**Lazy Loading, Eager Loading, Explicit Loading**

* Was passiert bei folgendem Code?

var orders = context.Orders.ToList();

Console.WriteLine(orders[0].**Customer**.ContactName);

* Die Anwendung stürzt ab, weil das Entity Framework verbundene Entitäten aus Performancegründen **automatisch nicht nachlädt**.

**Lazy Loading (verzögertes Laden)**

**Ziel:** Verwandte Daten werden **erst beim Zugriff** auf die Navigationseigenschaft aus der Datenbank geladen.

**Vorteil:**

* Nur das geladen, was wirklich gebraucht wird.
* Spart initiale Ladezeit.

**Nachteil:**

* Es erzeugt unter Umständen **viele kleine SQL-Abfragen**
* Schwerer zu debuggen und zu kontrollieren.

**Lazy Loading aktivieren:**

* dotnet add package Microsoft.EntityFrameworkCore.Proxies
* services.AddDbContext<NorthwindContext>(options =>

options.UseLazyLoadingProxies().UseSqlite("Data Source=Northwind.sqlite"));

* Navigationseigenschaften im Modell virtual machen

**Test**

var orders = context.Orders.ToList();

Console.WriteLine(orders[0].**Customer**.ContactName);

Ergebnis: ContactName wird problemlos ausgegeben.

**Eager Loading (Laden nach Wunsch)**

**Ziel**: Verwandte Daten werden sofort mit der Hauptabfrage aus der Datenbank geladen.  
Technik: Verwendung von .Include() und .ThenInclude()

**Test**

var orders = context.Orders.OrderBy(o => o.OrderId).Include(o => o.Customer).ToList();

Console.WriteLine(orders[0].Customer.ContactName);

**Vorteil**:

* Nur eine Datenbankabfrage, die alles enthält.
* Gut für Daten, die immer gebraucht werden.

**Nachteil**:

* Es werden immer alle angegebenen Objekte geladen, auch wenn sie später nicht genutzt werden.
* Bei zu vielen Includes kann es unübersichtlich oder ineffizient werden.

**Explicit Loading (explizites Laden)**

**Ziel:** Verwandte Daten werden **manuell bei Bedarf** geladen, wenn der Kontext noch verfügbar ist.

**Test**

var orders = context.Orders.OrderBy(o => o.OrderId).ToList();

foreach (var order in orders)

{

if (order.OrderId % 2 == 0)

{

// "Manuelles" Nachladen von Customer mit gerader OrderId

context.Entry(order)

.Reference(o => o.Customer)

.Load();

Console.WriteLine(order.Customer.ContactName);

}

}

**Vorteil:**

* Volle Kontrolle, was wann geladen wird.
* Besser als Lazy Loading, wenn man gezielt laden will.

**Nachteil:**

* Zusätzlicher Code und mehr Abfragen nötig.
* Kontext muss noch verfügbar sein.

**Merke:**

├── **Eager**: Entität wird sofort mitgeladen

├── **Lazy**: Entität wird erst bei Zugriff geladen

└── **Explicit**: Entität wird auf konkreten, logischen Wunsch manuell geladen

|  |  |  |
| --- | --- | --- |
| Merkmal | Lazy Loading | Eager Loading |
| Ladezeitpunkt | Automatisch beim **Zugriff** | Automatisch beim **Abruf** der Hauptdaten |
| Steuerung | implizit über Navigation | explizit über .Include() / .ThenInclude() |
| Voraussetzungen | - Navigationseigenschaften müssen virtual sein - UseLazyLoadingProxies() muss aktiviert sein | keine besonderen |
| Anzahl Datenbankzugriffe | viele kleine (bei Bedarf) | meist nur 1 (JOIN) oder wenige |
| Kontrolle im Code | gering | mittel |
| Performance-Risiko | N+1-Problem bei Schleifen oder Listen | große Mengen an Daten auf einmal |
| Einsatzgebiet | einfache Detailseiten, kleine Datenmengen | komplexe Listenansichten, strukturierte Ausgabe |