Table Per Hierarchy Inheritance

Entity Framework Core will represent an object-oriented hierarchy in a single table that takes the name of the base class and includes a "discriminator" column to identify the specific type for each row. EF Core will only include types that are explicitly included in the model and will only implement the discriminator column if two or more types (including the base type) have a `DbSet` configured for them.

The Model

Here is the model being used to illustrate inheritance in EF Core. It represents the type of contracts that you might enter into with a media/communications business.

```
public class Contract
{
    public int ContractId { get; set; }
    public DateTime StartDate { get; set; }
    public int Months { get; set;}
    public decimal Charge { get; set; }
}
public class MobileContract : Contract
{
    public string MobileNumber { get; set; }
}
public class TvContract : Contract
{
    public PackageType PackageType { get; set; }
}
public class BroadBandContract : Contract
{
    public int DownloadSpeed { get; set; }
}
public enum PackageType
{
    S, M, L, XL
}
```

Convention

Entity Framework Core offers two approaches to configuring the model so that the framework recognises that an inheritance hierachy exists. The first is to explicitly include `DbSet` properties in the `DbContext` for the base class and each of the derived types:

```
public class SampleContext : DbContext
{
    public DbSet<Contract> Contracts { get; set; }
    public DbSet<MobileContract> MobileContracts { get; set; }
    public DbSet<TvContract> TvContracts { get; set; }
    public DbSet<BroadBandContract> BroadBandContracts { get; set; }
}
```

This approach has the benefit of enabling you to easily query contract types explicitly e.g.

```
var mobileContracts = context.MobileContracts.ToList();
```

An alternative is to incldue a `DbSet` for the base class and then implicitly include the derived entities in the `OnModelCreating` method:

```
public class SampleContext : DbContext
{
    public DbSet<Contract> Contracts { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<MobileContract>();
        modelBuilder.Entity<TvContract>();
        modelBuilder.Entity<BroadBandContract>();
    }
}
```

If you adopt this approach, you will need to use the `OfType<TEntity>` method to query for contracts of a particular type:

```
var tvContracts = context.Contracts.OfType<TvContract>().ToList();
```

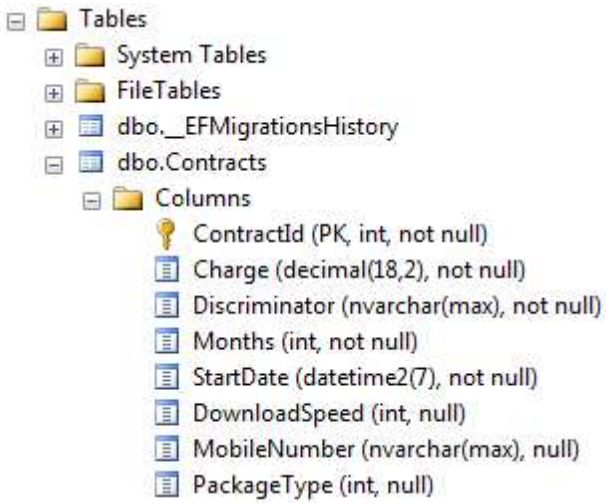
Abstract base class

If instances of the base class are not intended to be instantiated, it would be more proper to declare the base class as `abstract` and to configure that in the `OnModelCreating` method while adding `DbSet` properties to the `DbContext` for the derived classes:

```
public class SampleContext : DbContext
{
    public DbSet<MobileContract> MobileContracts { get; set; }
    public DbSet<TvContract> TvContracts { get; set; }
    public DbSet<BroadbandContract> BroadbandContracts { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Contract>().ToTable("Contracts");
    }
}

public abstract class Contract
{
    public int ContractId { get; set; }
    public DateTime StartDate { get; set; }
    public int Months { get; set;}
    public decimal Charge { get; set; }
}
public class MobileContract : Contract
{
    public string MobileNumber { get; set; }
}
public class TvContract : Contract
{
    public PackageType PackageType { get; set; }
}
public class BroadbandContract : Contract
{
    public int DownloadSpeed { get; set; }
}
public enum PackageType
{
    S, M, L, XL
}
```

Each of the approaches outlined above will result in the same table structure being created:



The table includes fields for the properties of the base `Contract` class, and for the properties in the derived classes. It also includes a column named "Discriminator", designed to identity the derived type that the data row represents. By default, EF Core will use the name of the class as a value for the discriminator column. So the value for a row that contains e.g. a TvContract will be "TvContract".

Configuration

You can configure aspects of the inheritance mapping via the Fluent API. The example above shows how to configure the table name for the abstract base class that didn't have a `DbSet` included in the `DbContext` using the **ToTable method (/configuration/fluent-api/totable-method)**. In addition, you can specify a name and data type for the discriminator column, and the values to represent each type:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Contract>()
            .ToTable("Contracts")
            .HasDiscriminator<int>("ContractType")
            .HasValue<MobileContract>(1)
            .HasValue<TvContract>(2)
            .HasValue<BroadbandContract>(3);
}
```

In this example, the data type for the discriminator column is specified as an `int`, and named "ContractType". If you choose to provide your own values for the discriminator column, you must provide values for **all** non-abstract entities in the mapped hierarchy. The `Contract` entity in this example is abstract so it hasn't been included. But if the base type is not defined as `abstract`, it must have a discriminator value provided.

On this page

Latest Updates

Package Manager Console Commands
(https://www.learnentityframeworkcore.com/migrations/commands/pmc-commands)

Command Line Interface commands
(https://www.learnentityframeworkcore.com/migrations/commands/cli-commands)

Commands in Entity Framework Core
(https://www.learnentityframeworkcore.com/migrations/commands)

The Fluent API ValueGeneratedNever Method
(https://www.learnentityframeworkcore.com/configuration/fluent-api/valuegeneratednever-method)

The Fluent API WithMany Method (https://www.learnentityframeworkcore.com/configuration/fluent-api/withmany-method)

The Fluent API WithOne Method (https://www.learnentityframeworkcore.com/configuration/fluent-api/withone-method)