CRUD Operationen mit EF Core

In diesem Artikel klären wir, was CRUD ist und wie wir es mit EF core einsetzen. Wir schauen außerdem an, welche Fallstricke es hier gibt.

Was ist CRUD?

CRUD steht für Create, Read, Update und Delete. Dies bezeichnet die Basis operationen, die wir an allen Daten ausüben können.

Datenhaltung	**Create**	**Read**	**Update**	**Delete**
List<T>	Add /AddRange	Index ([0]), Iterieren	Index [0] = „value“;	list.Remove(item);
SQL	INSERT	SELECT	UPDATE	DELETE
EF core	Add, AddRange	First, ToList, Single usw.	Update, UpdateRange	Remove, RemoveRange

CRUD kommt also immer wieder vor in der Informatik und der Softwareentwicklung.

CRUD mit EF Core
Oben in der Tabelle abgebildet sehen wir ja bereits die möglichen Methoden die wir jeweils aufrufen müssen, das ist aber natürlich nur die halbe Wahrheit. Grundsätzlich nutzen wir mit EF immer die folgende Herangehensweise:

1.) DbContext initialisieren / Session mit der Datenbank
2.) Read Operation:
    a. Auf dem dbContext das DbSet aufrufen, auf dem wir Daten lesen wollen
    b. Filter, Order usw. anwenden
    c. Per Executor ausführen
       Collection - (ToList, ToDictionary, ToArray)
       Objekt – First, FirstOrDefault, Single, SingleOrDefault
       Skalar – Count, Any, All, Average, Sum
3.) Create Operation:
    a. Entity erstellen/ Collection von Entites erstellen
    b. Add / AddRange auf dem dbContext aufrufen
    c. SaveChanges auf dem dbContext aufrufen
4.) Update Operation:
    a. Entity/Entities aus der Datenbank lesen
    b. Properties auf dem Entity/den Entities verändern
    c. Update/UpdateRange auf dem dbContext aufrufen
    d. SaveChanges auf dem dbContext aufrufen
5.) Delete Operation:
    a. Entity/Entities aus der Datenbank lesen
    b. Remove/RemoveRange auf dem dbContext aufrufen

c. SaveChanges auf dem dbContext aufrufen

Wir nehmen als Beispiel die Applikation aus dem Kurs und nutzen die Produkte tabelle als Beispiel:

```
public class Product : Entity
{
    public DateTime Start { get; set; }
    public DateTime? End { get; set; }

    public List<Part> Parts { get; set; }
    public Round Round { get; set; }
}
```

Für alle Beispiele erstellen wir den Context wie folgt:

var session = new LeanTrainingDbContext();

1. Read

```
var first = session.Products.First();
var products = session.Products.ToList();
// zum inkludieren der Navigations(erzeugt einen Join)
var products = session.Products.Include(x => x.Round).ToList();
// usw. usf
```

2. Create

Fürs Create gibt es zwei Möglichkeiten:

1.) Erstellen eines Products und aller seiner assoziierten Objekte
    a. Beim SaveChanges werden alle Entitäten erzeugt
       Auch Round und alle Parts in der Liste
2.) Erstellen eines Products und Füllen der NavigationProperties mit bereits vorhandenen Objekten
    a. Beim Aufruf des SaveChanges wird nur das Product erzeugt, die Parts werden lediglich mit einem foreign Key Eintrag versehen

1.)

```
var product1 = new Product
{
    Start = DateTime.Now,
    Parts = new List<Part>() { new Part() },
    Round = new Round()
};

session.Products.Add(product1);
session.SaveChanges();
```

```
2.)
var product2 = new Product
{
    Start = DateTime.Now,
    Parts = session.Take(2).ToList(),
    Round = session.Rounds.First()
};

session.Products.Add(product2);
session.SaveChanges();
```

3. Update

Fürs Updaten sind ebenfalls zwei Fälle für die Navigationproperties denkbar:

1. Mit erzeugen von neuen Daten
    a. Dann wird ein Insert erzeugt für diese Datei
2. Mit dem Überschreiben durch vorhandene Daten
    a. Dann wird einfach nur ein Update Statement generiert

```
1.
var first1 = session.Products.First();

first1.Round = new Round();

session.Products.Update(first1);

session.SaveChanges();

2.
var first2 = session.Products.First();

first2.Round = session.Rounds.OrderByDescending(x=>x.Id).First();

session.Products.Update(first2);

session.SaveChanges();
```

4. Delete

Im Falle von Delete werden die Abhängigen NavigationProperty Entitäten nicht mit gelöscht, lediglich in einer Linkingtabelle würde dieser Eintrag entfernt werden.

```
var first1 = session.Products.First();
session.Products.Remove(first1);
session.SaveChanges();
```

Was ist zu beachten?

Bei diesen grundsätzlichen Operationen gibt es natürlich etliche weiter Möglichkeiten und Ausprägungen aber das sind die einfachsten vorstellbaren.

Id Verwaltung

Grundsätzlich sollte Entity Framework Core die Schlüsselerzeugung für Primary Keys verwalten. Das hat den Vorteil, dass man nicht mit Ids im Code arbeiten muss. Anders ausgedrückt eine **Best Practice** ist, dass Ids auf den Entitäten nur lesbar für den Code zugreifbar sind.

Im Falle eines Updates/Inserts sollten Navigation properties anstelle der Foreign Key Ids verwendet werden. Dies ist natürlicher im Sinne der objektorientierten Programmierung.

AsNoTracking

Wenn wir Entitäten ohne Tracking abfragen und diese dann für Updates oder Adds in NavigationProperties verwenden, dann wird dies zwangsläufig zu einer Exception beim Aufruf der SaveChanges Methode führen.

Warum ist dies so? Entitäten die wir von der Datenbank abfragen haben bereits eine Id. Da EF für alle Entitäten immer eine Id haben muss. Auf einem Insert das beim SaveChanges generiert wird, wird dann versucht diese Id in einem Insert in der Datenbank unterzubringen. Primärschlüssel (Ids) dürfen aber nur einmal vorhanden sein.

Die Ursache dafür ist das ChangeTracking feature. Dieses versieht alle Entitäten mit einem Zustand. Bei AsNoTracking ist dieser Detached. Beim SaveChanges Aufruf wird für alle Entitäten mit dem Zustand Detached ein Insert SQL Statement generiert.

Kurzum, sollen Entitäten für ein Insert/Update verwendet werden, dürfen diese nicht mit AsNoTracking abgefragt werden.