Absolutely. Here's a clear and concise set of notes on **Eager Loading** in Entity Framework Core — with examples and explanations.

**🧠 Eager Loading in EF Core — Notes & Examples**

**✅ What is Eager Loading?**

Eager loading is the process where related data is **loaded from the database as part of the initial query**. It's the opposite of lazy loading (where related data is only loaded when it's accessed later).

EF Core uses .Include() and .ThenInclude() to achieve this.

**🔁 Why Use Eager Loading?**

* Prevents the "N+1 query problem"
* Loads all the required data in a **single round-trip** to the database
* Better performance when related data **is always needed**

**🧱 Basic Syntax**

var students = await context.Students

.Include(s => s.Courses)

.ToListAsync();

🔹 This loads all Students and their associated Courses.

**🔗 Nested Relationships: .ThenInclude()**

When the navigation property itself has related data, you chain with .ThenInclude():

var activities = await context.Activities

.Include(a => a.Attendees) // Load Attendees

.ThenInclude(at => at.User) // Then load Users for each Attendee

.ToListAsync();

📌 Include() is for the first level relation.  
📌 ThenInclude() is for **next level navigation**.

**📁 Real-world Example**

var orders = await context.Orders

.Include(o => o.Customer) // Load Customer for each Order

.Include(o => o.OrderItems) // Load OrderItems

.ThenInclude(oi => oi.Product) // Load Product for each OrderItem

.ToListAsync();

This fetches:

* Orders ✅
* Their associated Customer ✅
* Their OrderItems ✅
* And the Products for each OrderItem ✅  
  All in **one DB query**.

**⚠️ When to Avoid Eager Loading**

* If related data is rarely used
* If the related data is **too large**
* If you're only loading a summary or list view

**📘 Tip: Multiple Includes on Same Entity**

var students = await context.Students

.Include(s => s.Courses)

.Include(s => s.Projects)

.ToListAsync();

Works fine — EF Core handles multiple includes efficiently.

Here are your notes on EF Core loading strategies:

**Entity Framework Core — Loading Strategies**

**1. 🟢 Eager Loading**

* **Definition**: Loads related entities *immediately* with the main entity.
* **Syntax**:
* var activities = await context.Activities
* .Include(a => a.Attendees)
* .ThenInclude(at => at.User)
* .ToListAsync();
* **Best For**: When you always need related data.
* **Avoid If**: The related data is heavy or rarely used.

**2. 🟠 Lazy Loading**

* **Definition**: Related data loads *only when accessed*.
* **Setup**:
  + Install: Microsoft.EntityFrameworkCore.Proxies
  + Enable:
  + options.UseLazyLoadingProxies()
  + Use virtual for navigation properties.
* **Best For**: On-demand related data.
* **Caution**: Can cause **N+1** problems and hidden performance hits.

**3. 🔵 Explicit Loading**

* **Definition**: You *manually* fetch related data after the entity is loaded.
* **Syntax**:
* var activity = await context.Activities.FirstOrDefaultAsync(a => a.Id == id);
* await context.Entry(activity).Collection(a => a.Attendees).LoadAsync();
* **Best For**: Selective or conditional loading.
* **Caution**: More code and possible multiple round-trips.

**⚖️ Quick Comparison**

| **Feature** | **Eager** | **Lazy** | **Explicit** |
| --- | --- | --- | --- |
| Trigger | Immediate | Property Access | Manual |
| Config Needed | ❌ | ✅ | ❌ |
| Control Level | Medium | Low | High |
| Performance Risk | Low | High | Medium |

Let me know if you want a PDF version or a diagram for these notes!