BULK

Bulk Operations

Multiple Update and Delete in Single Query

Entity Framework Plus

- EF Core **does not** support bulk operations
- `Z.EntityFramework.Plus` gives you the ability to perform **bulk update/delete** of entities
- Entity Framework Plus

```
Install-Package Z.EntityFramework.Plus.EFCore
```

- Read more: https://entityframework-plus.net

Bulk Delete

- Delete all users where **FirstName** matches given string

```
context.Employees
  .Where(e => e.FirstName == "Pesho")
  .Delete();
```

```
DELETE [dbo].[Employees]
FROM [dbo].[Employees] AS j0 INNER JOIN (
SELECT
    [Extent1].[Id] AS [Id]
    FROM [dbo].[Employees] AS [Extent1].[Name]
    WHERE N'Pesho' = [Extent1].[Name]
) AS j1 ON (j0.[Id] = j1.[Id])
```

Bulk Update: Syntax

- Update all Employees with name "Niki" to "Stoyan"

```
context.Employees
  .Where(t => t.Name == "Niki")
  .Update(u => new Employee { Name = "Stoyan" });
```

- Update all Employees' age to 99 who have the name "Plamen"

```
IQueryable<Employee> employees = context.Employees
  .Where(employee => employee.Name == "Plamen");

employees.Update(employee => new Employee { Age = 99 });
```

Types of Loading

Lazy, Eager and Explicit Loading

Explicit Loading

- **Explicit loading** loads all records when they're needed
- Performed with the `.Reference().Load()` and `Collection().Load()` methods

```
var employee = context.Employees.First();

context.Entry(employee)
   .Reference(e => e.Department)
   .Load();

context.Entry(employee)
   .Collection(e => e.EmployeeProjects)
   .Load();
```

Eager Loading

- **Eager loading** loads **all related records** of an entity **at once**
- Performed with the `Include()` **and** `ThenInclude()` methods

```
context.Towns.Include("Employees");
```

```
context.Towns.Include(town => town.Employees);
```

```
context.Employees
   .Include(employee => employee.Address)
   .ThenInclude(address => address.Town)
```

Lazy Loading

- Lazy Loading **delays** loading of data until it is used
- EF Core enables lazy-loading for any navigation property that can be **overridden** (**virtual**)
- Offers better performance in certain cases
  - Less RAM usage
  - Smaller result sets returned
- Each loading of navigational property is an addition query (N+1)

Enable Lazy Loading Proxies

- Install Lazy Loading Proxies

```
Install-Package Microsoft.EntityFrameworkCore.Proxies
```

- Enable the package

```
void OnConfiguring (DbContextOptionsBuilder options)
{
  options
    .UseLazyLoadingProxies()
    .UseSqlServer(myConnectionString);
}
```

N+1 Problem

- Refreshing the article list page, sends 11 queries to the database
  - The **first query** finds the first 10 articles
  - The subsequent **10 queries**, find each article's comments
  - Total of 11 queries (N + 1)

Concurrency Checks

Optimistic Concurrency Control in EF

- EF Core runs in **optimistic concurrency** mode (no locking)
  - By default, the conflict resolution strategy in EF is "**last one wins**"
  - The last change overwrites all previous concurrent changes
- Enabling "**first wins**" strategy for certain property in EF
  - `[ConcurrencyCheck]`

Last One Wins – Example

```
var contextFirst = new SoftUniDbContext();
var lastProjectFirstUser = contextFirst.Projects.First();
lastProjectFirstUser.Name = "Changed by the First User";

// The second user changes the same record
var contextSecondUser = new SoftUniDbContext();
var lastProjectSecond = contextSecondUser.Projects.First();
lastProjectSecond.Name = "Changed by the Second User";

// Conflicting changes: last wins
contextFirst.SaveChanges();
contextSecondUser.SaveChanges();
```

Second user wins

First One Wins – Example

```
var context = new SoftUniDbContext();
var lastTownFirstUser = contextFirst.Towns.First();
lastTownFirstUser.Name = "First User";

var contextSecondUser = new SoftUniDbContext();
var lastTownSecondUser = contextSecondUser.Towns.First();
lastTownSecondUser.Name = "Second User";

context.SaveChanges();
contextSecondUser.SaveChanges();
```

[ConcurrencyCheck]

Changes get saved

DbUpdateConcurrencyException

cascade

Cascade Operations

Deleting Related Entities

Cascade Delete Scenarios

- **Required FK** with **cascade delete** set to **`true`**, **deletes everything** related to the deleted property
- **Required FK** with **cascade delete** set to **`false`**, **throws exception** (it cannot leave the navigational property with no value)
- **Optional FK** with **cascade delete** set to **`true`**, **deletes everything** related to the deleted property
- **Optional FK** with **cascade delete** set to **`false`**, **sets** the value of the **FK to NULL**

Cascade Delete with Fluent API (1)

- Using `OnDelete` with `DeleteBehavior` Enumeration:
  - `DeleteBehavior.Cascade`
    - Deletes related entities (default for required FK)
  - `DeleteBehavior.Restrict`
    - Throws exception on delete
  - `DeleteBehavior.ClientSetNull`
    - Default behavior for optional FK (does not affect database)
  - `DeleteBehavior.SetNull`
    - Sets the property to null (affects database)

Cascade Delete with Fluent API (2)

- Cascade delete syntax

```
modelBuilder.Entity<User>()
    .HasMany(u => u.Replies)
    .WithOne(a => a.Author)
    .OnDelete(DeleteBehavior.Restrict);
```

```
modelBuilder.Entity<User>()
    .HasMany(u => u.Replies)
    .WithOne(a => a.Author)
    .OnDelete(DeleteBehavior.Cascade);
```

Summary

- Databases can be accessed directly with **SQL queries** from C# code
- EF keeps track of the **model state**
- **Entity Framework-Plus** lets you bundle **update** and **delete** operations
- EF supports lazy, eager and explicit **loading**
- With multiple users, **concurrency** of operations must be observed
- **Cascade delete** is on by default

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License