Working with SQL, Views, and Stored Procedures

Julie Lerman

EF Core Expert and Software Coach

@julielerman | thedatafarm.com

EF Core Allows You To Work Directly With

- SQL Commands
- Stored Procedures
- Views
- Scalar Functions
- Table Value Functions
- Queries Defined in DbContext

The Return of Mapped Stored Procedures

from Entity Framework Days

Module Overview

Querying with raw SQL commands and stored procedures

Using migrations to add views and stored procedures and more to a database

Understanding and mapping keyless entities

Mapping and querying with database views

Executing non-query SQL commands on the database

Querying with SQL

Use Query Commands or Call Stored Procedures

```
SELECT * FROM Authors
```

Pass in the SQL

```
EXEC MyStoredProc, p1, p1
```

Execute stored procedures and pass in any needed parameters

Return Entities ...

DbSet.FromSql

For interpolated strings*
Safer
Variables are parameterized

DbSet.FromSqlRaw

For simple strings

**FromSqlInterpolated() (still supported) prior to EF7*

Return Entities … or Non-Entities

DbSet.FromSql

For interpolated strings*
Safer
Variables are parameterized

DbSet.FromSqlRaw

For simple strings

Database .SqlQuery & .SqlQueryRaw

Returns scalar data or unmapped class*

**FromSqlInterpolated() (still supported) prior to EF7*
Scalar support came in EF7
Unmapped types support came in EF8

```
_context.Authors.FromSql($“some sql string {var}”).ToList();

_context.Authors.FromSqlRaw(“some sql string”).ToList();
```

DbSet Methods to Run Raw SQL

Retrieve entities without relying on LINQ or EF Core’s generated SQL
Use parameters to avoid SQL injection
Expects results to be in the shape of the DbSet’s entity
Creates an IQueryable, so you still need an execution method

```
_context.Authors.FromSql($“some sql string {var}”).ToList();

_context.Authors.FromSqlRaw(“some sql string”).ToList();
```

DbSet Methods to Run Raw SQL

Retrieve entities without relying on LINQ or EF Core’s generated SQL
Use parameters to avoid SQL injection
Expects results to be in the shape of the DbSet’s entity
Creates an IQueryable, so you still need an execution method

```
_context.Authors.FromSql($“some sql string {var}”).ToList();

_context.Authors.FromSqlRaw(“some sql string”).ToList();
```

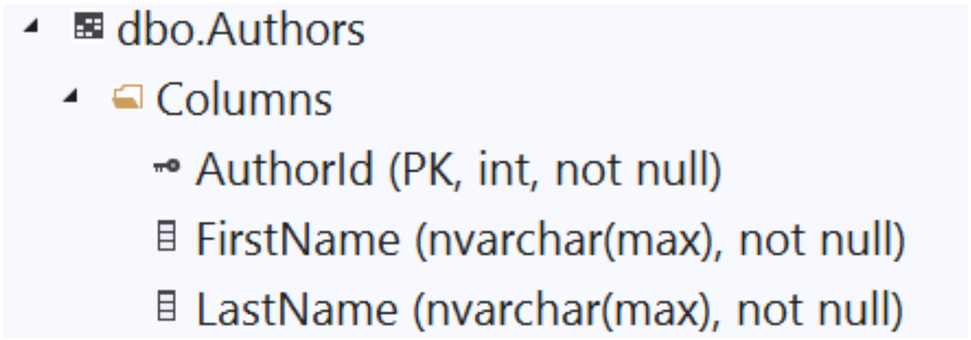
DbSet Methods to Run Raw SQL

Can return (tracked) entities, unmapped types and scalar types
Results must match
Special method for interpolated strings
Creates an IQueryable, so you still need an execution method (can be an async method)
Use parameters to avoid SQL injection

Must Align with Entity Scalars & DB Columns

FromSql($“SELECT AuthorId, FirstName, LastName FROM Authors”)

```
public class Author
{
    public int AuthorId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public List<Book> Books { get; set; } = new();
}
```



```
.FromSql($"select * from authors").FirstOrDefault(a=>a.Id==3)

.FromSql($"select * from authors").OrderBy(a=>a.LastName)

.FromSql($"select * from authors").Include(a=>a.Books)
```

You Can Compose on Top of Raw SQL Queries

LINQ methods such as Where and OrderBy
LINQ execution methods like ToList and FirstOrDefault
IQueryable methods like Include and AsNoTracking
Not DbSet methods e.g., Find!

Rules and Limitations for SQL Results

Must return data for all properties of the queried type

Column names in results match mapped column names

Query can’t contain related data, use Include

Composed queries may result in SQL that’s a bit “messier”

Interpolation for Parameters

Interpolated Strings

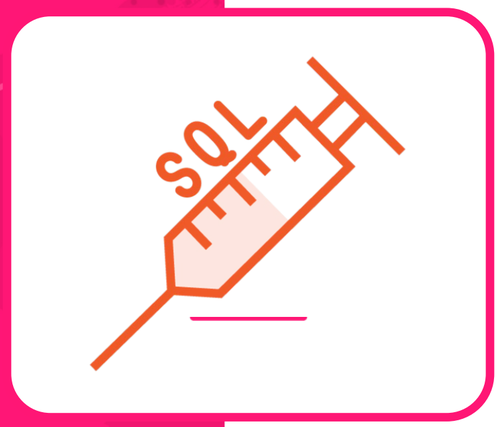
```
string name=“Josie”;
string s=$“Happy Birthday, {name}”;
```

Interpolated Strings in SQL Queries

```
var lastnameStart = "L";
 var ontheflySQL = _context.Authors.FromSql (
       $"SELECT * FROM authors WHERE lastname LIKE '{lastnameStart}%'")
       .ToList();
 var storedProc = _context.Authors.FromSql(
       $“EXEC FindAuthorByLastNameStart'{lastnameStart}%'")
       .ToList();
```

Keeping Your Database Safe with Parameterized Raw SQL Queries

Never use SQL with parameters embedded directly into the string



Learn about SQL Injection from Microsoft Docs

SQL Injection

docs.microsoft.com/sql/relational-databases/security/sql-injection

Combining Strings in C#

```
string name="Josie";
string s="Happy Birthday," + name;
```

- Concatenated string

  Happy Birthday, Josie

```
string name="Josie";
int age=6;
string s= String.Format(
  "Happy {0} Birthday {1},",
   age, name);
```

- Formatted string

  Happy 6 Birthday, Josie

```
string name="Josie";
string s=$"Happy Birthday, {name}";
```

- Interpolated string

  Happy Birthday, Josie

The Compiler Will Catch a Few Bad Apples

FromSql
expects one formatted string
as its parameter

FromSql
will not accept a string

Don’t depend on the compiler to protect you from unsafe raw SQL!

Review of Safe & Unsafe/Uncompilable Queries

Safe Query strings

Formatted as param of FromSqlRaw

Interpolated as param of FromSql

Unsafe Raw Queries

All concatenated queries are unsafe

Formatted strings in a variable

Interpolated strings in a variable

Won’t even compile:

String in FromSql

Formatted string in FromSqlRaw

Adding Stored Procedures and Other Database Objects Using Migrations

Embedding SQL in Your Apps

If you can keep SQL commands out of your code base, that's a bonus

Sometimes you don't have that option and EF Core is there to support your need

EF Core’s raw SQL methods support calling stored procedures

We Need a Stored Procedure in Our DB

While there are various ways to achieve this, we will use EF Core migrations to do the job.

Consistent workflows are important whether you are solo or working on a team

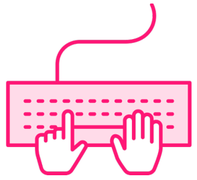
The Stored Procedure

Retrieve authors who published a book in a range of years

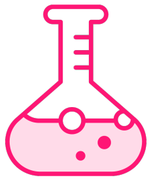
```
EXEC thesproc @startyear, @endyear
```

Workflow for Adding DB Objects with Migrations

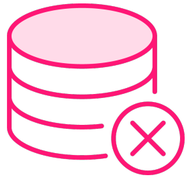
Work out the query directly against the database



Build the command to create the procedure



Test the command by creating & running the stored procedure



Remove the procedure from the database

Add the create procedure command to a new migration

Why Use the Migration for This?

Other team members can easily migrate their own development databases

The migration becomes part of your source code

Useful in other environments such as CI/CD, acceptance testing and possibly production

Running Stored Procedure Queries with Raw SQL

```
FromSqlRaw with formatted string

DbSet.FromSqlRaw("Exec MyStoredProc, {0}, {1}", firstValue, secondValue)


FromSql with interpolated string

DbSet.FromSql($"EXEC MyStoredProc {firstValue}, {secondValue}")
```

Querying via DbSets Using Stored Procedures

Include values of expected parameters
EF Core will transform this as needed by the database
There are more variations. Check EF Core docs.
As always, you cannot compose on top of Stored Procs in SQL Server

Stored procedures aren’t composable!

Stored procedures aren’t composable!

Watch out for in-memory processing.

Raw SQL Method Rules Apply to Sprocs, Too

SQL cannot return shaped data but with sprocs, you can't Include!

Schema of results must match the entity of the DbSet

Column names of results must match property names of entity

EF Core cannot capture random data, e.g., anonymous types, from raw SQL

Compiler Cannot Detect Bad SQL

SQL errors will only be caught at runtime by the database which will return an error

Querying for Scalars and Other Non-Entity Types

```
DbContext.Database
  .SqlQuery<type>(expression)
  .LINQExecutionMethod();
```

Query for Scalar or Class Data

SqlQuery

(Interpolated string)

SqlQueryRaw

(Raw or Formatted string)

Query tables/views

Or stored procedures

Querying for Scalar Data

```
_context.Database
    .SqlQuery<int>($"SELECT AuthorId FROM Authors").ToList();

_context.Database
 .SqlQuery<string>($"SELECT Title FROM Books").ToList();

_context.Database
     .SqlQuery<string>($"SELECT Title as VALUE FROM Books")
     .Where(t => t.Contains("The")).ToList();
```

Querying for Un-Mapped Class Types

```
_context.Database
     .SqlQuery<AuthorName>($"select lastname, firstname from authors")
     .ToList();

_context.Database
     .SqlQuery<AuthorName>($"GetAuthorNames").ToList();
```

Using Keyless Entities to Map to Views

Historically, EF and EF Core only understood entities with keys

But now you can use keyless entities

Keyless Entities != Non-Tracking Queries

Non-Tracking Query

Entity has a key prop

No-tracking is optional

Maps to tables with PK

Mixed Use

Entity has a key prop

Maps to view and table

Query from the view,
update to the table

Keyless entities are always read-only

Dear Programmers,

We need to see for which authors the artists are working on book covers.

Thanks!

Your fans, the editors

This calls for a database view.

Hey, DB guru, we have a favor to ask ....

Mapping Keyless Entities & ...

Create DB View

Define Keyless Entity

A mapping!

Another mapping!

Mapping Keyless Entities & DB Views

Create DB View

Define Keyless Entity

Mapping the Keyless Entity

Another mapping: *for the view!*

EF Core maps entities to tables by default

It will expect (& ask migrations to create) a table named AuthorsByArtist!

BUT, we want it to use our existing view.

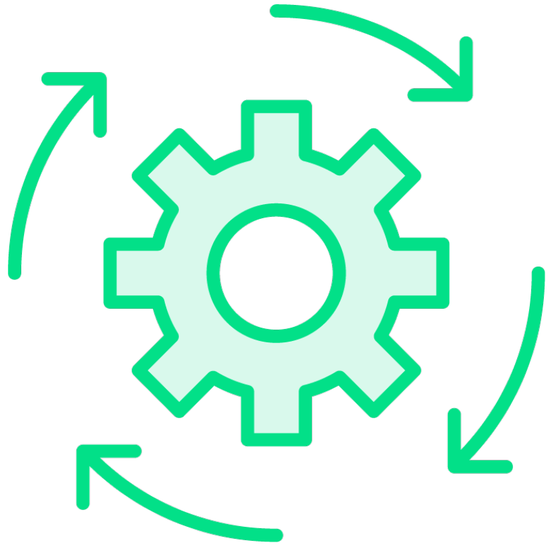
Migrations will not attempt to create a database view that's mapped in a DbContext

Keyless entities will never be tracked.

Full stop.

Querying the Database Views

Not All DbSet Methods Work with Keyless Entities



Find() will compile

But Find() will fail at runtime!

Executing Non-Query SQL Commands

Commands to Execute Non-Querying SQL and Sprocs

ExecuteSqlRaw

For plain text

ExecuteSql

For interpolated text

Commands to Execute Non-Querying SQL and Sprocs

DbContext.Database .ExecuteSqlRaw

For plain text

DbContext.Database .ExecuteSql

For interpolated text

```
_context.Database.ExecuteSqlRaw(“some SQL string”);

_context.Database.ExecuteSql($“some SQL string {variable}”);

_context.Database.ExecuteSqlInterpolated($“some SQL string {var}”);
```

ExecuteSql Methods are Not For Queries

Only result is number of rows affected
On-the-fly SQL or Stored Procedures
Async versions as well
...and just a reminder that the “interpolated” versions still work. It’s not a breaking change!

Demo

Finally, we'll use a stored procedure to delete data

Like ExecuteUpdate and ExecuteDelete, deleting with a stored procedure will not interact with the change tracker.

Mapping Insert, Update, & Delete Stored Procedures to Entities

Most dev is done against a legacy database with a lot of pre-existing procedures

Why Map to Procedures?

Important biz rules may be expressed in those procedures

Company policy to use them

“Support for stored procedure mapping does not imply that stored procedures are recommended.” – EF Core docs

Procedure Mapping Methods

```
ModelBuilder.Entity<someEntity>()

                .InsertUsingStoredProcedure()

                .UpdateUsingStoredProcedure
                ()
                .DeleteUsingStoredProcedur
                e()
```

Review

More querying using raw SQL and stored procedures including interpolated strings

Used customized migrations to add views, stored procedures and more to DB

Learned about keyless entities and mapped them to a database view

Used DbSet to query that database view

Execute non-query commands from the DbContext.Database property

Return non-entity data & map sprocs

Up Next:

Using EF Core in ASP.NET Core Apps

Resources

Stored Procedure Mapping in EF Core docs: learn.microsoft.com/ef/core/what-is-new/ef-core-7.0/whatsnew#stored-procedure-mapping

About SQL Injection:
learn.microsoft.com/sql/relational-databases/security/sql-injection

EF Core Query Types Consolidated with Entity Types:
learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.0/breaking-changes#qt

Resources, Continued

Raw SQL Queries with LINQ:
https://learn.microsoft.com/ef/core/querying/raw-sql

"EF Core 7: It Just Keeps Getting Batter" (Code Magazine Code Focus)
codemag.com/Article/2211072

Database functions in EF Core Docs:
learn.microsoft.com/en-us/ef/core/querying/database-functions