**Configurar y asignar propiedades y tipos con la API fluida**

Al trabajar con Code First de Entity Framework, el comportamiento predeterminado es asignar las clases POCO a las tablas mediante un conjunto de convenciones establecidas en EF. Sin embargo, en ocasiones no puede o no desea seguir dichas convenciones y tiene que asignar entidades a elementos distintos de los que las convenciones dictan.

Hay dos métodos principales para configurar EF de modo que use otro criterio diferente a las convenciones al asignar los datos, a saber: las [anotaciones](https://msdn.microsoft.com/es-ar/data/jj591583) o la API fluida de EF. Las anotaciones solo abarcan un subconjunto de la funcionalidad de la API fluida, por lo que hay escenarios de asignación que no se pueden lograr con anotaciones. Este artículo está diseñado para demostrar cómo usar la API fluida para configurar propiedades.

Además de para controlar la asignación, la API fluida y las anotaciones también se pueden usar para configurar las restricciones como la longitud de campo o el atributo required; cuando estas restricciones se configuran, afectará a la base de datos creada por Code First así como a la validación que se efectúe con Entity Framework.

**Introducción**

Se suele tener acceso a la API fluida de Code First invalidando el método [de OnModelCreating](https://msdn.microsoft.com/library/system.data.entity.dbcontext.onmodelcreating(v=vs.103).aspx) en el [DbContext](https://msdn.microsoft.com/library/system.data.entity.dbcontext(v=vs.103).aspx) derivado. Los ejemplos siguientes están diseñados para mostrar cómo realizar varias tareas con la API fluida y permitirle copiar el código y personalizarlo para adaptarlo a su modelo; si desea ver el modelo con el que se pueden usar tal cual, se proporciona al final de este artículo.

**Asignación de propiedad**

El método [Property](https://msdn.microsoft.com/library/system.data.entity.infrastructure.dbentityentry.property(v=vs.103).aspx) se usa para configurar los atributos de cada propiedad que pertenece a una entidad o un tipo complejo. El método Property se usa para obtener un objeto configuration para una propiedad determinada. Las opciones del objeto configuration son específicas del tipo que se configura; IsUnicode solo está disponible en las propiedades de cadena, por ejemplo.

**Configurar una clave principal**

La convención de Entity Framework para las claves principales es:

1.       La clase define una propiedad cuyo nombre es “ID” o “Id”

2.       o un nombre de clase seguido de “ID” o “Id”

Para establecer explícitamente que una propiedad sea una clave principal, puede usar el método HasKey. En el ejemplo siguiente, el método HasKey se usa para configurar la clave principal InstructorID en el tipo OfficeAssignment.

modelBuilder.Entity<OfficeAssignment>().HasKey(t => t.InstructorID);

**Configurar una clave principal compuesta**

En el siguiente ejemplo se configuran las propiedades DepartmentID y Name para ser la clave principal compuesta del tipo Department.

modelBuilder.Entity<Department>().HasKey(t => **new** { t.DepartmentID, t.Name });

**Cambiar la identidad para las claves principales numéricas**

El ejemplo siguiente establece la propiedad DepartmentID en System.ComponentModel.DataAnnotations.DatabaseGeneratedOption.None para indicar que la base de datos no generará el valor.

modelBuilder.Entity<Department>().Property(t => t.DepartmentID)   
    .HasDatabaseGeneratedOption (DatabaseGeneratedOption.None);

**Especificar la longitud máxima en una propiedad**

En el ejemplo siguiente, la propiedad Name no debe tener más de 50 caracteres. Si hace que el valor tenga más de 50 caracteres, obtendrá una excepción[DbEntityValidationException](https://msdn.microsoft.com/library/system.data.entity.validation.dbentityvalidationexception(v=vs.103).aspx). Si Code First crea una base de datos a partir de este modelo, también establecerá la longitud máxima de la columna Name a 50 caracteres.

modelBuilder.Entity<Department>().Property(t => t.Name).HasMaxLength(50);

**Configurar la propiedad para que sea required**

En el ejemplo siguiente, se requiere la propiedad Name. Si no especifica Name, obtendrá una excepción DbEntityValidationException. Si Code First crea una base de datos a partir de este modelo, la columna usa da para almacenar esta propiedad no admitirá valores NULL.

modelBuilder.Entity<Department>().Property(t => t.Name).IsRequired();

**Especificar No asignar una propiedad de CLR a una columna de la base de datos**

En el siguiente ejemplo se muestra cómo especificar que una propiedad de un tipo CLR no está asignada a una columna de la base de datos.

modelBuilder.Entity<Department>().Ignore(t => t.Budget);

**Asignar una propiedad de CLR a una columna específica de la base de datos**

El ejemplo siguiente asigna la propiedad Name CLR a la columna de base de datos DepartmentName.

modelBuilder.Entity<Department>()   
    .Property(t => t.Name)   
    .HasColumnName("DepartmentName");

**Cambiar el nombre de una clave externa que no está definida en el modelo**

Si decide no definir una clave externa en un tipo CLR pero desea especificar qué nombre debe tener en la base de datos, haga lo siguiente:

modelBuilder.Entity<Course>()   
    .HasRequired(c => c.Department)   
    .WithMany(t => t.Courses)   
    .Map(m => m.MapKey("ChangedDepartmentID"));

**Configurar si la propiedad de cadena admite o no contenido Unicode**

De forma predeterminada, las cadenas son Unicode (nvarchar en SQL Server). Puede usar el método IsUnicode para especificar que una cadena debe ser de tipo varchar.

modelBuilder.Entity<Department>()   
    .Property(t => t.Name)   
    .IsUnicode(**false**);

**Configurar el tipo de datos de una columna de base de datos**

El método [HasColumnType](https://msdn.microsoft.com/library/system.data.entity.modelconfiguration.configuration.stringcolumnconfiguration.hascolumntype(v=vs.103).aspx) permite la asignación a varias representaciones del mismo tipo básico. Con este método, no puede realizar ninguna conversión de datos en tiempo de ejecución. Tenga en cuenta que IsUnicode es la mejor manera de establecer las columnas en varchar, ya que no depende de la base de datos.

modelBuilder.Entity<Department>()     
    .Property(p => p.Name)     
    .HasColumnType("varchar");

**Configurar las propiedades en un tipo complejo**

Hay dos maneras de configurar las propiedades escalares en un tipo complejo.

Puede llamar a Property en ComplexTypeConfiguration.

modelBuilder.ComplexType<Details>()   
    .Property(t => t.Location)   
    .HasMaxLength(20);

También puede usar la notación de puntos para tener acceso a una propiedad de tipo complejo.

modelBuilder.Entity<OnsiteCourse>()   
    .Property(t => t.Details.Location)   
    .HasMaxLength(20);

**Configurar una propiedad para que sea usa da como un token de simultaneidad optimista**

Para especificar que una propiedad de una entidad representa un token de simultaneidad, puede usar el atributo ConcurrencyCheck o el método IsConcurrencyToken.

modelBuilder.Entity<OfficeAssignment>()   
    .Property(t => t.Timestamp)   
    .IsConcurrencyToken();

También puede usar el método IsRowVersion para configurar que la propiedad sea una versión de fila en la base de datos. Si se establece la propiedad para que sea una versión de fila, se configura automáticamente para que sea un token de simultaneidad optimista.

modelBuilder.Entity<OfficeAssignment>()   
    .Property(t => t.Timestamp)   
    .IsRowVersion();

**Asignación de tipos**

**Especificar que una clase es de un tipo complejo**

Por convención, un tipo que no tenga ninguna clave principal especificada se trata como un tipo complejo. Hay algunos escenarios en los que Code First no detectará un tipo complejo (por ejemplo, si tiene una propiedad denominada ID pero no desea con ello que sea una clave principal). En casos como este, usaría la API fluida para especificar explícitamente que un tipo es complejo.

modelBuilder.ComplexType<Details>();

**Especificar No asignar un tipo de entidad CLR a una tabla de la base de datos**

En el ejemplo siguiente se muestra cómo excluir un tipo CLR para que no se asigne a una tabla de la base de datos.

modelBuilder.Ignore<OnlineCourse>();

**Asignar un tipo de entidad CLR a una tabla específica de la base de datos**

Todas las propiedades de Department se asignarán a columnas de la tabla denominada t\_ Department.

modelBuilder.Entity<Department>()    
    .ToTable("t\_Department");

También puede especificar el nombre de esquema de esta forma:

modelBuilder.Entity<Department>()    
    .ToTable("t\_ Department", "school");

**Asignar la herencia de tabla por jerarquía (TPH)**

En este escenario de asignación TPH, todos los tipos de una jerarquía de herencia se asignan a una única tabla. Una columna de discriminador se usa para identificar el tipo de cada fila. Al crear el modelo con Code First, TPH es la estrategia predeterminada para los tipos que participan en la jerarquía de herencia. De forma predeterminada, la columna de discriminador se agrega a la tabla con el nombre “Discriminator” y el nombre de tipo CLR de cada tipo de la jerarquía se usa para los valores de discriminador. Puede modificar el comportamiento predeterminado mediante la API fluida.

modelBuilder.Entity<Course>()    
    .Map<Course>(m => m.Requires("Type").HasValue("Course"))    
    .Map<OnsiteCourse>(m => m.Requires("Type").HasValue("OnsiteCourse"));

**Asignar la herencia de tabla por tipo (TPT)**

En este escenario de asignación TPT, todos los tipos se asignan a tablas individuales. Las propiedades que pertenecen solamente a un tipo base o a un tipo derivado se almacenan en una tabla que se asigna a ese tipo. Las tablas que se asignan a tipos derivados también almacenan una clave externa que une la tabla derivada con la tabla base.

modelBuilder.Entity<Course>().ToTable("Course");    
modelBuilder.Entity<OnsiteCourse>().ToTable("OnsiteCourse");

**Asignar la herencia de tabla por clase concreta (TPC)**

En este escenario de asignación TPC, todos los tipos no abstractos de la jerarquía se asignan a tablas individuales. Las tablas que se asignan a las clases derivadas no tienen ninguna relación con la tabla que se asigna a la clase base de la base de datos. Todas las propiedades de una clase, incluidas las propiedades heredadas, se asignan a las columnas de la tabla correspondiente,

Llame al método MapInheritedProperties para configurar cada tipo derivado. MapInheritedProperties reasigna todas las propiedades que se heredaron de la clase base a las nuevas columnas de la tabla para la clase derivada.

**Nota:** observe que, ya que las tablas que participan en la jerarquía de herencia TPC no comparten una clave principal, habrá claves de entidad duplicadas al insertar en las tablas que se asignan a las subclases si tiene valores generados de la base de datos con la misma inicialización de identidad. Para solucionar este problema, puede especificar un valor distinto de inicialización para cada tabla o cambiar la identidad en la propiedad de clave principal. La identidad es el valor predeterminado para las propiedades de clave entera al trabajar con Code First.

modelBuilder.Entity<Course>()   
    .Property(c => c.CourseID)   
    .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);   
   
modelBuilder.Entity<OnsiteCourse>().Map(m =>   
{   
    m.MapInheritedProperties();   
    m.ToTable("OnsiteCourse");   
});   
   
modelBuilder.Entity<OnlineCourse>().Map(m =>   
{   
    m.MapInheritedProperties();   
    m.ToTable("OnlineCourse");   
});

**Asignar las propiedades de CLR de un tipo de entidad a varias tablas de la base de datos (división de entidades)**

La división de entidades permite a las propiedades de un tipo de entidad extenderse a través de varias tablas. En el siguiente ejemplo, la entidad Department se divide en dos tablas: Department y DepartmentDetails. La división de entidades usa varias llamadas al método Map para asignar un subconjunto de propiedades a una tabla concreta.

modelBuilder.Entity<Department>()   
    .Map(m =>   
    {   
        m.Properties(t => **new** { t.DepartmentID, t.Name });   
        m.ToTable("Department");   
    })   
    .Map(m =>   
    {   
        m.Properties(t => **new** { t.DepartmentID, t.Administrator, t.StartDate, t.Budget });   
        m.ToTable("DepartmentDetails");   
    });

**Asignar varios tipos de entidades a una tabla en la base de datos (división por tabla)**

El ejemplo siguiente asigna dos tipos de entidad que comparten una clave principal a una tabla.

modelBuilder.Entity<OfficeAssignment>()   
    .HasKey(t => t.InstructorID);   
               
modelBuilder.Entity<Instructor>()   
    .HasRequired(t => t.OfficeAssignment)   
    .WithRequiredPrincipal(t => t.Instructor);   
   
modelBuilder.Entity<Instructor>().ToTable("Instructor");   
   
modelBuilder.Entity<OfficeAssignment>().ToTable("Instructor");

**Modelo usado en ejemplos**

El modelo de Code First siguiente se usa para los ejemplos de esta página.

**using** System.Data.Entity;   
**using** System.Data.Entity.ModelConfiguration.Conventions;   
// add a reference to System.ComponentModel.DataAnnotations DLL   
**using** System.ComponentModel.DataAnnotations;   
**using** System.Collections.Generic;   
**using** System;   
   
**public** **class** SchoolEntities : DbContext   
{   
    **public** DbSet<Course> Courses { get; set; }   
    **public** DbSet<Department> Departments { get; set; }   
    **public** DbSet<Instructor> Instructors { get; set; }   
    **public** DbSet<OfficeAssignment> OfficeAssignments { get; set; }   
   
    **protected** **override** **void** OnModelCreating(DbModelBuilder modelBuilder)   
    {   
        // Configure Code First to ignore PluralizingTableName convention   
        // If you keep this convention then the generated tables will have pluralized names.   
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();   
    }   
}   
   
**public** **class** Department   
{   
    **public** Department()   
    {   
        **this**.Courses = **new** HashSet<Course>();   
    }   
    // Primary key   
    **public** **int** DepartmentID { get; set; }   
    **public** **string** Name { get; set; }   
    **public** **decimal** Budget { get; set; }   
    **public** System.DateTime StartDate { get; set; }   
    **public** **int**? Administrator { get; set; }   
   
    // Navigation property   
    **public** **virtual** ICollection<Course> Courses { get; **private** set; }   
}   
   
**public** **class** Course   
{   
    **public** Course()   
    {   
        **this**.Instructors = **new** HashSet<Instructor>();   
    }   
    // Primary key   
    **public** **int** CourseID { get; set; }   
   
    **public** **string** Title { get; set; }   
    **public** **int** Credits { get; set; }   
   
    // Foreign key   
    **public** **int** DepartmentID { get; set; }   
   
    // Navigation properties   
    **public** **virtual** Department Department { get; set; }   
    **public** **virtual** ICollection<Instructor> Instructors { get; **private** set; }   
}   
   
**public** partial **class** OnlineCourse : Course   
{   
    **public** **string** URL { get; set; }   
}   
   
**public** partial **class** OnsiteCourse : Course   
{   
    **public** OnsiteCourse()   
    {   
        Details = **new** Details();   
    }   
   
    **public** Details Details { get; set; }   
}   
   
**public** **class** Details   
{   
    **public** System.DateTime Time { get; set; }   
    **public** **string** Location { get; set; }   
    **public** **string** Days { get; set; }   
}   
       
**public** **class** Instructor   
{   
    **public** Instructor()   
    {   
        **this**.Courses = **new** List<Course>();   
    }   
   
    // Primary key   
    **public** **int** InstructorID { get; set; }   
    **public** **string** LastName { get; set; }   
    **public** **string** FirstName { get; set; }   
    **public** System.DateTime HireDate { get; set; }   
   
    // Navigation properties   
    **public** **virtual** ICollection<Course> Courses { get; **private** set; }   
}   
   
**public** **class** OfficeAssignment   
{   
    // Specifying InstructorID as a primary   
    [Key()]   
    **public** Int32 InstructorID { get; set; }   
   
    **public** **string** Location { get; set; }   
   
    // When the Entity Framework sees Timestamp attribute   
    // it configures ConcurrencyCheck and DatabaseGeneratedPattern=Computed.   
    [Timestamp]   
    **public** Byte[] Timestamp { get; set; }   
   
    // Navigation property   
    **public** **virtual** Instructor Instructor { get; set; }   
}