Relación de 1 a varios

```
class Cliente
{
    public int ClienteId { get; set; }
    public string NombreCliente { get; set; }
    public ICollection<Direccion> Direcciones { get; set; }
}

[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
}
```

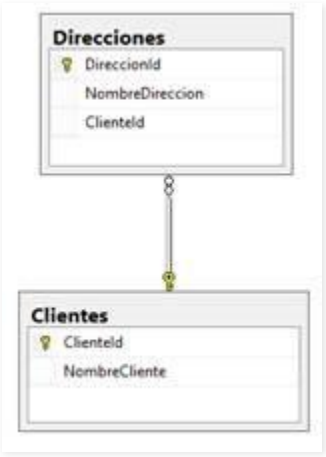
En este ejemplo podemos ver que se creó el campo *Cliente_ClienteId* porque CF no encontró en la tabla Direcciones ningún campo donde poder guardar la clave externa a Clientes. En realidad, CF buscó alguno de los siguientes campos:

- [Target Type Key Name]
- [Target Type Name] + [Target Type Key Name]

Es decir, si hubiera existido la propiedad *Direccion.ClienteId* o *Direccion.ClienteClienteId*se hubiera utilizado esa propiedad en vez de crear una nueva. Además, también puedes ver en este ejemplo que no es necesario que una propiedad de navegación sea bidireccional, por ejemplo desde Direccion no se puede navegar a Cliente y no hay ningún problema.

Para solucionar el problema anterior, agregaremos el campo ClienteId a Direccion:

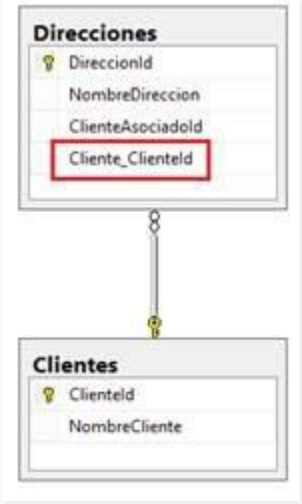
```
[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    public int ClienteId { get; set; }
}
```



La verdad es que siempre es una buena idea crear una propiedad de clave externa, además de la propiedad de navegación. Además de para resolver el mapeo, en operaciones de manipulación de datos, podremos simplemente asignar la propiedad de clave externa sin tener que disponer de un objeto para asignar a la propiedad de navegación. Es decir, ¿Qué es más intuitivo? *Direccion.ClienteId = 5* o tener que recuperar un cliente y asignarlo a la propiedad de navegación como *Direccion.Cliente = unCliente*.

Otro escenario que suele ocurrir es que la propiedad de clave externa exista pero su nombre no coincida con ninguna convención. Por ejemplo:

```
[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    public int ClienteAsociadoId { get; set; }
}
```



La solución pasa por la utilización del atributo ForeignKey.

```
class Cliente
{
    public int ClienteId { get; set; }
    public string NombreCliente { get; set; }
    [ForeignKey("ClienteAsociadoId")]
    public ICollection<Direccion> Direcciones { get; set; }
}
```

Si queremos resolver esto mismo con Fluent API, estamos obligados a crear una propiedad de navegación Cliente en la clase Direccion porque para poder especificar el comportamiento de una relación con Fluent API tenemos que definir la relación entera.

```
[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    public int ClienteAsociadoId { get; set; }
    public Cliente Cliente { get; set; }
}

class ConvencionesEF : DbContext
{
    public DbSet<Cliente> Clientes { get; set; }
    protected override voidOnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Cliente>().
            HasMany(p => p.Direcciones).
            WithRequired(p => p.Cliente).
            HasForeignKey(p=>p.ClienteAsociadoId);
    }
}
```

Ahora que ya hemos introducido la propiedad de navegación Cliente en Direccion, también podemos utilizar DataAnnotations (en vez de Fluent API) para especificar la propiedad de clave externa:

```
class Cliente
{
    public int ClienteId { get; set; }
    public string NombreCliente { get; set; }
    public ICollection<Direccion> Direcciones { get; set; }
```

```
}


[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    [ForeignKey("Cliente")]
    public int ClienteAsociadoId { get; set; }
    public Cliente Cliente { get; set; }
}
```

También sería válido

```
[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    public int ClienteAsociadoId { get; set; }
    [ForeignKey("ClienteAsociadoId")]
    public Cliente Cliente { get; set; }
}
```

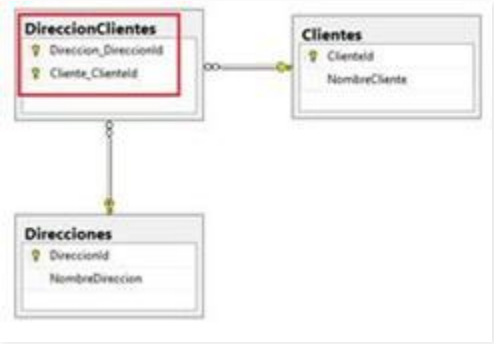
Incluso ambas 2 a la vez tampoco hay problema (aunque es redundante):

```
[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    [ForeignKey("Cliente")]
    public int ClienteAsociadoId { get; set; }
    [ForeignKey("ClienteAsociadoId")]
    public Cliente Cliente { get; set; }
}
```

Relación de muchos a muchos

En esta situación, la tabla intermedia (que no acepta ningún atributo más excepto las claves primarias de ambas tablas), se creará automáticamente por nosotros cuando CF detecte 2 propiedades de navegación de colección entre 2 entidades:

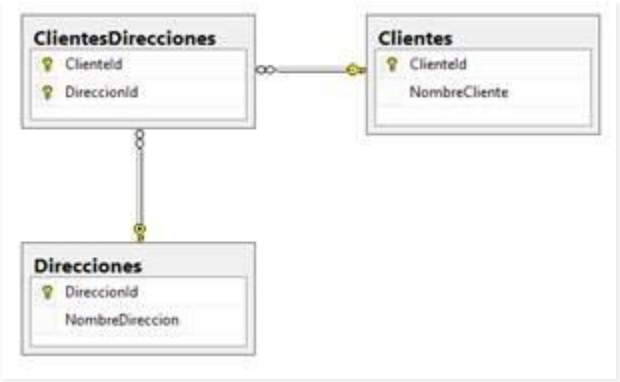
```
class Cliente
{
    public int ClienteId { get; set; }
    public string NombreCliente { get; set; }
    public ICollection<Direccion>
Direcciones { get; set; }
}


[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    public ICollection<Cliente>
Clientes { get; set; }
}
```



Si queremos controlar la tabla que guardará nuestra relación de varios a varios, así como los nombres de los campos creados, tendremos que utilizar Fluent API:

```
modelBuilder.Entity<Cliente>().
    HasMany(p => p.Direcciones).
    WithMany(p => p.Clientes)
    .Map(p => p.ToTable("ClientesDirecciones").
        MapLeftKey("ClienteId").
```

```
        MapRightKey("DireccionId"));
```



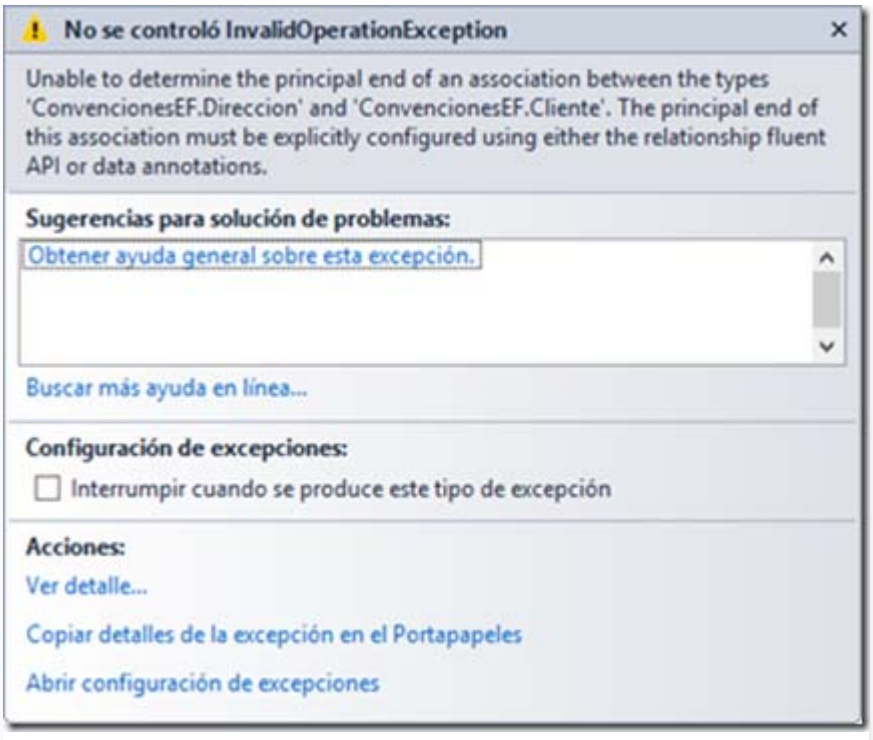
Relación de 1 a 1

En este tipo de relaciones siempre se requiere configuración extra porque CF no es capaz de inferir quién es el lado principal y quien el lado dependiente en la relación.

Con este código:

```
class Cliente
{
    public int ClienteId { get; set; }
    public string NombreCliente { get; set; }
    public int DireccionId { get; set; }
    public Direccion Direccion { get; set; }
}


[Table("Direcciones")]
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    public int ClienteId { get; set; }
    public Cliente  Cliente { get; set; }
}
```

Obtendremos la siguiente excepción:



Para resolver esto, lo más sencillo es utilizar el atributo *ForeignKey* en la tabla dependiente. Además CF requiere que la propiedad de clave externa sea también clave primaria en la tabla dependiente.

```
    [Table("Direcciones")]
```

```
class Direccion
{
    public int DireccionId { get; set; }
    public string NombreDireccion { get; set; }
    [Key]
    [ForeignKey("Cliente")]
    public int? ClienteId { get; set; }
    public Cliente  Cliente { get; set; }
}
```