In **Entity Framework Core**, cascade delete is enabled by default for **required relationships** (**`DeleteBehavior.Cascade`**) and set to **`ClientSetNull`** for optional relationships.
If you want to **disable cascade delete globally** or for specific relationships, you can configure it in **`OnModelCreating`**.

1️⃣ Disable Cascade Delete Globally

You can loop through all foreign keys in the model and set their **`DeleteBehavior`** to **`Restrict`** (or **`NoAction`** in EF Core 5+):

```
using Microsoft.EntityFrameworkCore;
using System.Linq;

public class MyDbContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Disable cascade delete globally
        foreach (var foreignKey in modelBuilder.Model
                     .GetEntityTypes()
                     .SelectMany(e => e.GetForeignKeys()))
        {
            foreignKey.DeleteBehavior = DeleteBehavior.Restrict; // or DeleteBehavior.NoAction
        }
```

```
    }
}

public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Order> Orders { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public User User { get; set; }
}
```

2️⃣ Disable Cascade Delete for a Specific Relationship

If you only want to disable it for certain relationships:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Order>()
        .HasOne(o => o.User)
```

```
        .WithMany(u => u.Orders)
        .HasForeignKey(o => o.UserId)
        .OnDelete(DeleteBehavior.Restrict); // or DeleteBehavior.NoAction
}
```

3️⃣ Notes

- **`DeleteBehavior.Restrict`** → Prevents deletion if related entities exist (throws exception).
- **`DeleteBehavior.NoAction`** → Lets the database decide (often same as **`Restrict`** in SQL Server).
- **`DeleteBehavior.ClientSetNull`** → Sets FK to **`null`** in memory, requires nullable FK.
- **`DeleteBehavior.Cascade`** → Deletes dependents automatically.

✅ Recommendation:

If you want to **completely prevent accidental cascading deletes**, use **`Restrict`** globally as in example 1.
If you just want to control specific relationships, use example 2.

If you want, I can give you a **ready-to-use EF Core migration** that updates all existing foreign keys to disable cascade delete in your database.
Do you want me to prepare that?