Tapping into EF Core's Pipeline

Julie Lerman

Most Trusted Authority on Entity Framework Core

@JulieLerman www.thedatafarm.com

Overview

Overriding SaveChanges

Saving and tracking event handlers

Interceptors for database commands and more

Exploring ChangeTracker Entries for Overriding the SaveChanges Method

SaveChanges Method is Virtual

You can override SaveChanges in your DbContext class

Add logic just before or just after EF Core calls SaveChanges internally

Accessing the ChangeTracker EntityEntries

```
DbContext.ChangeTracker.Entries().ToList();
```

EF Core uses the info in these entries to construct its SQL command

```
public override int SaveChanges()
{
  //perform your custom logic on ChangeTracker.Entries() or other data
  return base.SaveChanges(); //sends data to database
}

public override int SaveChanges()
{
  //perform your custom logic on ChangeTracker.Entries() or other data
  int affected=base.SaveChanges(); //sends data to database
  //perform some other custom logic after DB calls
  return affected;
}
```

Override SaveChanges to Apply Custom Logic Around DB Call

Unless you want to completely override SaveChanges, call base.SaveChanges, which will send the commands to the database.

Please learn more than these fundamentals of ChangeTracker.Entries before altering them in your code

```
public override int SaveChanges()
{
  //perform your custom logic on ChangeTracker.Entries() or other data
  return base.SaveChanges(); //sends data to database
}
```

Override SaveChanges to Apply Custom Logic Around DB Call

Unless you want to completely override SaveChanges, call base.SaveChanges, which will send the commands to the database.

Updating Shadow Properties During SaveChanges

Row created or updated timestamps

User who created or modified the row

Useful for auditing reports against the database

Extraneous to business logic and in the way

Why Shadow Properties?

Store data that's irrelevant to your business logic

Examples:
Row created/updated timestamps

User who created/modified row

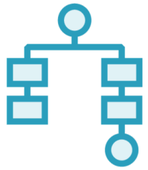
Useful for reports against the database such as auditing

Extraneous to business logic and would be in the way

Shadow Properties

Defined by the DbContext and persisted into database

Use migrations to get the new property into the database



Not part of your class so use the DbContext to set values

Just before SaveChanges builds SQL is a great time to set values

You can define a shadow property for all entities & update them, too

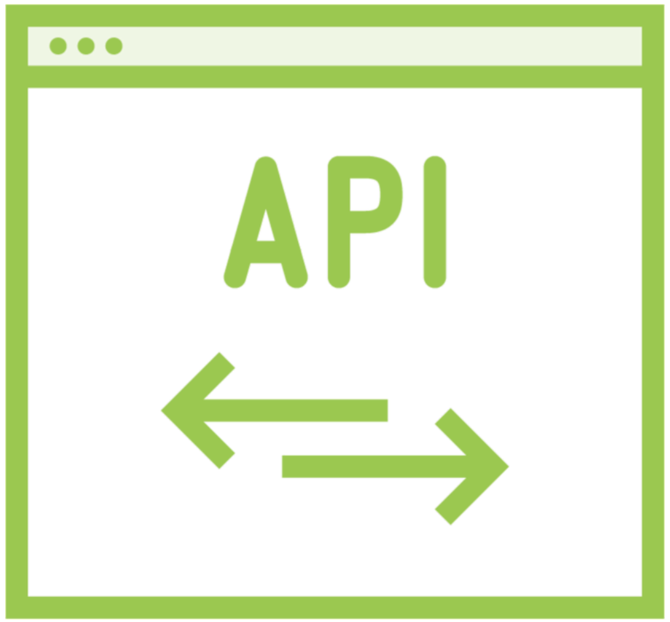
Use SaveChanges & Shadow Properties to Populate Audit Data

```
modelBuilder.Entity<Author>()
.Property<DateTime>("LastUpdated");

private void UpdateAuditData()
{
  foreach(var e in ChangeTracker.Entries()
     .Where(e=>e.Entity is Author))
  {
    entry.Property("LastUpdated")
         .CurrentValue =DateTime.Now;
  }
}

public override int SaveChanges()
{
  UpdateAuditData();
  return base.SaveChanges();
}
```

- Define shadow properties in OnModelCreating
- Create a method that updates shadow properties in the ChangeTracker.Entries()
- Call the new method in SaveChanges override
- Make sure internal SaveChanges is still called!

Using EF Core Pipeline Events

Events Exposed in EF Core API




```
DbContext.SavingChanges
DbContext.SavedChanges
DbContext.SaveChangesFailed
ChangeTracker.Tracked
ChangeTracker.StateChanged
```

Implementing Event Handlers

A method to execute custom logic

Declare the method as a handler of the target event

```
1. private void UpdateAuditData()
  {
    foreach (var entry in ChangeTracker.Entries().Where(e=>e.Entity is Author))
    {      entry.Property("LastUpdated").CurrentValue =DateTime.Now;    }
  }

2. SavingChanges += SavingChangesHandler;

3. private void SavingChangesHandler(object sender, SavingChangesEventArgs e)
  {
    UpdateAuditData();
  }
```

Using SavingChanges & Shadow Properties to Populate Audit Data

SavingChanges occurs before the SaveChanges logic is performed

1. Create a method that updates shadow property in the ChangeTracker.Entries
2. Declare event hander in constructor
3. Add logic to event handler method

SaveChange Events vs. Override SaveChanges

Individual Event Handlers

```
private void SavingChangesHandler(...)
{
    //logic before save
}

private void SavedChangesHandler(...)
{
    //logic after save
}

private void SaveFailedHandler(...)
{
    //error handling
}
```

Override SaveChanges

```
public override int SaveChanges()
{
    //some actions before save

    try
    {
       int affected=base.SaveChanges();

       //some actions after saved
       return affected;
     }
     catch
     {
       //save failed: error handling
     }
}
```

You could use these events to emit alternate information that isn’t tracked by the logger.

Using Interceptors to Inject Logic into EF Core's Pipeline

If you used Interceptors in EF6,
these work in a similar way.

Interceptors

Entity Framework Core (EF Core) interceptors enable interception, modification, and/or suppression of EF Core operations. This includes low-level database operations such as executing a command, as well as higher-level operations, such as calls to SaveChanges.

https://docs.microsoft.com/en-us/ef/core/logging-events-diagnostics/interceptors

Intercepting Database Operations

Interceptor	Database operations intercepted
IDbCommandInterceptor	Before sending a command to the database After the command has executed Command failures Disposing the command's DbDataReader
IDbConnectionInterceptor	Opening and closing connections Connection failures
IDbTransactionInterceptor	Creating transactions Using existing transactions Committing transactions Rolling back transactions Creating and using savepoints Transaction failures

Setting Up Interception

```
internal class MyInterceptor :
 DbCommandInterceptor
{
  public override
   InterceptionResult<DbDataReader>
   ReaderExecuting(DbCommand command,
   CommandEventData eventData,
   InterceptionResult<DbDataReader> result)
   {
     //do something e.g., edit command
     return result;
   }
}
```

- Define an interceptor class

```
optionsBuilder.UseSqlServer(connString)
.AddInterceptors(new MyInterceptor());
```

- Register the interceptor in your DbContext

```
builder.Services.AddDbContext
(options=> options
.AddInterceptors(new MyInterceptor());
```

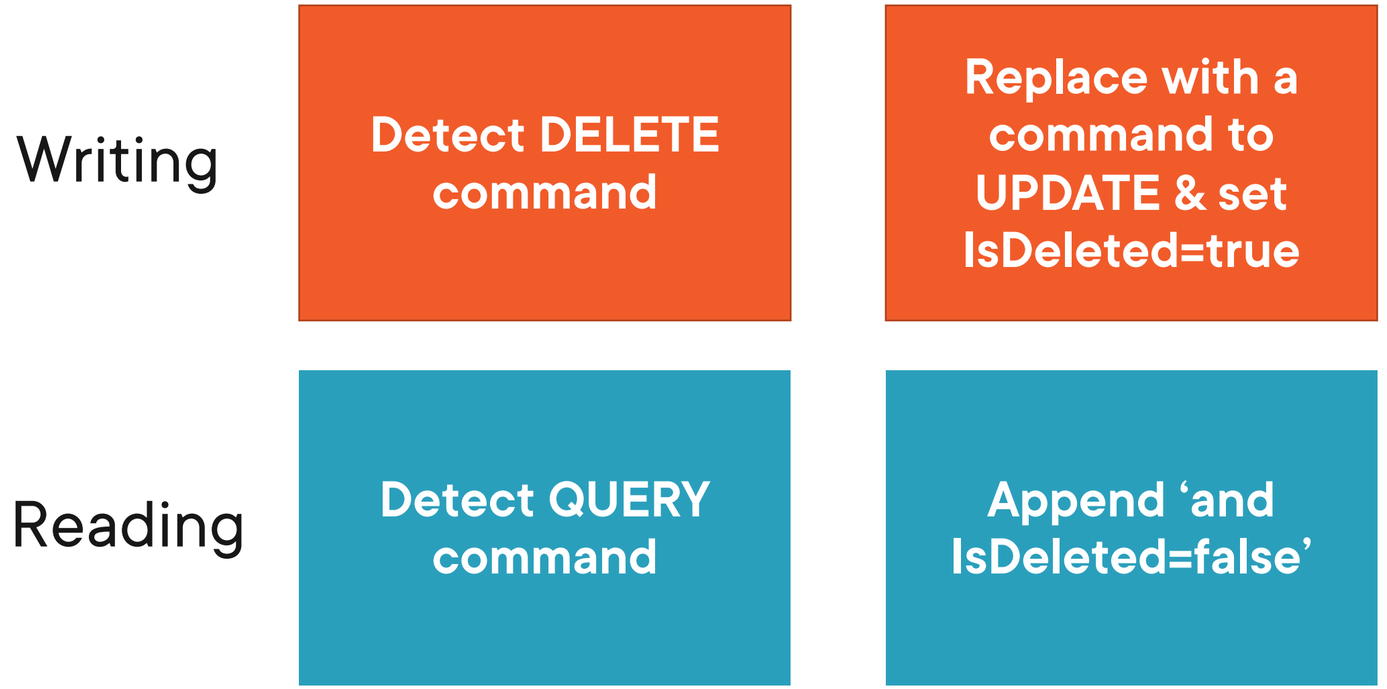
- Or via ASP.NET Core services configuration

Adding Query Hints to Commands and Other Interceptor Examples

I've run an analysis on your database and have some suggestions to improve performance.

One is to add a *Robust Plan* query hint for queries of the Authors table.

Interceptors Can Help Implement Soft Deletes



There is more to understand about soft delete before implementing it!

DbCommandInterceptor with SaveChanges

NonQueryExecuting
NonQueryExecuted

This may seem like the right option, but it is not

ReaderExecuting
ReaderExecuted

Surprise! Because SaveChanges returns a result, a reader is needed

Review

There are a variety of ways to tap into EF Core's pipeline

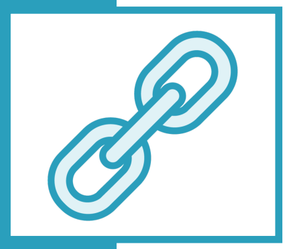
Unlike logging, these methods let you change data and behavior

ChangeTracker.Entries is a critical tool when affecting how SaveChanges works

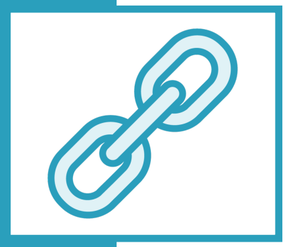
Changing how EF Core's pipeline works is powerful but requires a good understanding to avoid problematic side effects

Thank you for watching
this course!

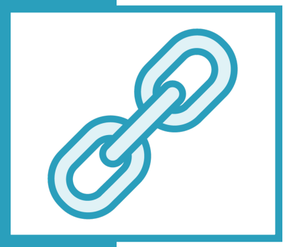
Resources



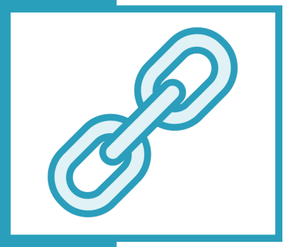
EF Core Documentation: docs.microsoft.com/ef



Tapping into EF Core's Pipeline
codemag.com/Article/2103051



EF Core 6: Fulfilling the Bucket List:
codemag.com/Article/2111072



Soft deleting data with Global Query Filters
thereformedprogrammer.net/ef-core-in-depth-soft-deleting-data-with-global-query-filters/