Entity Framework CRUD

Inhalt

1 CRUD

CRUD = **C**reate, **R**ead, **U**pdate, **D**elete.
Man möchte mit Entity Framework nicht nur aus der Datenbank lesen, sondern auch schreiben, aktualisieren und löschen.

1.1 SaveChanges

Änderungen in der Datenbank führt man durch, indem man die C#-Objekte (die sogenannten Entities) verändert bzw. erzeugt, und diese dann mit **`SaveChanges()`** in die Datenbank speichert.
Wichtig: Es gibt keine separaten Methoden wie InsertRecords() o.ä., egal welche Änderung man durchführt heißt die Methode immer SaveChanges()!

1.2 Insert

Ein Insert sieht so aus:

```
var category = new Category { CategoryName = "abc", Description = "aa bb cc" };
db.Categories.Add(category);
db.SaveChanges();
```

Die Vorgangsweise dabei ist also folgende:

1. **neues C#-Objekt** erzeugen
2. dem zugehörigen **DbSet hinzufügen**
3. **`db.SaveChanges()`** aufrufen

Vor allem den zweiten Schritt (hinzufügen zum DbSet) darf man nicht vergessen. Ein Datensatz-Objekt ist ja ein ganz normales C#-Objekt (POCO), das nichts von einer Datenbank weiß. Erst das DbSet stellt das Bindeglied zw. Objekt und Datenbank her.

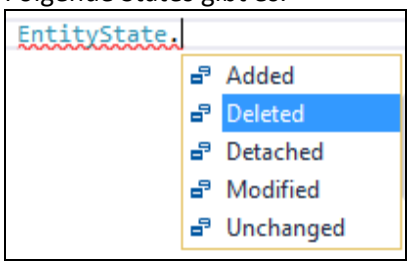
1.3 State

SaveChanges ist deswegen wichtig, weil für die einzelnen Entities vorerst nur ein Status gespeichert wird. Diesen Status ist über den DbContext zugänglich, und zwar mit **`db.Entry(entity).State`**.
Abhängig von diesem Status wird ein entsprechendes SQL-Statement (INSERT, DELETE,…) generiert.
Mit folgendem Beispiel sollte das Problem klar werden:

```
var category = new Category      { /*...*/};
AppendLog($"--> State before Add: {db.Entry(category).State}");
db.Categories.Add(category);
AppendLog($"--> State before Save: {db.Entry(category).State}");
db.SaveChanges();
AppendLog($"--> State after Save: {db.Entry(category).State}");
```

```
State before Add: Detached
State before Save: Added
State after Save: Unchanged
```

Folgende States gibt es:



1.4 Update

Aktualisieren kann man nur Datensätze, die den State **`Modified`** haben. Es muss daher immer folgendes erfolgen:

- Entity aus Datenbank lesen
- Entity beliebig verändern
- **`db.SaveChanges()`**

Der Code sieht daher so aus:

```
var category = db.Categories.Single(x => x.CategoryName == "abc");
category.Description = "New description now";
db.SaveChanges();
```

Auch hier kann man wieder den State beobachten:

```
var category = db.Categories.Single(x => x.CategoryName == "abc");
AppendLog($"--> State before Update: {db.Entry(category).State}");
category.Description = "New description now";
AppendLog($"--> State before Save: {db.Entry(category).State}");
db.SaveChanges();
AppendLog($"--> State after Save: {db.Entry(category).State}");
```

```
--> State before Update: Unchanged
--> State before Save: Modified
--> State after Save: Unchanged
```

1.5 Delete

Löschen von Datensätzen erfolgt ebenfalls über den Datenbankkontext, und zwar über die Methode **`Remove()`**. Auch hier ist der Aufruf von SaveChanges() wichtig. Denn es wird für das Entity nur ein Flag gesetzt, das es als „zu löschen“ markiert.

```
int id = int.Parse(txtId.Text);
var category = db.Categories.Single(x => x.CategoryID == id);
AppendLog($"--> State before Delete: {db.Entry(category).State}");
db.Categories.Remove(category);
AppendLog($"--> State before Save: {db.Entry(category).State}");
db.SaveChanges();
AppendLog($"--> State after Save: {db.Entry(category).State}");
```

```
--> State before Delete: Unchanged
--> State before Save: Deleted
--> State after Save: Detached
```

1.5.1 Mehrere Datensätze löschen

Immer jeden Record einzeln zu löschen ist nicht sehr performant. D.h. will man z.B. eine Tabelle mit 1000 Einträgen komplett löschen, dauert obige Variante viel zu lange (es wird ja jeweils ein SQL-Delete generiert und einzeln zur Datenbank geschickt).
In diesem Fall macht es **ausnahmsweise** Sinn, auf die SQL-Variante auszuweichen:

```
string city = "Reims";
db.Database.ExecuteSqlRaw($"DELETE FROM Orders WHERE ShipCity ={city}");
```

Der Nachteil ist, dass man wieder auf der String-Ebene ist, und die Anweisung nicht mehr kompiliert wird u. somit fehleranfällig ist. Außerdem muss man wissen, die die Tabelle heißt (wie wir später sehen werden, kann die generierte Klasse anders heißen wie die Tabelle.

1.5.2 Truncate Table

Will man alle Datensätze einer Tabelle löschen, geht das somit auf zwei Arten:

Langsam:
```
db.OrderDetails.RemoveRange(db.OrderDetails);
db.SaveChanges();
```

Schnell:
```
db.Database.ExecuteSqlRaw("DELETE FROM [Order Details]");
```

Achtung: Die Tabelle heißt „Order Details“, daher muss man sie als **`[Order Details]`** angeben.