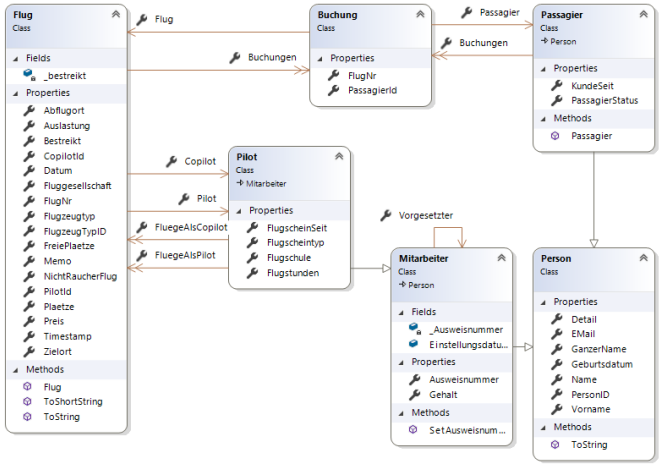
Spickzettel („Cheat Sheet“) LINQ mit Entity Framework/Entity Framework Core

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

www.IT-Visions.de
Dr. Holger Schwichtenberg

Objektmodell für diese Beispiele



Alle folgenden Befehle gehen von der vorherigen Instanziierung des Kontextes aus:

```
WWWingsContext ctx = new WWWingsContext();
```

Es wird jeweils neben dem LINQ-Befehl auch die alternative Lambda-Schreibweise dargestellt. Der resultierende SQL-Befehl ist immer gleich für die LINQ- und die Lambda-Schreibweise.

Einfache SELECT-Befehle (Alle Datensätze)

```
Flug[] flugSet0a = (from f in ctx.FlugSet select f).ToArray();
Flug[] flugSet0b = ctx.FlugSet.ToArray();

List<Flug> flugSet1a = (from f in ctx.FlugSet select f).ToList();
List<Flug> flugSet1b = ctx.FlugSet.ToList();

Dictionary<int, Flug> flugSet2a = (from f in ctx.FlugSet select f)
        .ToDictionary(f=>f.FlugNr, f=>f);
Dictionary<int, Flug> flugSet2b = ctx.FlugSet
        .ToDictionary(f => f.FlugNr, f => f);
```

Bedingungen (where)

```
List<Flug> flugSet3a = (from f in ctx.FlugSet
                where f.Abflugort == "Essen/Mülheim" &&
                (f.Zielort == "Rom" || f.Zielort == "Paris")
                && f.FreiePlaetze > 0
                select f)
                .ToList();
```

```
List<Flug> flugSet3b = ctx.FlugSet
        .Where(f => f.Abflugort == "Essen/Mülheim" &&
           (f.Zielort == "Rom" || f.Zielort == "Paris")
           && f.FreiePlaetze > 0)
        .ToList();
```

Bedingungen mit Mengen (in)

```
List<string> Orte = new List<string>() { "Berlin", "Hamburg", "Köln",
"Essen/Mülheim" };

List<Flug> flugSet4a = (from f in ctx.FlugSet
                where Orte.Contains(f.Abflugort)
                select f)
                .ToList();

List<Flug> flugSet4b = ctx.FlugSet
                .Where(f =>Orte.Contains(f.Abflugort))
                .ToList();
```

Sortierungen (orderby)

```
List<Flug> flugSet5a = (from f in ctx.FlugSet
                where f.Abflugort == "Essen/Mülheim"
                orderby f.Datum, f.Zielort, f.FreiePlaetze descending
                select f).ToList();

List<Flug> flugSet5b = ctx.FlugSet
                .Where(f => f.Abflugort == "Essen/Mülheim")
                .OrderBy(f => f.Datum)
                .ThenBy(f => f.Zielort)
                .ThenByDescending(f => f.FreiePlaetze)
                .ToList();
```

Paging (Skip() und Take())

```
List<Flug> flugSet6a = (from f in ctx.FlugSet
                where f.Abflugort == "Essen/Mülheim"
                orderby f.Datum
                select f).Skip(100).Take(10).ToList();

List<Flug> flugSet6b = ctx.FlugSet
                .Where(f => f.Abflugort == "Essen/Mülheim")
                .OrderBy(f => f.Datum)
                .Skip(100).Take(10).ToList();
```

Projektion

```
List<Flug> flugSet7a = (from f in ctx.FlugSet
                where f.Abflugort == "Essen/Mülheim"
                orderby f.Datum
                select new Flug()
                {
                 FlugNr = f.FlugNr,
                 Datum = f.Datum,
                 Abflugort = f.Abflugort,
                 Zielort = f.Zielort,
                 FreiePlaetze = f.FreiePlaetze,
                 Timestamp = f.Timestamp
                }).ToList();

List<Flug> flugSet7b = ctx.FlugSet
                .Where(f => f.Abflugort == "Essen/Mülheim")
                .OrderBy(f => f.Datum)
                .Select(f => new Flug()
                {
                 FlugNr = f.FlugNr,
                 Datum = f.Datum,
                 Abflugort = f.Abflugort,
                 Zielort = f.Zielort,
                 FreiePlaetze = f.FreiePlaetze,
                 Timestamp = f.Timestamp
                }).ToList();
```

Aggregate (Count(), Min(), Max(), Average(), Sum())

```
int agg1a = (from f in ctx.FlugSet select f).Count();
int? agg2a = (from f in ctx.FlugSet select f).Sum(f => f.FreiePlaetze);
int? agg3a = (from f in ctx.FlugSet select f).Min(f => f.FreiePlaetze);
int? agg4a = (from f in ctx.FlugSet select f).Max(f => f.FreiePlaetze);
double? agg5a = (from f in ctx.FlugSet select f).Average(f =>
f.FreiePlaetze);

int agg1b = ctx.FlugSet.Count();
int? agg2b = ctx.FlugSet.Sum(f => f.FreiePlaetze);
int? agg3b = ctx.FlugSet.Min(f => f.FreiePlaetze);
int? agg4b = ctx.FlugSet.Max(f => f.FreiePlaetze);
double? agg5b = ctx.FlugSet.Average(f => f.FreiePlaetze);
```

Einzelobjekte (SingleOrDefault(), FirstOrDefault())

```
Flug flug1a = (from f in ctx.FlugSet select f)
                .SingleOrDefault(f => f.FlugNr == 101);

Flug flug1b = ctx.FlugSet
                .SingleOrDefault(f => f.FlugNr == 101);

Flug flug2a = (from f in ctx.FlugSet
                where f.FreiePlaetze > 0
                orderby f.Datum
                select f).FirstOrDefault();

Flug flug2b = ctx.FlugSet
                .Where(f => f.FreiePlaetze > 0)
                .OrderBy(f => f.Datum)
                .FirstOrDefault();
```

Gruppierungen (Group By)

```
var group1a = (from f in ctx.FlugSet
               group f by f.Abflugort into g
               select new { Ort = g.Key, Anzahl = g.Count(), Sum =
g.Sum(f => f.FreiePlaetze), Avg = g.Average(f => f.FreiePlaetze) })
              .ToList();

var group1b = ctx.FlugSet
              .GroupBy(f => f.Abflugort)
              .Select(g => new
              {
               Ort = g.Key,
               Anzahl = g.Count(),
               Sum = g.Sum(f => f.FreiePlaetze),
               Avg = g.Average(f => f.FreiePlaetze)
              }).ToList();
```

Hinweis: LINQ-Gruppierungen werden in Entity Framework Core auch in Version 2.0 noch im RAM ausgeführt. Erst seit Version 2.1 werden diese z.T. korrekt in SQL übersetzt werden. Gruppierungen sollte man daher in den EFCore-Versionen 1.0 bis 2.0 direkt in SQL formulieren. Auch in Version 2.1 sollte man prüfen, ob eine Gruppierung wirklich in der Datenbank ausgeführt wird.

Verbundene Objekte (Include())

```
List<Flug> flugDetailsSet1a = (from f in ctx.FlugSet
              .Include(f => f.Pilot)
              .Include(f => f.Buchungen).ThenInclude(b => b.Passagier)
              where f.Abflugort == "Essen/Mülheim"
              orderby f.Datum
              select f)
              .ToList();

List<Flug> flugDetailsSet1b = ctx.FlugSet
              .Include(f => f.Pilot)
              .Include(f => f.Buchungen).ThenInclude(b => b.Passagier)
              .Where(f => f.Abflugort == "Essen/Mülheim")
              .OrderBy(f => f.Datum)
              .ToList();
```

Hinweis: Entity Framework Core führt hier direkt nacheinander zwei SQL-Befehle aus, um Joins zu vermeiden.

Inner Join (Join)

Explizite Join-Operationen sind nicht notwendig, wenn es Navigationsbeziehungen gibt (vgl. "Verbundene Objekte"). Im nachfolgenden Beispiel werden, um einen Fall ohne Navigationsbeziehung zu konstruieren, alle Flüge gesucht, die die gleiche ID wie ein Pilot haben.

```
var flugDetailsSet2a = (from f in ctx.FlugSet
                        join p in ctx.PilotSet
                        on f.FlugNr equals p.PersonID
                        select new { Nr = f.FlugNr, Flug = f, Pilot = p })
                        .ToList();

var flugDetailsSet2b = ctx.FlugSet
                       .Join(ctx.PilotSet, f => f.FlugNr, p => p.PersonID,
                       (f, p) => new { Nr = f.FlugNr, Flug = f, Pilot = p })
                       .ToList();
```

Cross Join (Kartesisches Produkt)

```
var flugDetailsSet3a = (from f in ctx.FlugSet
                        from b in ctx.BuchungSet
                        from p in ctx.PassagierSet
                        where f.FlugNr == b.FlugNr &&
                          b.PassagierId == p.PersonID && f.Abflugort == "Rom"
                        select new { Flug = f, Passagiere = p })
                         .ToList();

var flugDetailsSet3b = ctx.FlugSet
      .SelectMany(f => ctx.BuchungSet, (f, b) => new  { f = f,  b = b})
      .SelectMany(z => ctx.PassagierSet, (x, p) => new {x = x, p = p})
      .Where(y => ((y.x.f.FlugNr == y.x.b.FlugNr) &&
        (y.x.b.PassagierId == y.p.PersonID)) && y.x.f.Abflugort == "Rom")
      .Select(z => new {Flug = z.x.f, Passagiere = z.p } )
      .ToList();
```

Join mit Gruppierung

```
var flugDetailsSet4a = (from b in ctx.BuchungSet
    join f in ctx.FlugSet on b.FlugNr equals f.FlugNr
    join p in ctx.PassagierSet on b.PassagierId equals p.PersonID
    where f.Abflugort == "Berlin"
    group b by b.Flug into g
    select new { Flug = g.Key, Passagiere = g.Select(x => x.Passagier) })
    .ToList();

var flugDetailsSet4b = ctx.BuchungSet
      .Join(ctx.FlugSet, b => b.FlugNr, f => f.FlugNr, (b, f) =>
       new { b = b, f = f })
      .Join(ctx.PassagierSet, x => x.b.PassagierId,
        p => p.PersonID, (x, p) => new { x = x, p = p })
      .Where(z => (z.x.f.Abflugort == "Berlin"))
      .GroupBy(y => y.x.b.Flug, y => y.x.b)
      .Select(g => new { Flug = g.Key, Passagiere = g.Select(x =>
        x.Passagier) })
      .ToList();
```

Unter-Abfragen

Achtung: Unterabfragen werden einzeln für jeden Ergebnisdatensatz ausgeführt!

```
List<Flug> flugDetailsSet5a = (from f in ctx.FlugSet
      where f.FlugNr == 101
      select new Flug()       {
       FlugNr = f.FlugNr,
       Datum = f.Datum,
       Abflugort = f.Abflugort,
       Zielort = f.Zielort,
       Pilot = (from p in ctx.PilotSet where
                p.PersonID == f.PilotId select p)
               .FirstOrDefault(),
       Copilot = (from p in ctx.PilotSet where
                  p.PersonID == f.CopilotId select p)
                 .FirstOrDefault(),
      }).ToList();

List<Flug> flugDetailsSet5b = ctx.FlugSet
  .Where(f => f.FlugNr == 101)
  .Select(f =>new Flug()       {
       FlugNr = f.FlugNr,
       Datum = f.Datum,
       Abflugort = f.Abflugort,
       Zielort = f.Zielort,
       Pilot = ctx.PilotSet
          .Where(p => (p.PersonID == f.PilotId))
          .FirstOrDefault(),
       Copilot = ctx.PilotSet
          .Where(p => (p.PersonID) == f.CopilotId)
          .FirstOrDefault()
      }).ToList();
```

Links

Eine ausführlichere Beispielsammlung von insgesamt 101 LINQ-Befehlen finden Sie hier:
https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

Über den Autor

Dr. Holger Schwichtenberg *gehört zu den bekanntesten Experten für die Programmierung mit Webtechniken und .NET in Deutschland. Er hat zahlreiche Bücher zu .NET und Webtechniken veröffentlicht und spricht regelmäßig auf Fachkonferenzen. Sie können ihn und seine Kollegen für Schulungen, Beratungen und Projektunterstützung buchen.*
E-Mail: ***buero@IT-Visions.de***
Website: ***www.IT-Visions.de***
Weblog: ***www.dotnet-doktor.de***