EF: Derived Stored Types

If you want to create an entity type that happens to derive from another entity type, JUST TO simplify the codebase, then you may run into this problem.

For example: We have a type Widget:

```
public class DOWidget_v1 : DomainBase_v1
{
    public string Name { get; protected set; }
}
```

And, you derive from it to create a SuperWidget:

```
public class DOSuperWidget_v1 : DOWidget_v1
{
    public string Description { get; protected set; }

    public Guid? ParentId { get; protected set; }
}
```

As far as C# is concerned, this is fine, and it deduplicates properties with inheritance.

And, you can define independent DbSet properties on your Db context, to access both types.

HOWEVER

EF Core sees this inheritance as a possible shared table... of the base type.

And, it throws an exception, because you haven't chosen to tell EF how to treat the inheritance.

The exception will be something like this:

```
Initialization method TestAppDomain_InMem_Tests.TestDbContext_Tests.Setup threw exception.
System.InvalidOperationException: A key cannot be configured on 'DOSuperWidget_v1' because it is a derived type.
The key must be configured on the root type 'DOWidget_v1'.
If you did not intend for 'DOWidget_v1' to be included in the model, ensure that it is not referenced by a DbSet property on your context, referenced in a configuration call to ModelBuilder, or referenced from a navigation on a type that is included in the model.
```

So, you have to tell EF core that the derived type does not have a base.

To do so, add this line to the top of your mapping class, like this:

```
public class DOSuperWidget_v1MapConfig : IEntityTypeConfiguration<DOSuperWidget_v1>
{
    public void Configure(EntityTypeBuilder<DOSuperWidget_v1> builder)
    {
        // NOTE: We only need this statement if we want DOSuperWidget to be a separate root entity from
DOWidget.
        // Our SuperWidget inherits from Widget, just to simplify the codebase.
        // It does NOT inherit, so that the two types share a common table.
        // So, we must tell EF to not treate SuperWidget as a derived type of Widget.
        // Break EF inheritance here...
        builder.Metadata.BaseType = null;
```

Once the above line is included in your mapping Configure() method, EF will know that your derived type has no base to concern with.

And, EF Core will treat the derived type as an independent type from its base.

Revision #1
Created 14 November 2025 04:00:34 by glwhite
Updated 14 November 2025 04:08:26 by glwhite