Data Seeding

Article • 05/12/2022 • 3 minutes to read

Data seeding is the process of populating a database with an initial set of data.

There are several ways this can be accomplished in EF Core:

- Model seed data
- Manual migration customization
- Custom initialization logic

Model seed data

Unlike in EF6, in EF Core, seeding data can be associated with an entity type as part of the model configuration. Then EF Core migrations can automatically compute what insert, update or delete operations need to be applied when upgrading the database to a new version of the model.

> **Note**
>
> Migrations only considers model changes when determining what operation should be performed to get the seed data into the desired state. Thus any changes to the data performed outside of migrations might be lost or cause an error.

As an example, this will configure seed data for a `Blog` in `OnModelCreating`:

C#

```
modelBuilder.Entity<Blog>().HasData(new Blog { BlogId = 1, Url =
"http://sample.com" });
```

To add entities that have a relationship the foreign key values need to be specified:

C#

```
modelBuilder.Entity<Post>().HasData(
    new Post { BlogId = 1, PostId = 1, Title = "First post", Content = "Test
1" });
```

If the entity type has any properties in shadow state an anonymous class can be used to provide the values:

C#

```
modelBuilder.Entity<Post>().HasData(
    new { BlogId = 1, PostId = 2, Title = "Second post", Content = "Test 2"
});
```

Owned entity types can be seeded in a similar fashion:

C#

```
modelBuilder.Entity<Post>().OwnsOne(p => p.AuthorName).HasData(
    new { PostId = 1, First = "Andriy", Last = "Svyryd" },
    new { PostId = 2, First = "Diego", Last = "Vega" });
```

See the full sample project for more context.

Once the data has been added to the model, migrations should be used to apply the changes.

> **Tip**
>
> If you need to apply migrations as part of an automated deployment you can **create a SQL script** that can be previewed before execution.

Alternatively, you can use `context.Database.EnsureCreated()` to create a new database containing the seed data, for example for a test database or when using the in-memory provider or any non-relational database. Note that if the database already exists, `EnsureCreated()` will neither update the schema nor seed data in the database. For relational databases you shouldn't call `EnsureCreated()` if you plan to use Migrations.

Limitations of model seed data

This type of seed data is managed by migrations and the script to update the data that's already in the database needs to be generated without connecting to the database. This imposes some restrictions:

- The primary key value needs to be specified even if it's usually generated by the database. It will be used to detect data changes between migrations.
- Previously seeded data will be removed if the primary key is changed in any way.

Therefore this feature is most useful for static data that's not expected to change outside of migrations and does not depend on anything else in the database, for example ZIP codes.

If your scenario includes any of the following it is recommended to use custom initialization logic described in the last section:

- Temporary data for testing
- Data that depends on database state
- Data that is large (seeding data gets captured in migration snapshots, and large data can quickly lead to huge files and degraded performance).
- Data that needs key values to be generated by the database, including entities that use alternate keys as the identity
- Data that requires custom transformation (that is not handled by value conversions), such as some password hashing
- Data that requires calls to external API, such as ASP.NET Core Identity roles and users creation

Manual migration customization

When a migration is added the changes to the data specified with `HasData` are transformed to calls to `InsertData()`, `UpdateData()`, and `DeleteData()`. One way of working around some of the limitations of `HasData` is to manually add these calls or custom operations to the migration instead.

C#

```
migrationBuilder.InsertData(
    table: "Blogs",
    columns: new[] { "Url" },
    values: new object[] { "http://generated.com" });
```

Custom initialization logic

A straightforward and powerful way to perform data seeding is to use DbContext.SaveChanges() before the main application logic begins execution.

C#

```
using (var context = new DataSeedingContext())
{
    context.Database.EnsureCreated();

    var testBlog = context.Blogs.FirstOrDefault(b => b.Url ==
"http://test.com");
    if (testBlog == null)
    {
        context.Blogs.Add(new Blog { Url = "http://test.com" });
    }

    context.SaveChanges();
}
```

> ⚠ **Warning**
>
> The seeding code should not be part of the normal app execution as this can cause concurrency issues when multiple instances are running and would also require the app having permission to modify the database schema.

Depending on the constraints of your deployment the initialization code can be executed in different ways:

- Running the initialization app locally
- Deploying the initialization app with the main app, invoking the initialization routine and disabling or removing the initialization app.

This can usually be automated by using publish profiles.