Controlling Database Creation and Schema Changes with Migrations

Julie Lerman
Most Trusted Authority on Entity Framework Core

@julielerman   thedatafarm.com

Module Overview

Overview of EF Core Migrations API

Setting up your project and Visual Studio for migrations

Create and inspect a migration file

Using EF Core Migrations to create a database or database scripts

Reverse engineer and existing database into classes and DbContext

Understanding EF Core Migrations

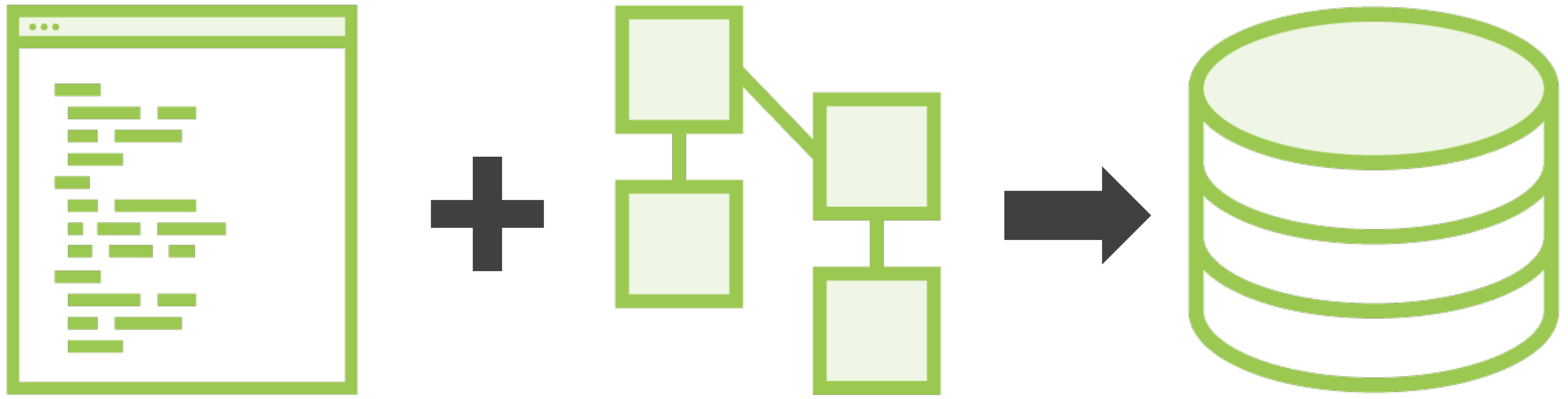
EF Core Needs to Comprehend the DB Schema

Build SQL from your LINQ queries

Materialize query results into objects

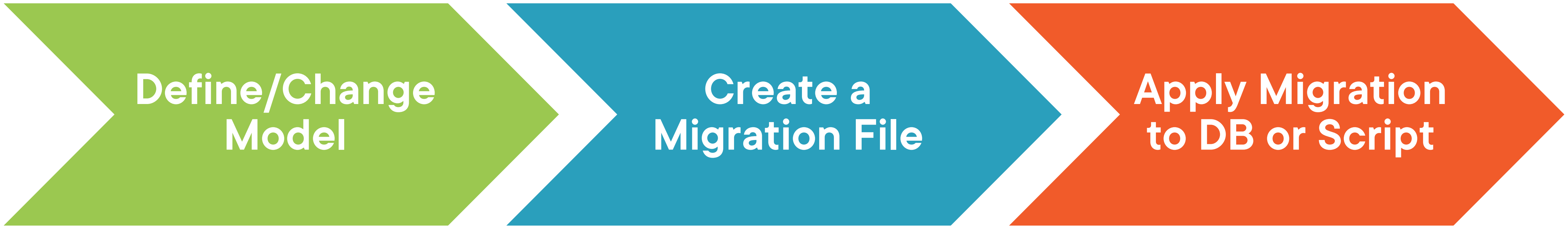
Build SQL to save data into database

Mapping Your Data Model to the Database



Mapping knowledge can also be used to evolve the database schema

EF Core Basic Migrations Workflow

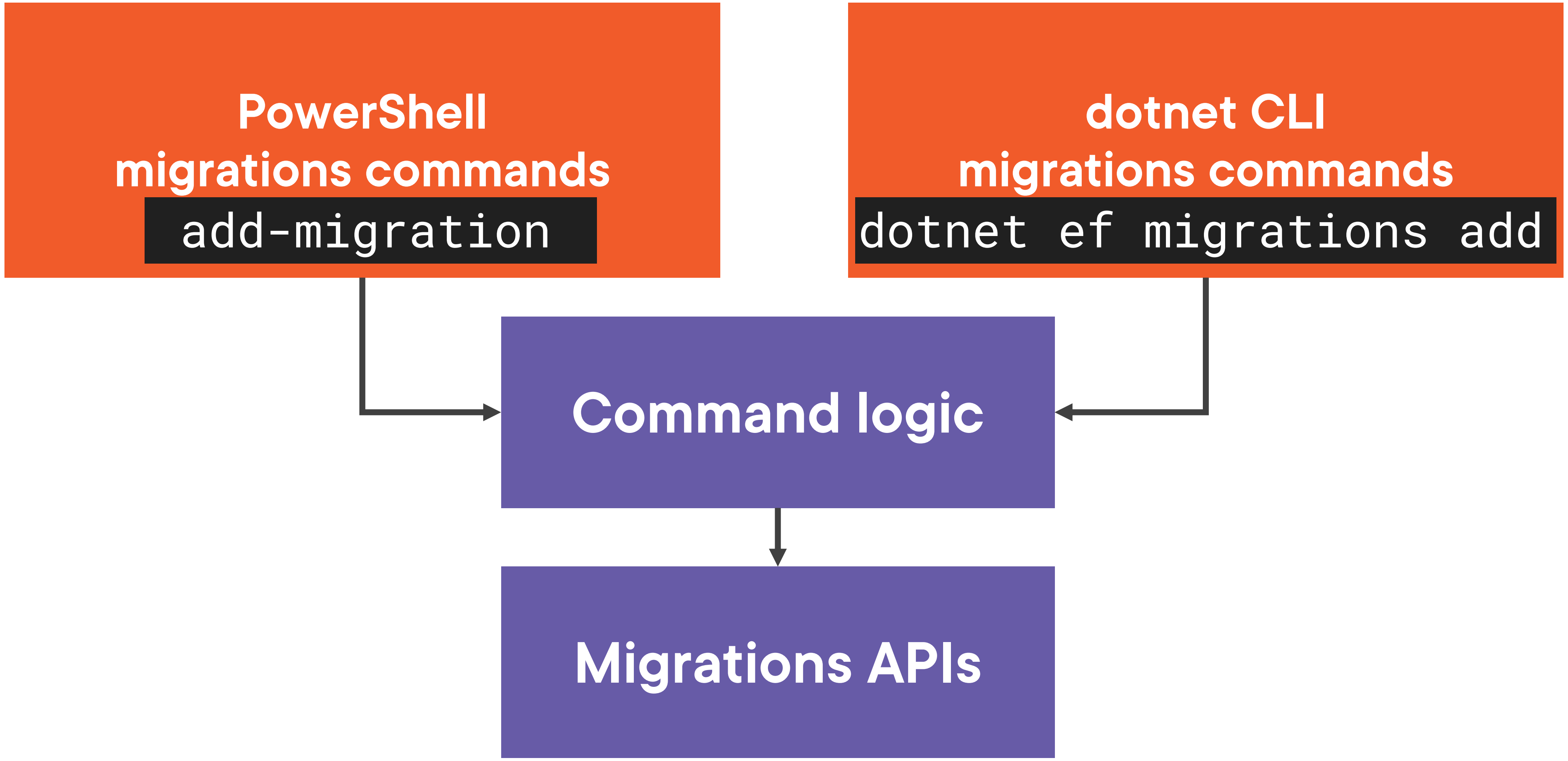


EF Core Migrations
are
source-control friendly

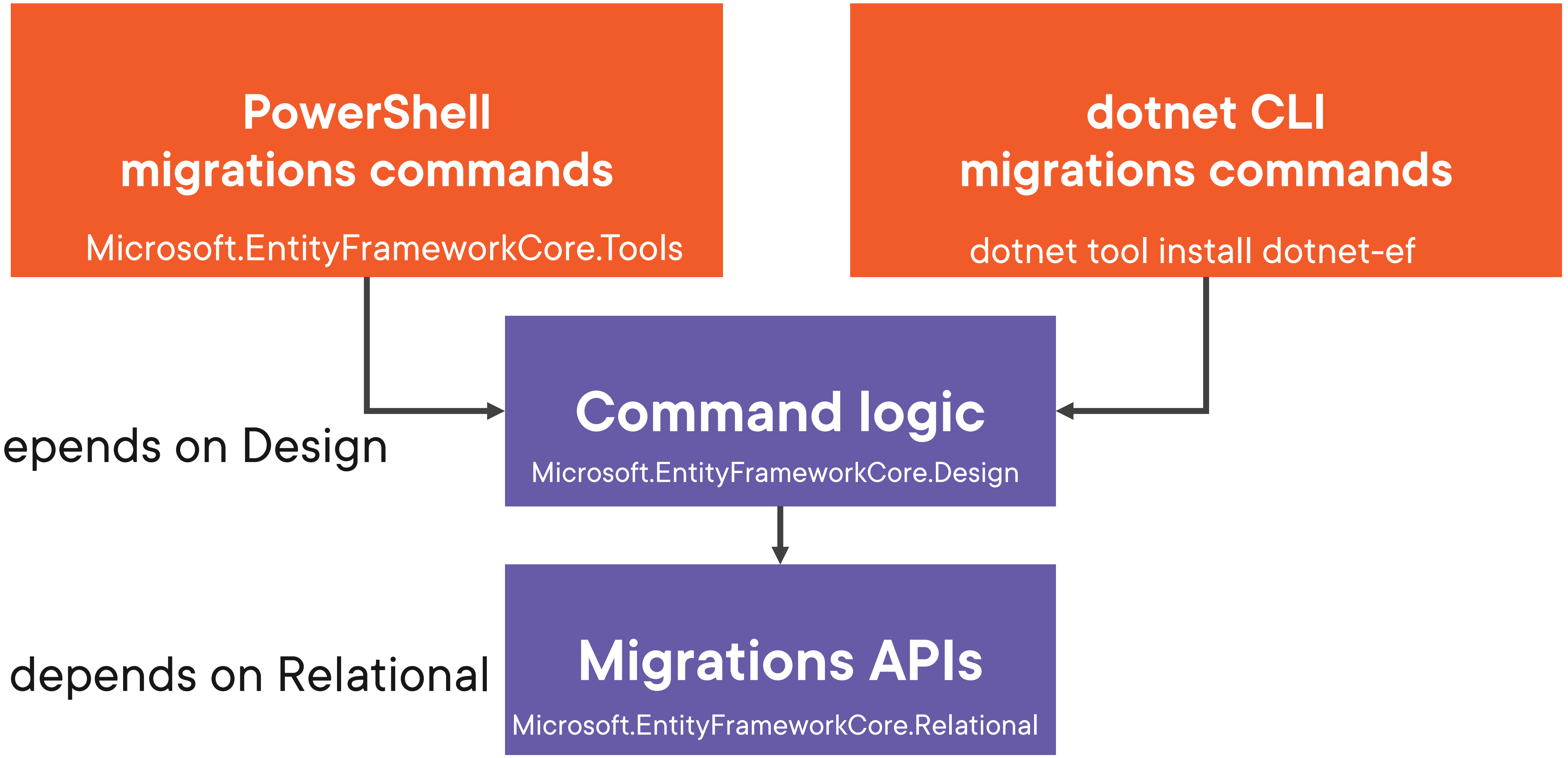
Getting and Understanding the Design-Time Migrations Tools

Creating and executing migrations happens at design time

Migration Commands Lead to Migrations APIs



Migration Commands Lead to Migrations APIs


PowerShell migrations commands
Microsoft.EntityFrameworkCore.Tools
dotnet CLI migrations commands
dotnet tool install dotnet-ef
Command logic
Microsoft.EntityFrameworkCore.Design
Migrations APIs
Microsoft.EntityFrameworkCore.Relational

Tools depends on Design

Design depends on Relational

Bottom Line

- Add Tools package to project
- Design comes for “free”

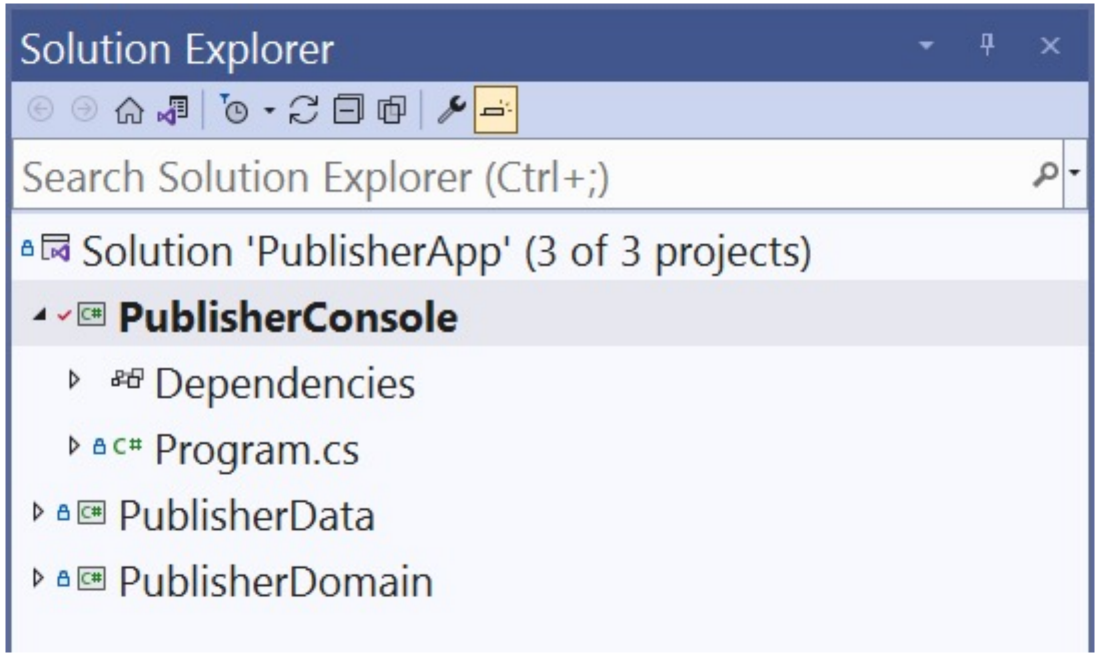
Visual Studio (Windows)

- Install the tools on your system
- Design package in your project

Command Line

Which Project Gets the Tools?

The executable project



Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'PublisherApp' (3 of 3 projects)

PublisherConsole

Dependencies

Program.cs

PublisherData

PublisherDomain

Getting the Package Manager Console Ready to Use Migrations

Using Migrations in Visual Studio When EF Core Is in a Class Library Project

- ✓ **Install Microsoft.EntityFrameworkCore.Tools package into executable project (e.g., console)**
- ✓ **Ensure the executable project is the startup project**
- ✓ **Set Package Manager Console (PMC) "default project" to class library with EF Core (e.g., data)**
- ✓ **Run EF Core Migration PowerShell commands in PMC**

Using Migrations in dotnet CLI

Add EF Core tools to system*

Add EF Core Design** package to executable project

Use "dotnet ef" commands at command line

*Command to install: dotnet tool install –global dotnet-ef
**Microsoft.EntityFrameworkCore.Design

Adding Your First Migration

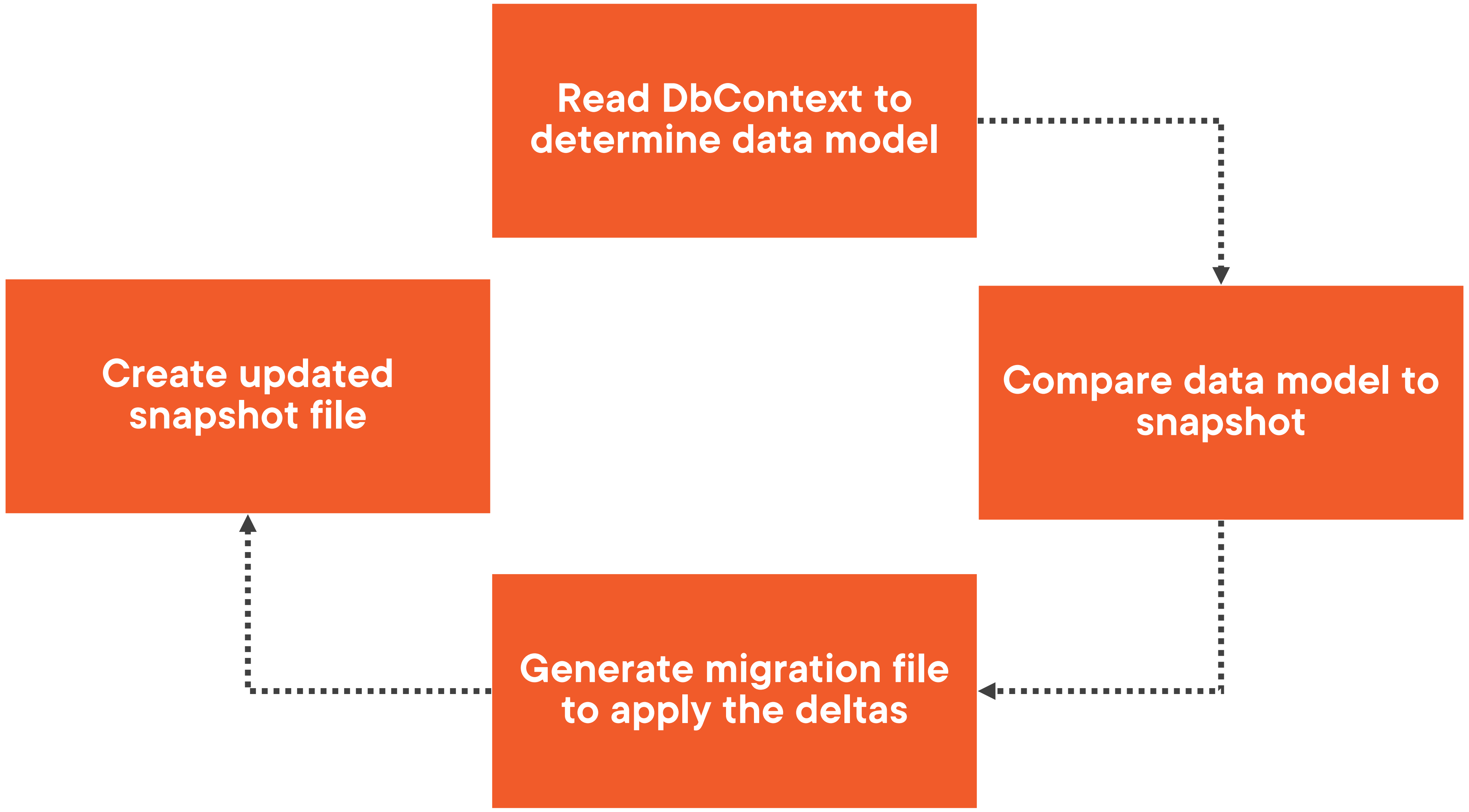
Add-Migration Tasks for Initial Schema

Read DbContext and determine data model

Create a migration file describing how to construct the database schema

Inspecting Your First Migration

Add-Migration Tasks for Model Changes


Read DbContext to determine data model
Compare data model to snapshot
Generate migration file to apply the deltas
Create updated snapshot file

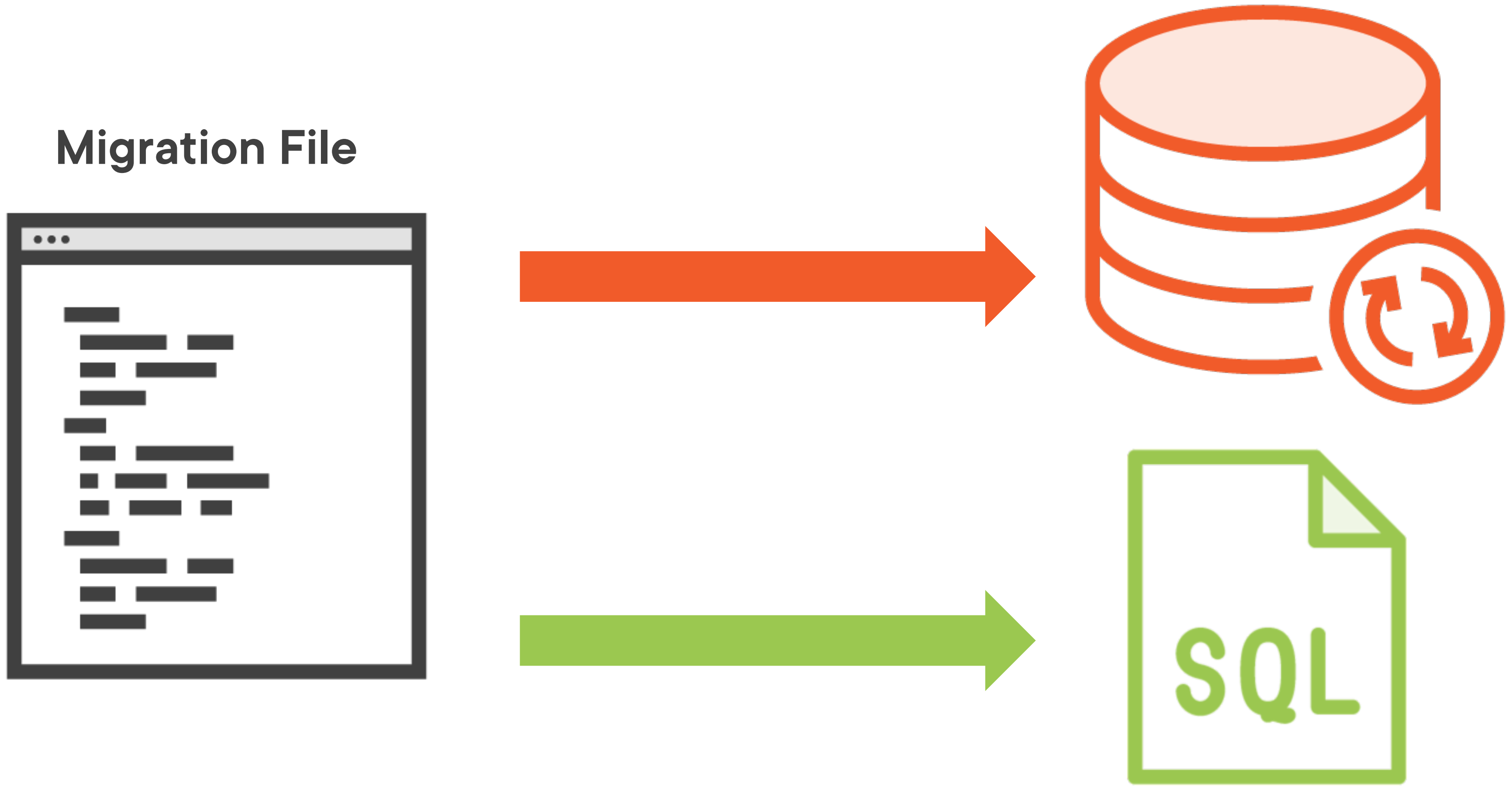
For EF 6 users:
EF Core 6 migrations is orders of magnitude easier to use in source control

Some EF Core Mapping Conventions (Defaults)

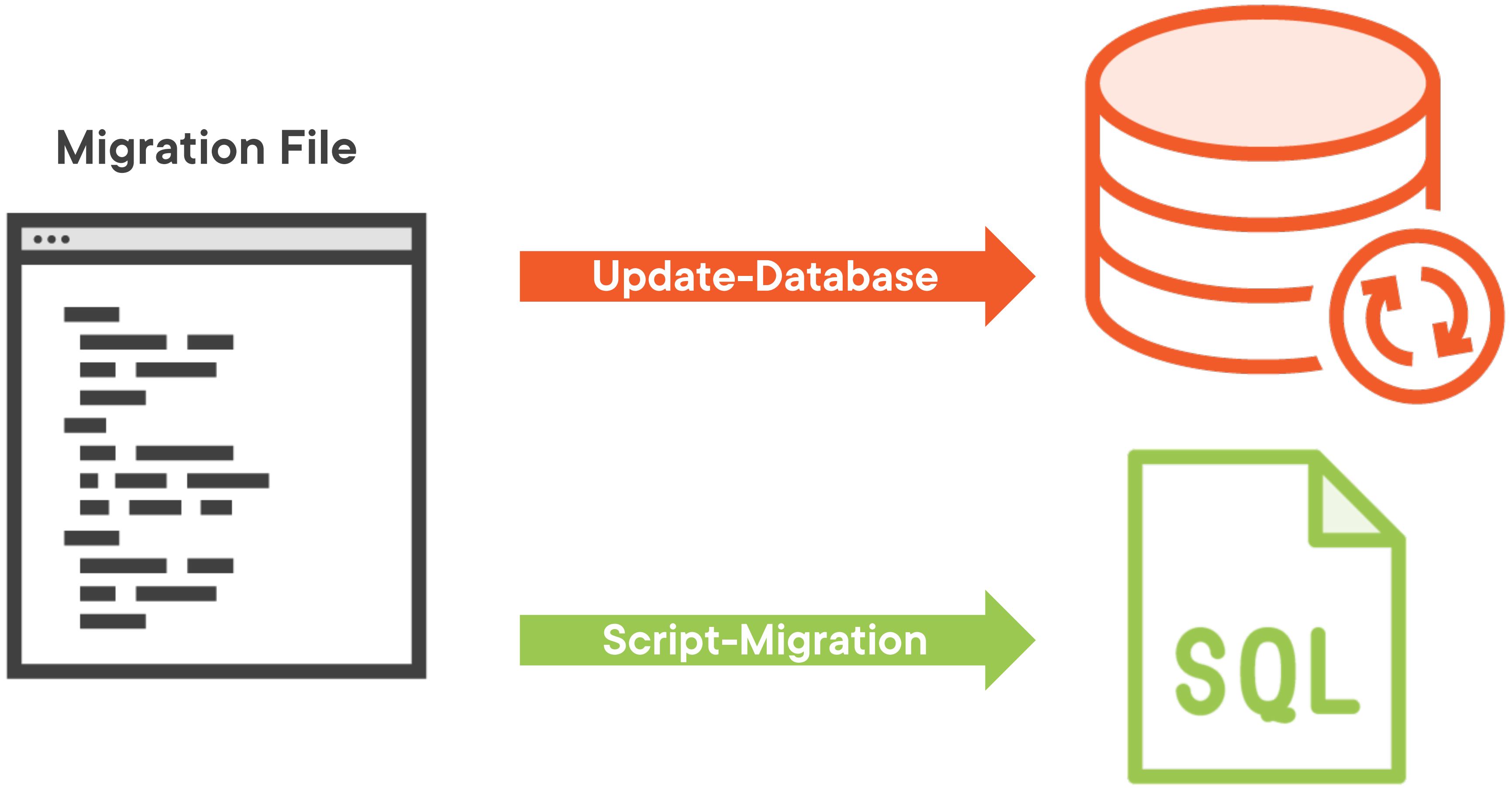
- **DbSet name is the table name**
- **Class property name is the column name**
- **Strings are determined by provider**
  e.g., SQL Server: nvarchar(max)
- **Decimals are determined by provider**
  e.g., SQL Server: decimal(18,2)
- **“Id” or “[type]Id” are primary keys**

Using Migrations to Script or Directly Create the Database

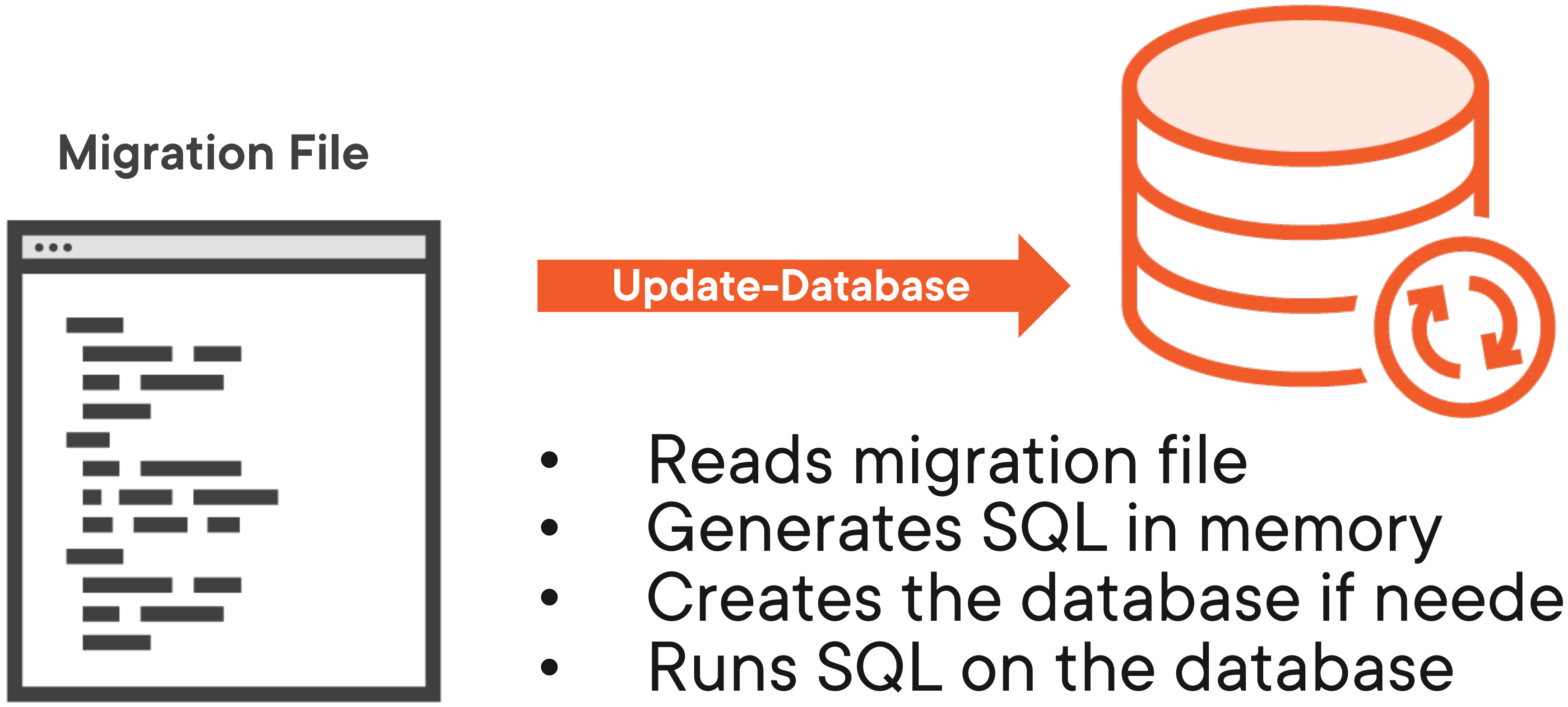
Applying Migrations



Applying Migrations



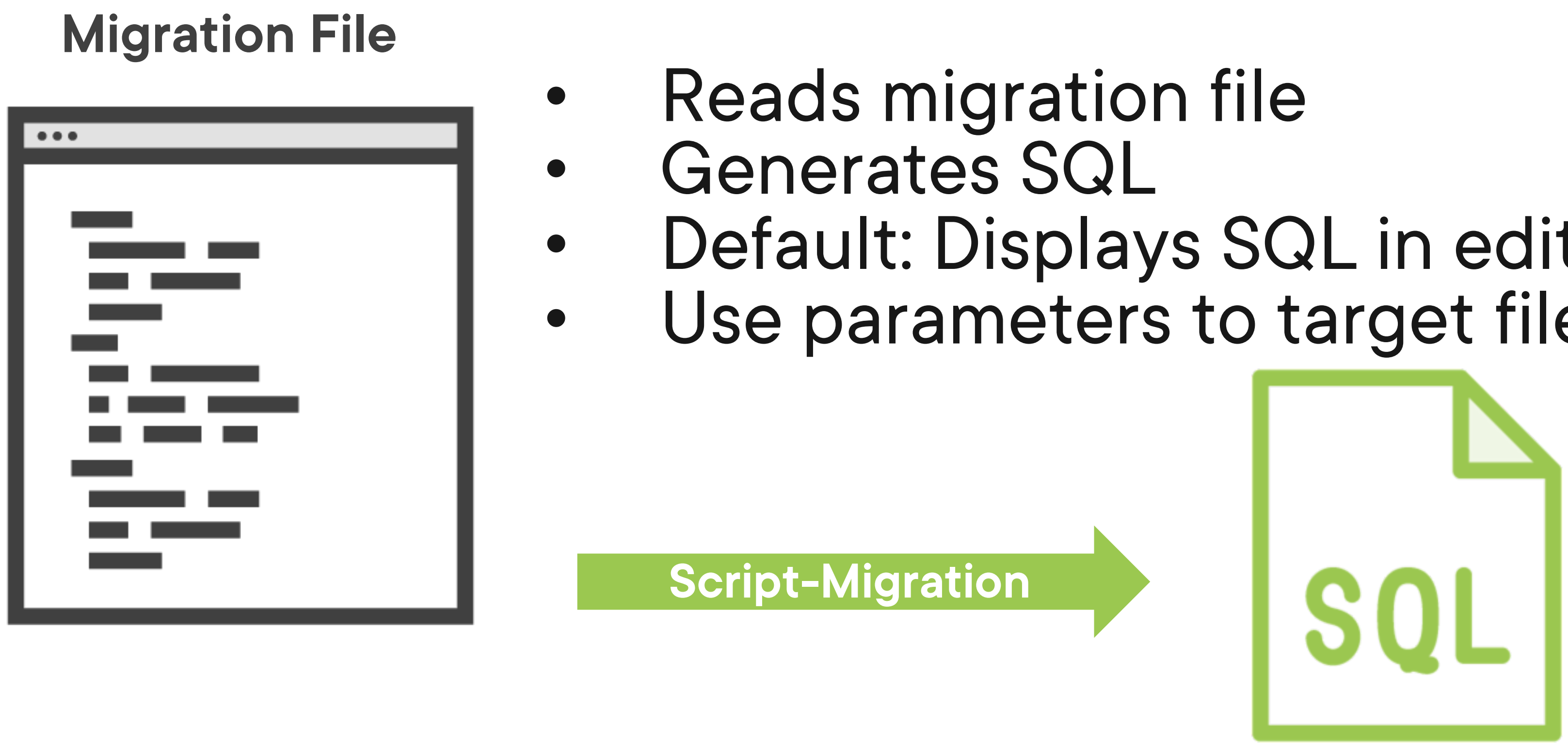
Applying Migrations Directly to the Database



- Reads migration file
- Generates SQL in memory
- Creates the database if needed
- Runs SQL on the database

CLI: dotnet ef database update

Applying Migrations into a SQL Script



- Reads migration file
- Generates SQL
- Default: Displays SQL in editor
- Use parameters to target file name etc.

CLI: dotnet ef migrations script

Migrations Recommendation

Development database

`update-database`

Production database

`script-migration`

We will create a fresh database in this module.

What If Database Does Not Exist?

`update-database`

API's internal code will create the database before executing migration code

`script-migration`

You must create the database before running the script

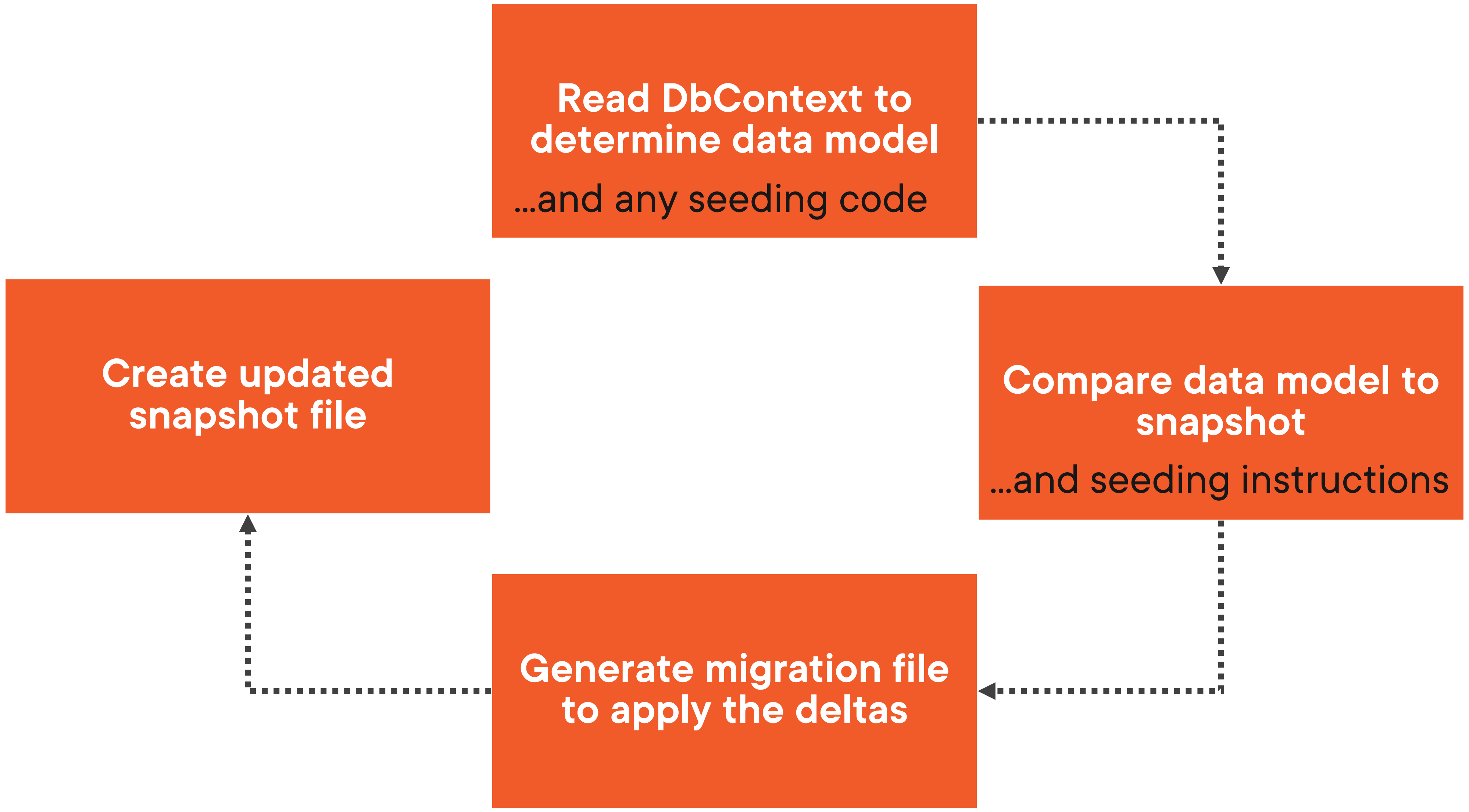
Seeding a Database via Migrations

```
modelBuilder.Entity<EntityType>().HasData(parameters)

modelBuilder.Entity<Author>().HasData(new Author {Id=1, FirstName="Julie", .. };
```

Specify Seed Data with ModelBuilder HasData Method

Provide all non-nullable parameters including keys and foreign keys
HasData will get translated into migrations
Inserts will get interpreted into SQL
Data will get inserted when migrations are executed

Add-Migration Tasks for Model Changes


Read DbContext to determine data model
...and any seeding code
Compare data model to snapshot
...and seeding instructions
Generate migration file to apply the deltas
Create updated snapshot file

Seeding with HasData will not cover all use cases for seeding

Use Cases for Seeding with HasData

Mostly static seed data

Sole means of seeding

No dependency on anything else in the database

Provide test data with a consistent starting point

HasData will also be recognized and applied by EnsureCreated

Scripting Multiple Migrations

Scripting migrations requires more control, so it works differently than update-database.

Some Scripting Options

`script-migration`

Default: Scripts every migration

```
script-migration
-idempotent
```

Scripts all migrations but checks for each object first e.g, table already exists

```
script-migration
FROM  TO
```

FROM Arg: Specifies last migration run, so start at the next one TO Arg: final one to apply

Reverse Engineering an Existing Database

Scaffolding Builds DbContext and Entity Classes

Reverse Engineer with the Scaffold Command

PowerShell

`Scaffold-DbContext`

EF Core CLI

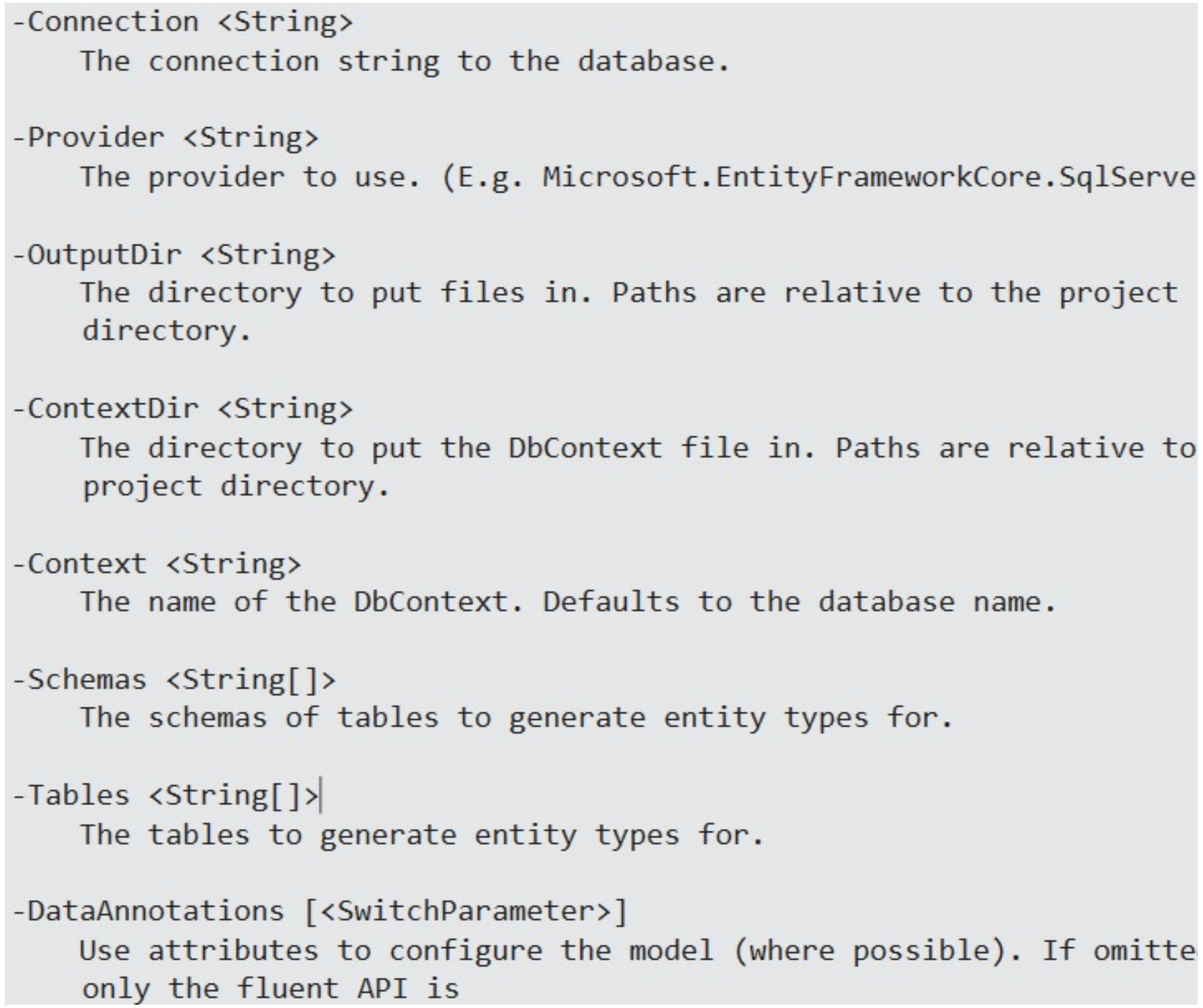
`dotnet ef dbcontext scaffold`

Scaffolding Limitations

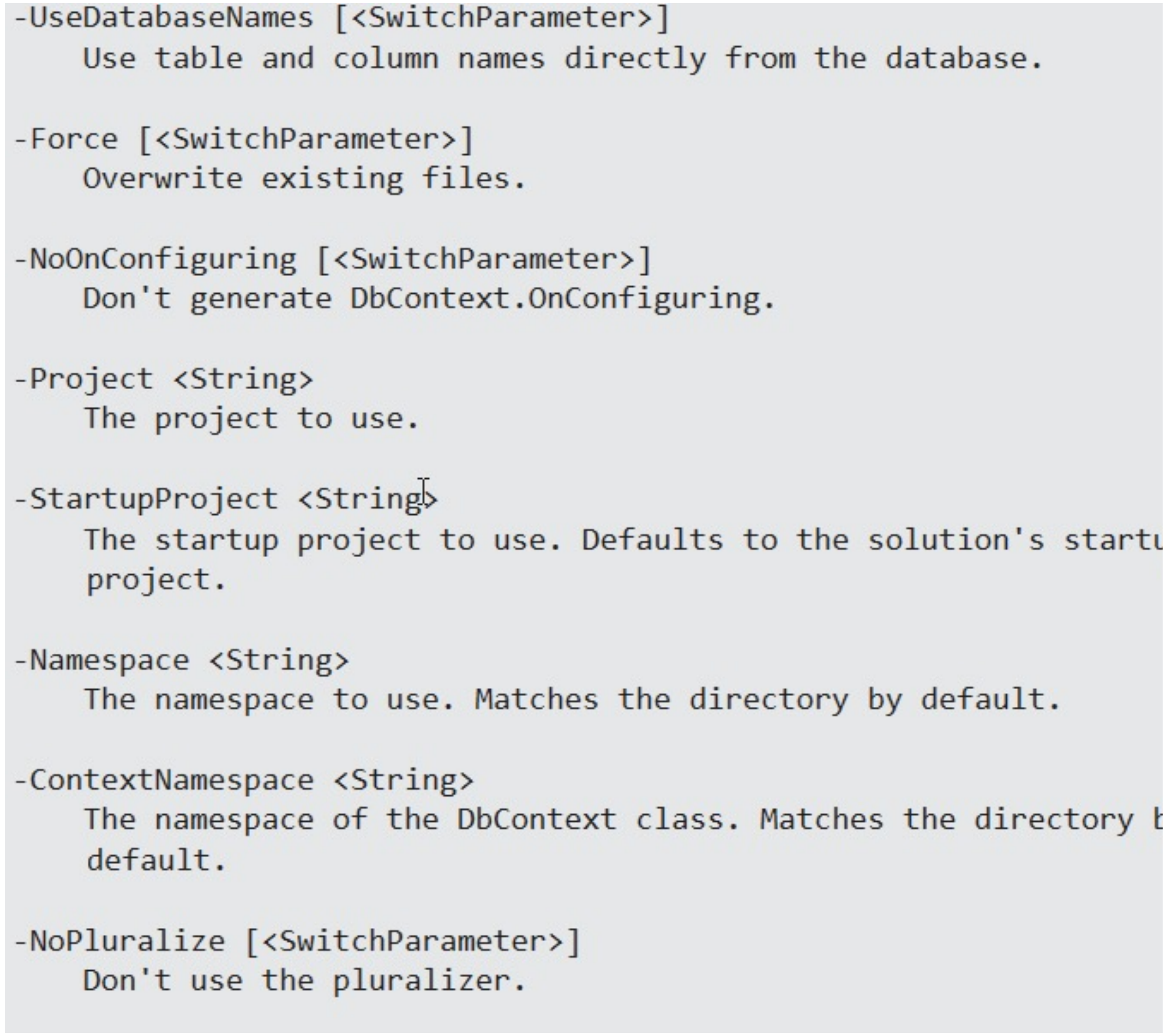
Updating model when database changes is not currently supported

Transition to migrations is not pretty. Look for helpful link in resources

The Many Parameters of scaffold-dbcontext

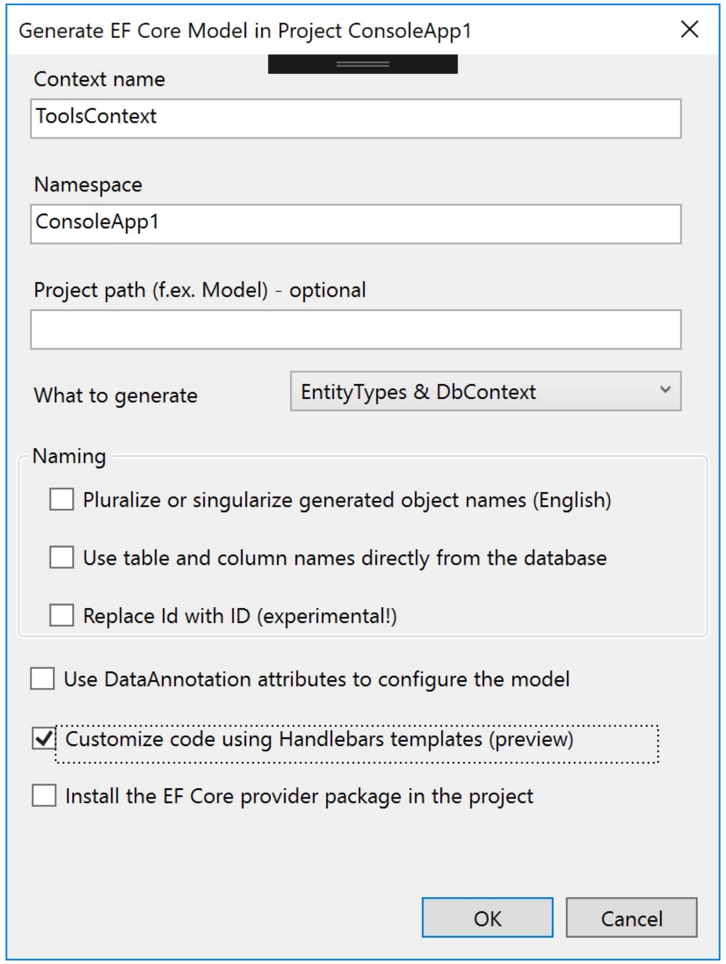


```
-Connection <String>
    The connection string to the database.

-Provider <String>
    The provider to use. (E.g. Microsoft.EntityFrameworkCore.SqlServe

-OutputDir <String>
    The directory to put files in. Paths are relative to the project
    directory.

-ContextDir <String>
    The directory to put the DbContext file in. Paths are relative to
    project directory.

-Context <String>
    The name of the DbContext. Defaults to the database name.

-Schemas <String[]>
    The schemas of tables to generate entity types for.

-Tables <String[]>
    The tables to generate entity types for.

-DataAnnotations [<SwitchParameter>]
    Use attributes to configure the model (where possible). If omitte
    only the fluent API is
```



```
-UseDatabaseNames [<SwitchParameter>]
    Use table and column names directly from the database.

-Force [<SwitchParameter>]
    Overwrite existing files.

-NoOnConfiguring [<SwitchParameter>]
    Don't generate DbContext.OnConfiguring.

-Project <String>
    The project to use.

-StartupProject <String>
    The startup project to use. Defaults to the solution's startu
    project.

-Namespace <String>
    The namespace to use. Matches the directory by default.

-ContextNamespace <String>
    The namespace of the DbContext class. Matches the directory b
    default.

-NoPluralize [<SwitchParameter>]
    Don't use the pluralizer.
```

EF Core Power Tools for Visual Scaffolding



Generate EF Core Model in Project ConsoleApp1

Context name: ToolsContext

Namespace: ConsoleApp1

Project path (f.ex. Model) - optional:

What to generate: EntityTypes & DbContext

Naming

- ☐ Pluralize or singularize generated object names (English)
- ☐ Use table and column names directly from the database
- ☐ Replace Id with ID (experimental!)

☐ Use DataAnnotation attributes to configure the model

[✓ Customize code using Handlebars templates (preview)]

☐ Install the EF Core provider package in the project

VS extension: ErikEJ.EFCorePowerTools

Free

Open-source (github.com/ErikEJ/EFCorePowerTools)

Built and maintained by Erik Ejlskov Jensen

Many more features besides reverse engineer

How EF Core Determines Mappings to DB

Conventions

Default assumptions

```
property name=column name
```

Override with Fluent Mappings

Apply in DbContext using Fluent API

```
modelBuilder.Entity<Book>()
 .Property(b => b.Title)
 .HasColumnName(“MainTitle");
```

Override with Data Annotations

Apply in entity

```
[Column(“MainTitle")]
public string Title{get;set;}
```

Review

Workflow of how EF Core determines database schema

Where Migrations API and tools fit in

PowerShell or CLI commands for creating and executing migrations

Created and explored a migrations file

Used migrations commands to generate script or create a new database directly

Reverse engineer existing database into classes and DbContext

Up Next: Defining One-to-Many Relationships

Resources

Entity Framework Core on GitHub: github.com/dotnet/efcore

EF Core Tools Documentation: docs.microsoft.com/ef/core/cli/

EF Core Power Tools Extension (model visualizer, scaffold and more): https://github.com/ErikEJ/EFCorePowerTools

Resources Cont.

EF Core migrations with existing database schema:
cmatskas.com/ef-core-migrations-with-existing-database-schema-and-data

Scott Hanselman "Magic Unicorn" blog post:
hanselman.com/blog/entity-framework-magic-unicorn-and-much-more-is-now-open-source-with-take-backs