Customizing ASP.NET Core Identity Tables

2

5

When man customize the Identity classes in asp.net core 2, the relations between tables do not create automatically.

How can I create relations between tables such a simplest way?

User class:

```
public class User : IdentityUser<int>
{
    // codes
}
```

Role class:

```
public class Role : IdentityRole<int>
{
    public Role() : base()
    {

    }
    public Role(string roleName) : base(roleName)
    {

    }
}
```

RoleClaim class:

```
public class RoleClaim : IdentityRoleClaim<int> { }
```

UserClaim class:

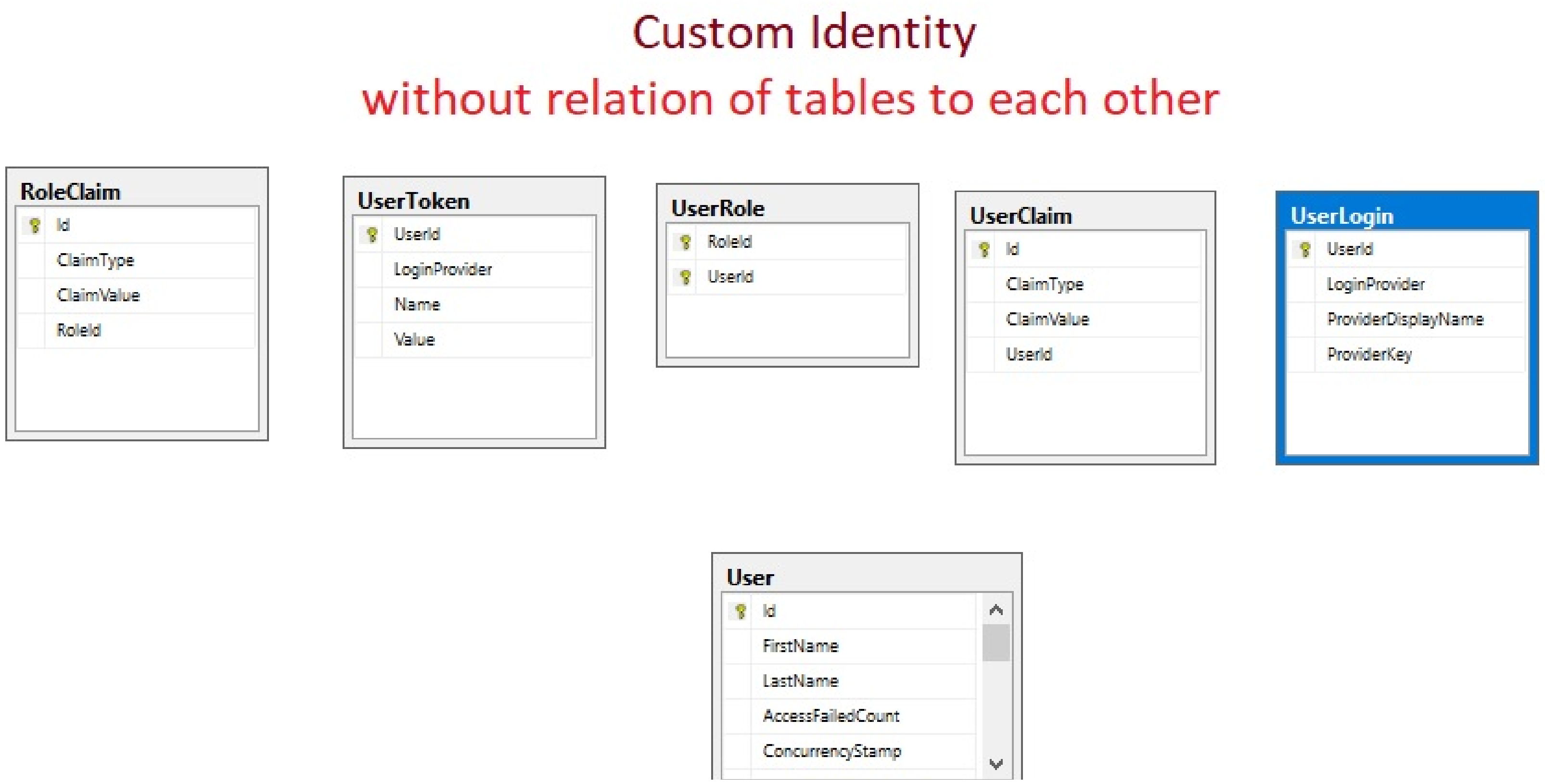
```
public class UserLogin : IdentityUserLogin<int> { }
```

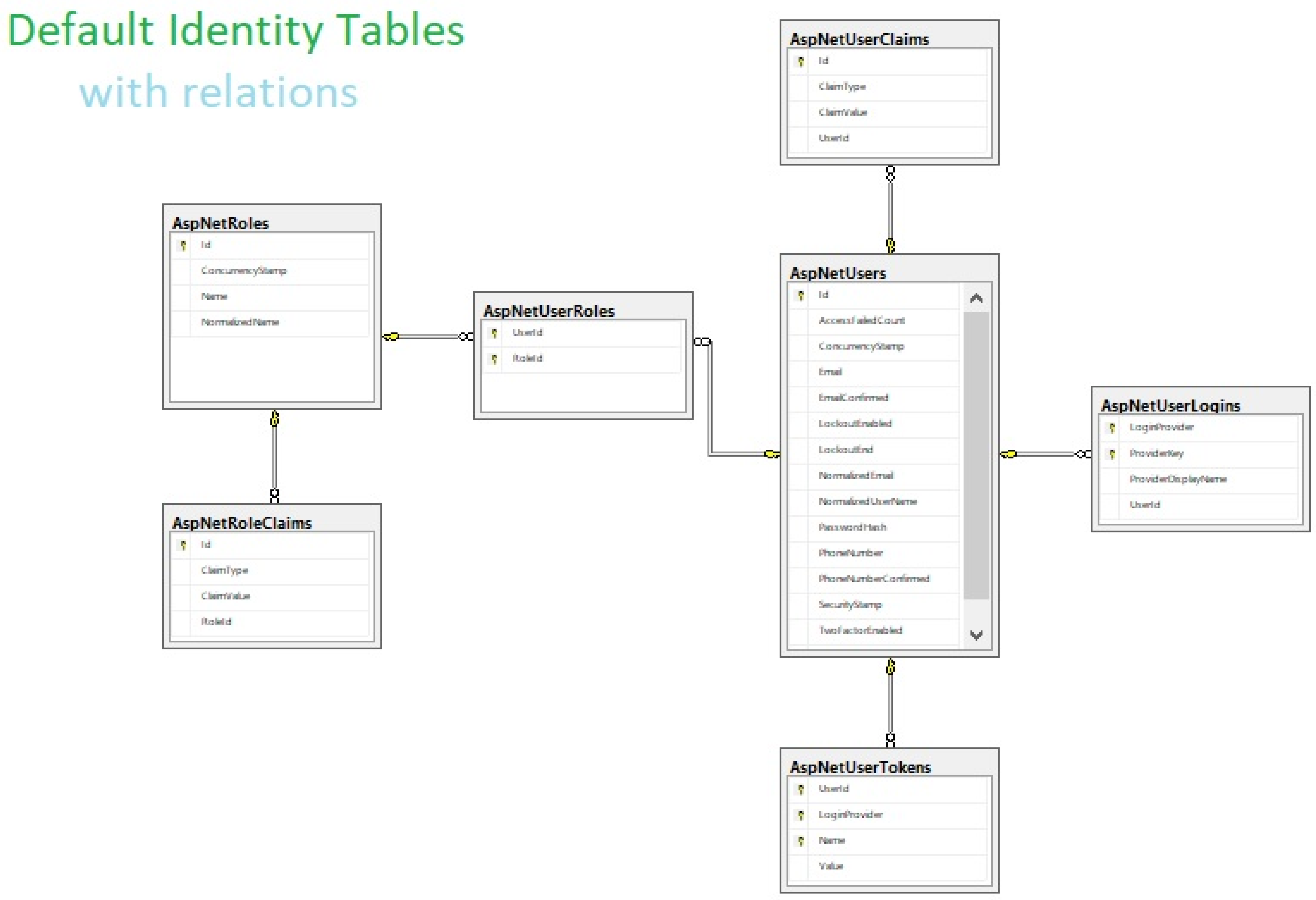
UserRole class:

```
public class UserRole : IdentityUserRole<int> { }
```

UserToken class:

```
public class UserToken : IdentityUserToken<int> { }
```





asp.net asp.net-core asp.net-identity asp.net-core-2.0 ef-migrations

edited Mar 27 '18 at 13:20 asked Mar 25 '18 at 1:46

Join Stack Overflow to learn, share knowledge, and build your career.

Email Sign Up OR SIGN IN WITH Google Facebook

2 Answers

5

I've defined the relations of tables by writing some codes in `User` , `Role` , `UserClaim` , `UserRole` , `UserLogin` , `RoleClaim` and `UserToken` class and in `ApplicationDbContext` class.

Role class:

```
public class Role : IdentityRole<int>
{
    public Role() : base()
    {

    }
    public Role(string roleName) : this()
    {
        Name = roleName;
    }

    public virtual ICollection<UserRole> Users { get; set; }
    public virtual ICollection<RoleClaim> Claims { get; set; }
}
```

RoleClaim class:

```
public class RoleClaim : IdentityRoleClaim<int>
{
    public virtual Role Role { get; set; }
}
```

User class:

```
public class User : IdentityUser<int>
{
    public virtual ICollection<UserToken> UserTokens { get; set; }
```

UserClaim class:

```
public class UserClaim : IdentityUserClaim<int>
{
    public virtual User User { get; set; }
}
```

UserLogin class:

```
public class UserLogin : IdentityUserLogin<int>
{
    public virtual User User { get; set; }
}
```

UserRole class:

```
public class UserRole : IdentityUserRole<int>
{
    public virtual User User { get; set; }
    public virtual Role Role { get; set; }
}
```

UserToken class:

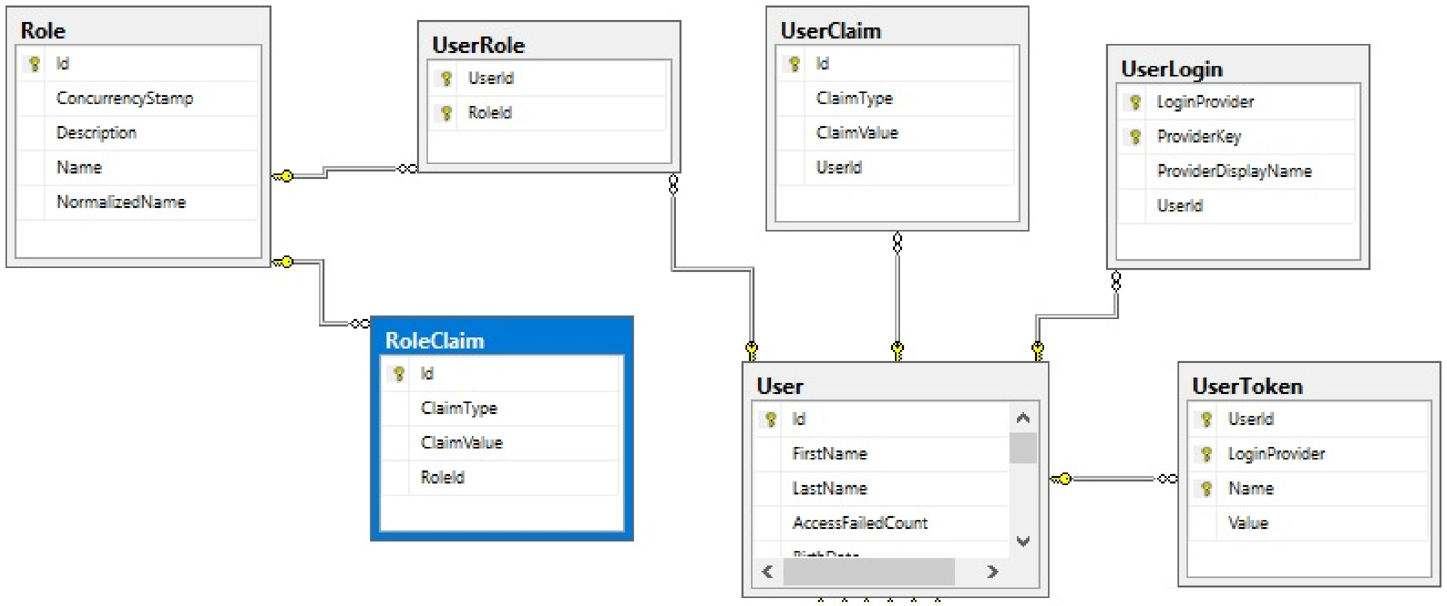
```
public class UserToken : IdentityUserToken<int>
{
    public virtual User User { get; set; }
}
```

ApplicationDbContext class:

```
public class ApplicationDbContext : IdentityDbContext<User, Role, int, UserClaim,
UserRole, UserLogin, RoleClaim, UserToken>
{
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<RoleClaim>(builder =>
        {
            builder.HasOne(roleClaim => roleClaim.Role).WithMany(role =>
role.Claims).HasForeignKey(roleClaim => roleClaim.RoleId);
            builder.ToTable("RoleClaim");
        });
        modelBuilder.Entity<Role>(builder =>
        {
            builder.ToTable("Role");
        });
        modelBuilder.Entity<UserClaim>(builder =>
        {
            builder.HasOne(userClaim => userClaim.User).WithMany(user =>
user.Claims).HasForeignKey(userClaim => userClaim.UserId);
            builder.ToTable("UserClaim");
        });
        modelBuilder.Entity<UserLogin>(builder =>
        {
            builder.HasOne(userLogin => userLogin.User).WithMany(user =>
user.Logins).HasForeignKey(userLogin => userLogin.UserId);
            builder.ToTable("UserLogin");
        });
        modelBuilder.Entity<User>(builder =>
        {
            builder.ToTable("User");
        });
        modelBuilder.Entity<UserRole>(builder =>
        {
            builder.HasOne(userRole => userRole.Role).WithMany(role =>
role.Users).HasForeignKey(userRole => userRole.RoleId);
            builder.HasOne(userRole => userRole.User).WithMany(user =>
user.Roles).HasForeignKey(userRole => userRole.UserId);
            builder.ToTable("UserRole");
        });
        modelBuilder.Entity<UserToken>(builder =>
        {
            builder.HasOne(userToken => userToken.User).WithMany(user =>
user.UserTokens).HasForeignKey(userToken => userToken.UserId);
            builder.ToTable("UserToken");
```

You can see the result in this picture:



answered Mar 26 '18 at 23:09

Jahan

3,605 6 33 76

```
public class MyContext : IdentityDbContext<User, CustomRole, int, CustomUserClaim,
CustomUserRole, CustomUserLogin, CustomRoleClaim, CustomUserLogin>
```

answered Mar 26 '18 at 16:06

Chris Pratt
167k 22 256 322

I did it already but that is not enough. I found the solution! – Jahan Mar 26 '18 at 23:17