LINQ to SQL

Proseminar *Objektorientiertes Programmieren mit .NET und C#*

Christoph Knüttel

Institut für Informatik
Software & Systems Engineering

Agenda

1. **LINQ allgemein**
   - Vorteile
   - Bausteine und Varianten
   - Syntax
   - Ausführung von Abfragen

2. **LINQ to SQL**
   - Motivation
   - Grundlagen
   - Ausführung von Abfragen
   - Beispiele für wichtige Funktionen

LINQ – Beispiel

```
// Ein Kasten als Bier-Array mit 4 vollen Augustinern
Bier[] kasten = new Bier[20];
kasten[2]  = new Bier { Leer = false, Marke = "Augustiner" };
kasten[6]  = new Bier { Leer = false, Marke = "Augustiner" };
kasten[10] = new Bier { Leer = false, Marke = "Augustiner" };
kasten[15] = new Bier { Leer = false, Marke = "Augustiner" };
```

```
// Bring zwei Augustiner mit!
// Mit Schleife
List<Bier> zweiVolleAugustiner = new List<Bier>();
int genommeneFlaschen = 0;
foreach (Bier flasche in kasten)
{
    if (genommeneFlaschen < 2)
    {
        if (flasche != null
                    && !flasche.Leer
                    && flasche.Marke == "Augustiner")
        {
            zweiVolleAugustiner.Add(flasche);
            genommeneFlaschen++;
        }
    }
    else
    {
        break;
    }
}
```

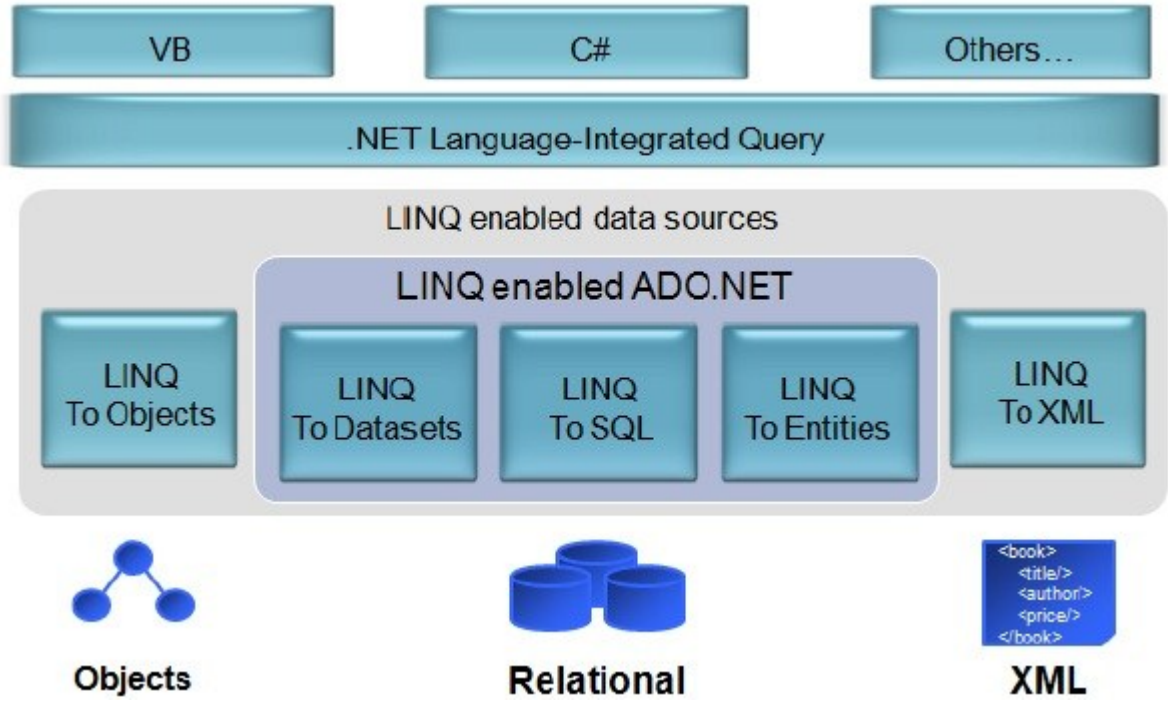
```
// Bring zwei Augustiner mit!
// Mit LINQ
List<Bier> nochMalzweiVolleAugustiner =
    kasten
    .Where(flasche => flasche != null
                    && !flasche.Leer
                    && flasche.Marke == "Augustiner")
    .Take(2)
    .ToList();
```

LINQ – Vorteile

- LINQ ist kürzer und leichter lesbar
- LINQ ist intuitiver, da man komplexe Abfragen in anschauliche Einzelschritte zerlegen kann
- LINQ ist deklarativ, d.h. man definiert, was man haben will, wie die Abfrage konkret ausgeführt wird, ist im Framework implementiert

```
// Bring zwei Augustiner mit!
// Mit LINQ
List<Bier> nochMalzweiVolleAugustiner =
    kasten
    .Where(flasche => flasche != null
                   && !flasche.Leer
                   && flasche.Marke == "Augustiner")
    .Take(2)
    .ToList();
```

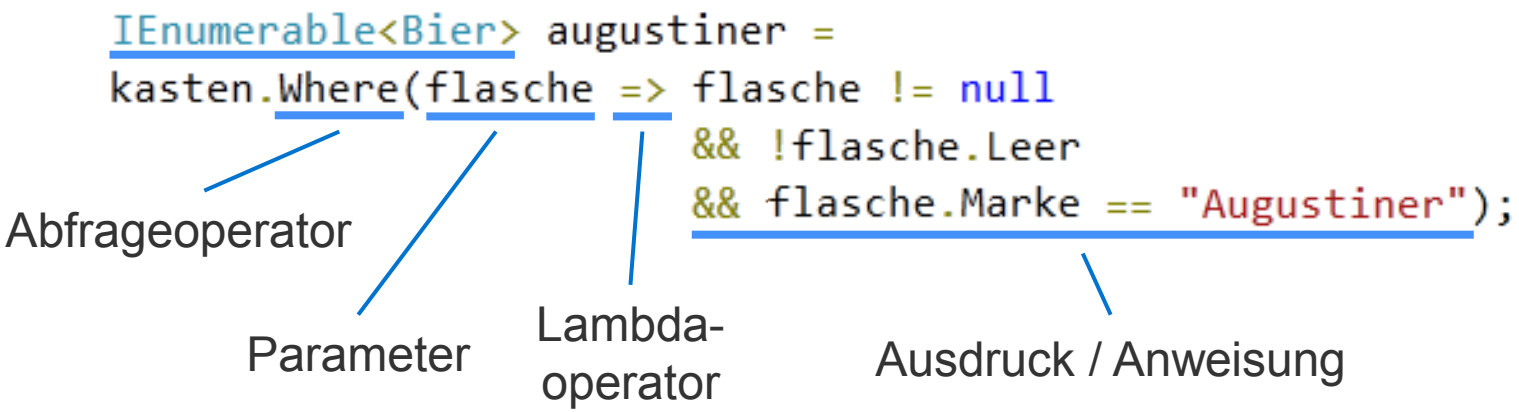
LINQ – Bausteine und Varianten



- LINQ ist über das Framework **in die Programmiersprache integriert**
- LINQ ermöglicht **Abfragen von verschiedenen Datenquellen** mit Hilfe **derselben Syntax**

LINQ – Syntax-Variante 1

Abfrageoperatoren



- Lambda-Ausdrücke
- Typ des Parameters wird implizit vom Typ der Elemente in der Collection abgeleitet
- Rechte Seite kann ein Ausdruck oder eine Anweisung sein
- Verkettete Methodenaufrufe werden sequenziell ausgeführt

LINQ – Syntax-Variante 2

Abfrageausdrücke

```
IEnumerable<Bier> augustiner =
    from  flasche  in kasten
    where flasche  != null
                   && !flasche.Leer
                   && flasche.Marke == "Augustiner"
    select flasche;
```

- Ähnlich der SQL-Syntax
- Typinferenz für Parameter
- Werden vom Compiler in die semantisch äquivalenten Methodenaufrufe übersetzt, die die Abfrageoperatoren anbieten

LINQ – Ausführung von Abfragen

```
// Wie viele volle Augustiner sind im Kasten?
IEnumerable<Bier> vorhandeneAugustiner =
    kasten.Where(flasche => flasche != null
                         && !flasche.Leer
                         && flasche.Marke == "Augustiner");

Console.WriteLine("\nEs sind noch {0} Augustiner da!",
                                  vorhandeneAugustiner.Count());
```

```
Es sind noch 4 Augustiner da!
```

```
// Nimm ein Augustiner aus dem Kasten (Prost!)
kasten[10] = null;

// Wie viele volle Augustiner sind jetzt noch im Kasten?
Console.WriteLine("\nEs sind noch {0} Augustiner da!",
                                  vorhandeneAugustiner.Count());
```

```
Es sind noch 3 Augustiner da!
```

LINQ to SQL – Motivation

- **Unterstützung bei der Formulierung von Datenbankabfragen** durch Compiler und IntelliSense zur Design-Zeit
- **Beheben von Nachteilen des traditionellen ADO.NET-Datenzugriffs**
- **Typsicheres Erzeugen von Objekten (= „Entities“)** aus den Datensätzen der Datenbank

LINQ to SQL – Objekt-Relationales Mapping

Datenbank – Datenmodell

Gaeste

Column Name	Data Type	Nullable
Id	int	No
Name	nvarchar(50)	No
Stammkneipe	int	Yes
Besuche	int	Yes
Schulden	money	Yes

Anwendung – Objektmodell

```
using System.Data.Linq.Mapping;

[Table(Name = "Gaeste")]
public class Gast
{

    [Column(Name = "Id", IsPrimaryKey = true,
                         IsDbGenerated = true)]
    public int Id { get; set; }

    [Column(Name = "Name", CanBeNull = false)]
    public string Name { get; set; }

    [Column(Name = "Stammkneipe", CanBeNull=true)]
    public System.Nullable<int> StammkneipeId
    { get; set; }

    [Column(Name = "Besuche")]
    public int Besuche { get; set; }

    [Column(Name="Schulden")]
    public System.Nullable<Decimal> Schulden { get;
```

LINQ to SQL – Verbindung zur Datenbank

```
using System.Linq;
using System.Data.Linq;

// Eine DataContext-Instanz erzeugen, die die Verbindung mit der Datenbank herstellt
DataContext dataContext =
    new DataContext(@"Data Source=(local);Initial Catalog=Nachtschicht;Integrated Security=True");
```

Servername

Datenbank

Authentifzierungs-modus

Connection String

LINQ to SQL – Read

```
using System.Linq;
using System.Data.Linq;

// Eine DataContext-Instanz erzeugen, die die Verbindung mit der Datenbank herstellt
DataContext dataContext =
    new DataContext(@"Data Source=(local);Initial Catalog=Nachtschicht;Integrated Security=True");

// Eine getypte Table-Instanz erzeugen
Table<Kneipe> Kneipen = dataContext.GetTable<Kneipe>();

// Abfrage definieren:
// Alle Kneipen, die mit "B" anfangen
IQueryable<Kneipe> kneipenMitB =
    from kneipe in Kneipen
    where kneipe.Name.StartsWith("B")
    select kneipe;

// Die gefundenen Kneipennamen ausgeben
foreach (Kneipe kneipe in kneipenMitB)
{
    Console.WriteLine(kneipe.Name);
}
```

```
Bertis Boazn
```

LINQ to SQL – Ausführung von Abfragen

Anwendung (.NET Framework)

```
IQueryable<Kneipe> kneipenMitB =
    from kneipe in Kneipen
    where kneipe.Name.StartsWith("B")
    select kneipe;

foreach (Kneipe kneipe in kneipenMitB)
{
    Console.WriteLine(kneipe.Name);
}
```

Datenbank (SQL Server)

```
SELECT
    [Id],
    [Name],
    [Brauerei] AS [BrauereiId]
FROM [Kneipen]
WHERE [Name] LIKE @p0

-- @p0: [B%]
```

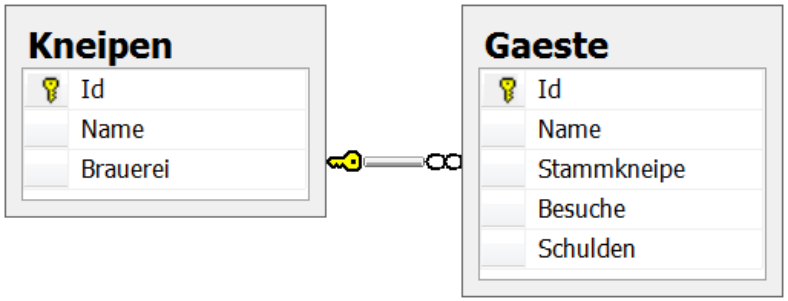
- **Übersetzung der gesamten Abfragedefinition** in **ein einziges SQL-Statement**
- **Verzögerte Ausführung** beim Aufrufen eines Enumerators: SQL-Statement wird an die Datenbank geschickt und ausgeführt

LINQ to SQL – Read mit Join

```
// Getypte Table-Instanzen
Table<Gast>   Gaeste  = dataContext.GetTable<Gast>();
Table<Kneipe> Kneipen = dataContext.GetTable<Kneipe>();

// Gäste und ihre Stammkneipen
var gaesteUndIhreStammkneipen =
    from gast   in Gaeste
    join kneipe in Kneipen
        on gast.StammkneipeId equals kneipe.Id
    orderby kneipe.Name
    select new { gastName   = gast.Name,
                 kneipeName = kneipe.Name };

// Gib den Namen jedes Gastes und seine Stammkneipe aus
foreach (var gastUndKneipe in gaesteUndIhreStammkneipen)
{
    Console.WriteLine("{0} sitzt in {1}.",
                      gastUndKneipe.gastName,
                      gastUndKneipe.kneipeName);
}
```




```
SELECT
    [Gaeste].[Name]  AS [gastName],
    [Kneipen].[Name] AS [kneipeName]
FROM [Gaeste] INNER JOIN [Kneipen]
    ON [Gaeste].[Stammkneipe] = [Kneipen].[Id]
ORDER BY [Kneipen].[Name]
```

```
Ernst sitzt in Bertis Boazn.
Bernd sitzt in Bertis Boazn.
Florian sitzt in Bertis Boazn.
Gunther sitzt in Kevins Kaschemme.
Alfons sitzt in Paulis Pilsquelle.
Helmut sitzt in Steffis Stüberl.
```

LINQ to SQL – Create

```
// Die Stammkneipe fuer den neuen Gast holen
Kneipe kneipeMitNeuemGast =
    Kneipen.Single(kneipe => kneipe.Name ==
                             "Paulis Pilsquelle");

// Eine neue Gast-Entity erzeugen
Gast neuerGast =
    new Gast { Name = "Manfred",
               StammkneipeId = kneipeMitNeuemGast.Id,
               Besuche = 1,
               Schulden = 1.00M };

// Die neue Gast-Entity an die Table-Instanz übergeben
Gaeste.InsertOnSubmit(neuerGast);

// Der DataContext-Instanz mitteilen, dass sie
// alle Änderungen in die Datenbank persistieren soll
dataContext.SubmitChanges();
```

LINQ to SQL – Update

```
// Alle Stammgaeste von Paulis Pilsquelle
IQueryable<Gast> gaesteVonPaulisPilsquelle =
    from gast in Gaeste
    join kneipe in Kneipen
        on gast.StammkneipeId equals kneipe.Id
    where kneipe.Name == "Paulis Pilsquelle"
    select gast;

// Schuldenstand auf 0,00 zurücksetzen
foreach (Gast gast in gaesteVonPaulisPilsquelle)
{
    gast.Schulden = 0.0M;
}

dataContext.SubmitChanges();
```

LINQ to SQL – Delete

```
// Den Gast definieren, der gelöscht werden soll
Gast zuLöschenderGast =
    Gaeste.Single(gast => gast.Name == "Ernst");

// Die zu löschende Gast-Entity
// an die Table-Instanz übergeben
Gaeste.DeleteOnSubmit(zuLöschenderGast);

dataContext.SubmitChanges();
```

LINQ – Vorteile

- Abfragen direkt im Programmcode
- Gleiche Syntax für verschiedene Datenquellen
- Compiler- und IntelliSense-Unterstützung
- Kurzer und intuitiver Code
- Deklaratives Formulieren (nicht „Wie“, sondern „Was“)

Literatur & Referenzen

- Aytekin, Özgür: *LINQ. Theorie und Praxis für Einsteiger*. München u.a.: Addison-Wesley, 2008.

- Kansy, Thorsten: *LINQ. Direkte Abfragen mit language integrated query*. München: Entwickler.press, 2009.

- Kulkarni, Dinesh; Bolognese, Luca; Warren, Matt; Hejlsberg, Anders; George, Kit: *LINQ to SQL: .NET Language-Integrated Query for Relational Data*, http://msdn.microsoft.com/en-us/library/bb425822.aspx, March 2007.

- Marguerie, Fabrice; Eichert, Steve; Wooley, Jim: *LINQ in Action*. Greenwich, Conn.: Manning, 2008.
  deutsche Übersetzung: *LINQ im Einsatz*. München: Hanser, 2008.

Vielen Dank für Eure Aufmerksamkeit!

Habt ihr Fragen?

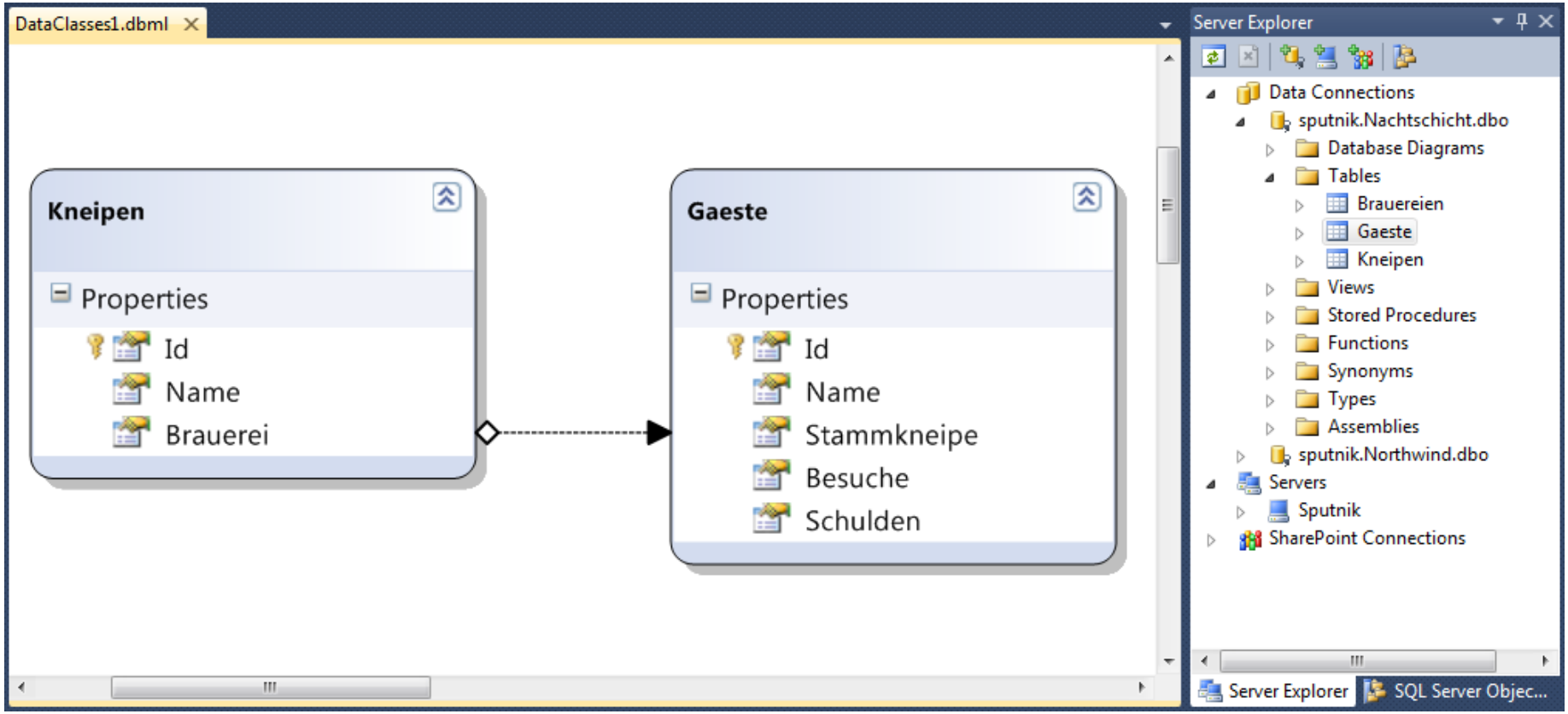
LINQ – Ausführung von Abfragen

```
// Abfrage sofort ausführen und Ergebnis dauerhaft speichern
List<Bier> nochVorhandeneAugustiner =
    kasten.Where(flasche => flasche != null
                         && !flasche.leer
                         && flasche.marke == "Augustiner")
          .ToList();
```

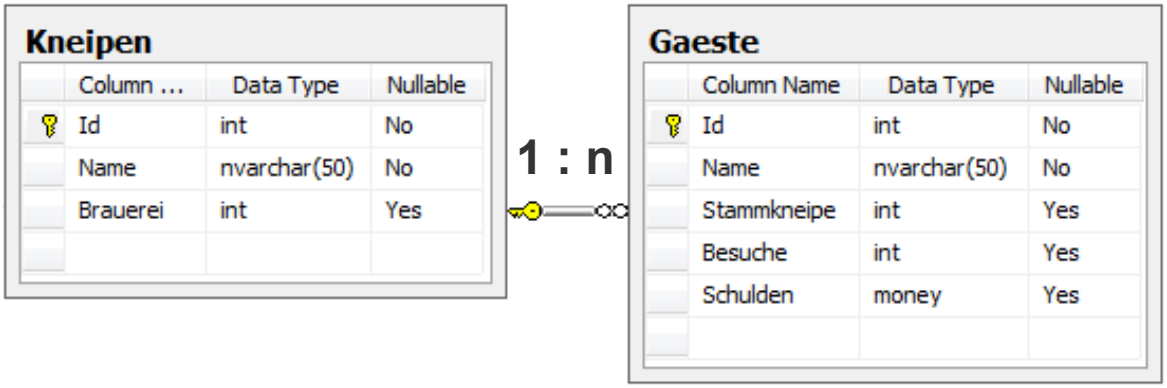
- **Sofortige Ausführung**
- muss durch Aufruf von `ToArray(), ToList()`... explizit erzwungen werden
- In der Ergebnis-Collection werden Clone der gefundenen Elemente gespeichert
- Änderungen in der ursprünglichen Sequenz werden nicht an die Ergebnis-Collection weitergereicht

LINQ to SQL-Designer

Backup



Objekt-Relationales Mapping - Associations




```
// n Stammgäste der Kneipe

private EntitySet<Gast> _Stammgaeste;

[Association(Name="FK_Gaeste_Kneipen",
             Storage="_Stammgaeste",
             OtherKey="StammkneipeId")]
public EntitySet<Gast> Stammgaeste
{
    get { return this._Stammgaeste; }
    set { this._Stammgaeste.Assign(value); }
}
```

```
// 1 Stammkneipe des Gastes

[Column(Name="Stammkneipe", CanBeNull=true)]
public System.Nullable<int> StammkneipeId { get; set; }

private EntityRef<Kneipe> _Stammkneipe;

[Association(Name = "FK_Gaeste_Kneipen",
             Storage = "_Stammkneipe",
             ThisKey = "Id")]
public Kneipe Stammkneipe
{
    get { return this._Stammkneipe.Entity; }
    set { this._Stammkneipe.Entity = value; }
}
```

Typsichere DataContext- und Table-Klassen

```
public partial class Nachtschicht : DataContext
{
    public Table<Kneipe> Kneipen;
    public Table<Gast>   Gaeste;

    public Nachtschicht(string connection) : base(connection) { }
}
```

```
Nachtschicht dataContext =
    new Nachtschicht(@"Data Source=(local);Initial Catalog=Nachtschicht;Integrated Security=True");

Gast zuLöschenderGast =
    dataContext.Gaeste.Single(gast => gast.Name == "Ernst");
```

LINQ to SQL – Create

```
// Die Stammkneipe fuer den neuen Gast holen
Kneipe kneipeMitNeuemGast =
    Kneipen.Single(kneipe => kneipe.Name ==
                             "Paulis Pilsquelle");

// Eine neue Gast-Entity erzeugen
Gast neuerGast =
    new Gast { Name = "Peter",
               StammkneipeId = kneipeMitNeuemGast.Id,
               Besuche = 1,
               Schulden = 1.00M };

// Die neue Gast-Entity an die Table-Instanz übergeben
Gaeste.InsertOnSubmit(neuerGast);

// Der DataContext-Instanz mitteilen, dass sie
// alle Änderungen in die Datenbank persistieren soll
dataContext.SubmitChanges();
```

```
INSERT INTO [Gaeste]
    (
    [Name],
    [Stammkneipe],
    [Besuche],
    [Schulden]
    )
VALUES
    (
    @p0,
    @p1,
    @p2,
    @p3
    )

-- @p0: [Peter]
-- @p1: [2]
-- @p2: [1]
-- @p3: [1.00]
```

LINQ to SQL – Read

```
// Getypte Table-Instanzen
Table<Gast>   Gaeste  = dataContext.GetTable<Gast>();

// Finde die fleissigsten Kneipengaenger
IQueryable<Gast> fleissigsteKneipengaenger =
    Gaeste.Where(gast => gast.Besuche ==
                         ( from gast1 in Gaeste
                           select gast1.Besuche)
                           .Max() );

// Gib die Namen aus
foreach (Gast gast in fleissigsteKneipengaenger)
{
    Console.WriteLine(gast.Name);
}
```

```
SELECT
    [Id],
    [Name],
    [Stammkneipe] AS [StammkneipeId],
    [Besuche],
    [Schulden]
FROM [Gaeste]
WHERE [Besuche] =
    (
    SELECT MAX([Besuche])
    FROM [Gaeste]
    )
```

```
Helmut
Manfred
```

LINQ to SQL – Update

```
// Alle Stammgaeste von Paulis Pilsquelle
IQueryable<Gast> gaesteVonPaulisPilsquelle =
    from gast in Gaeste
    join kneipe in Kneipen
        on gast.StammkneipeId equals kneipe.Id
    where kneipe.Name == "Paulis Pilsquelle"
    select gast;

// Schuldenstand auf 0,00 zurücksetzen
foreach (Gast gast in gaesteVonPaulisPilsquelle)
{
    gast.Schulden = 0.0M;
}

dataContext.SubmitChanges();
```

```
UPDATE [Gaeste]
SET    [Schulden]     = @p5
WHERE  [Id]           = @p0
  AND  [Name]         = @p1
  AND  [Stammkneipe]  = @p2
  AND  [Besuche]      = @p3
  AND  [Schulden]     = @p4

-- @p0: [2]
-- @p1: [Alfons]
-- @p2: [2]
-- @p3: [200]
-- @p4: [2.0000]
-- @p5: [0]
```

LINQ to SQL – Delete

```
// Den Gast definieren, der gelöscht werden soll
Gast zuLöschenderGast =
    Gaeste.Single(gast => gast.Name == "Ernst");

// Die zu löschende Gast-Entity
// an die Tabellen-Instanz übergeben
Gaeste.DeleteOnSubmit(zuLöschenderGast);

// Der DataContext-Instanz mitteilen, dass sie
// alle Änderungen in die Datenbank persistieren soll
dataContext.SubmitChanges();
```

```
DELETE FROM [Gaeste]
WHERE [Id]          = @p0
  AND [Name]        = @p1
  AND [Stammkneipe] = @p2
  AND [Besuche]     = @p3
  AND [Schulden]    = @p4

-- @p0: [1]
-- @p1: [Ernst]
-- @p2: [1]
-- @p3: [250]
-- @p4: [5.00]
```