**Change Tracking in Entity Framework Core**

EF Core verfolgt automatisch Änderungen, die an Entitäten vorgenommen werden. Dies geschieht durch den **DbContext**, der den Status jeder Entität überwacht.

**Change Tracking** ist ein zentrales Feature von **Entity Framework Core**. Es verfolgt Änderungen an Objekten (Entities), die vom **DbContext** verwaltet werden. Dadurch kann EF Core beim Aufruf von SaveChanges() automatisch entscheiden, welche SQL-Befehle (INSERT, UPDATE, DELETE) ausgeführt werden müssen.

**Vorteile von Change Tracking:**

**1. Automatische Erkennung von Änderungen**

* EF Core merkt sich den Ursprungszustand eines Objekts.
* Änderungen an Eigenschaften werden automatisch erkannt – kein manuelles SQL notwendig.

**Beispiel**: Wenn du produkt.Preis = 5.99 setzt, erkennt EF Core diese Änderung und erzeugt beim Speichern das passende UPDATE-Statement.

**2. Effizientes Speichern**

* Nur geänderte Objekte werden in die Datenbank geschrieben.
* Spart Zeit und Datenbankressourcen.

**3. Verbundene Entitäten werden mitverfolgt**

* Auch Änderungen an verknüpften Objekten (z. B. Navigationseigenschaften) werden erkannt, solange sie Teil des gleichen DbContext sind.

**4. Concurrency Handling möglich**

* EF Core kann erkennen, ob jemand anders den Datensatz verändert hat (z. B. über TimeStamp-Spalten), weil es den ursprünglichen Zustand kennt.

**5. Optimiert für einfache CRUD-Anwendungen**

* Du musst keine expliziten SQL-Befehle schreiben.
* Ideal für Anwendungen mit vielen Datenänderungen und Benutzerinteraktionen.

**6. Unterstützt Undo/Redo-Logik (mit Aufwand)**

* Da EF den Ursprungszustand kennt, kann man theoretisch auch zurücksetzen – nützlich z. B. in GUIs mit "Änderung verwerfen"-Funktion.

**Achtung:**

* Change Tracking kann bei vielen Objekten im Speicher speicherintensiv werden.
* Wenn du nur lesen willst, ist AsNoTracking() oft performanter.

**Tracking State ausgeben / abfragen**

Jede Entität hat einen Status, der vom DbContext überwacht wird. Die möglichen Zustände sind:

* **Added:** Die Entität wurde hinzugefügt und wird bei SaveChanges() eingefügt.
* **Modified:** Eine vorhandene Entität wurde geändert.
* **Deleted:** Die Entität wurde markiert, um aus der Datenbank gelöscht zu werden.
* **Unchanged:** Die Entität hat keine Änderungen erfahren.

**Überprüfen des Zustands einer Entität**

var product = new Product

{

ProductName = "Testprodukt",

UnitPrice = 9.99m,

CategoryId = category.CategoryId,

Discontinued = false

};

var entry = context.Entry(produkt);

Console.WriteLine($"Status von Product: {entry.State}");

**Change Tracking manuell deaktivieren**

Sie möchten z.B. eine Liste aller Produkte nur anzeigen (**READONLY**), ohne sie nachträglich zu bearbeiten? Dann brauchen Sie keine Änderungsverfolgung durch Entity Framework Core.

Das ergibt insbesondre viel Sinn, wenn wir große Mengen von Daten abfragen.

var products = context.Products.**AsNoTracking().**ToList();

* **Change Tracking** ermöglicht es, Änderungen an Entitäten zu **erkennen** und die Datenbank nur dann zu aktualisieren, wenn Änderungen vorgenommen wurden.
* EF Core **verfolgt den Zustand** der Entitäten **automatisch** und hilft dabei, nur die tatsächlich geänderten Entitäten zu speichern.
* Durch **AsNoTracking()** können wir die Leistung verbessern, wenn wir nur Daten abfragen und keine Änderungen verfolgen müssen.
* Entitäten, die mit **AsNoTracking()** aus DB abgerufen und danach im Code geändert wurden, werden **NICHT** in der DB **committed**.

**Alter & neuer Wert**

**Übung 1:**

1. Laden Sie einen Ihrer Wahl Kunden mit der CustomerID aus der Datenbank.
2. Ändern Sie lokal das Feld ContactName.
3. Nutzen Sie den Change Tracker, um den alten und den neuen Wert auszugeben.
4. Verwenden Sie KEIN SaveChanges().
5. Setzen Sie die Änderung zurück (Revert durch SetValues).
6. Geben Sie nach dem Revert den ContactName erneut aus.

**Übung 2**

1. Laden Sie ein Produkt mit ProductID = 1 aus der Datenbank.
2. Ändern Sie lokal den Produktnamen (ProductName) und den Preis (UnitPrice).
3. Nutzen Sie den Change Tracker, um die alten und neuen Werte anzuzeigen.
4. Verwenden Sie KEIN SaveChanges().
5. Setzen Sie die Änderungen vollständig zurück.
6. Geben Sie die finalen Werte nach dem Zurücksetzen aus.