

LIHQ

Language-INtegrated Query

LINQ?

**LINQ werkt op elke collectie die IEnumerable
IEnumerable<T>, IQueryable, of IQueryable<T>
interface implementeert**

**Niet uitsluitend voor databanken, kan ook
objecten of op XML bestanden gebruikt
worden.**

Een eerste voorbeeldje

```
string[] colors =  
    {  
        "Red",  
        "Brown",  
        "Orange",  
        "Yellow",  
        "Black",  
        "Green",  
        "White",  
        "Violet",  
        "Blue"  
    };
```

Een eerste voorbeeldje

```
IEnumerable<string> results =  
    from string c in colors  
    where c.StartsWith("B")  
    orderby c  
    select c;  
  
foreach (var color in results){  
    Console.WriteLine(color);  
}
```

Demo

Overeenkomst met SQL

FROM, WHERE, SELECT

- Bij SQL: eerst select, dan from en dan pas where
- Bij LINQ: eerst from, dan where en dan pas select

Reden voor omwisseling van keywords?

- IntelliSense ;-)

Je kan ook expliciet data type van c opgeven:

```
from string c in colors
```

Hier is dat niet nodig omdat de collection typed is

=> compiler kan zelf type achterhalen

- Je had hier evengoed var kunnen gebruiken

Uitgestelde uitvoering

In Engels: deferred execution

LINQ voert de query niet uit totdat applicatie resultaten nodig heeft

Consequentie

- Indien verschillende keren over resultaat geïtereerd wordt, wordt data source opnieuw benaderd en kan resultaat verschillen

Voorbeeld

```
foreach (var color in results) {  
    Console.WriteLine(color);  
}  
colors[4] = "Slate";  
txtLog.AppendText("-----" + Environment.NewLine);  
foreach (var color in results) {  
    Console.WriteLine (color);  
}
```

Resultaat zal 2 maal verschillend zijn (Demo)

Daar query opnieuw uitgevoerd wordt op data die ondertussen veranderd is

Doel van deferred execution: up-to-date data!!!

Alternatief

Je kan deferred execution omzeilen door na LINQ query `ToList()` te plaatsen

```
List<string> results = (from string c in colors
                        where c.StartsWith("B")
                        orderby c
                        select c).ToList();
foreach (var color in results) {
    Console.WriteLine(color);
}
```


LINQ Providers

LINQ providers

LINQ Enabled ADO.NET

LINQ to
Objects

LINQ to
DataSets

LINQ to
SQL

LINQ to
Entities

LINQ to
XML

LINQ Providers

LINQ to Objects

- Mogelijkheid om LINQ queries uit te voeren op .NET objecten en collecties

LINQ to DataSet

- Mogelijkheid om LINQ queries te schrijven voor tabellen en kolommen in DataSet

LINQ to Entities

- Mogelijkheid om LINQ queries uit te voeren over een Entity Data Model (zie Entity Framework, zie volgende les)

LINQ to SQL

- Mogelijkheid om LINQ queries te doen over data in een MS SQL Server databank

LINQ to XML

- Mogelijkheid om XML tags en attributen te ondervragen

Wat maakt LINQ LINQ?

Wat werd er aan .NET toegevoegd om LINQ te maken?

- Object Initializers
- Implicit Typed Local Variable Declarations
- Anonymous Types
- Lambda Expressions
- Extension Methods
- Query Extension Methods

Object Initializers

In kort: initialiseren van enkele/alle properties van een object in hetzelfde statement dat het object zelf instantieert

Voorbeeld (zonder object initializer):

```
public class Car {  
    public string VIN { get; set; }  
    public string Make { get; set; }  
    public string Model { get; set; }  
    public int Year { get; set; }  
    public string Color { get; set; }  
}
```

Voorbeeld vervolg

```
Car c = new Car();  
c.VIN = "ABC123";  
c.Make = "Ford";  
c.Model = "F-250";  
c.Year = 2000;
```

5 statements nodig om object te creëren en initialiseren

Merk op: color is niet geïnitialiseerd!

Doek open: object initializer

Object initializer: voorbeeld

```
Car c = new Car() {  
    VIN = "ABC123",  
    Make = "Ford",  
    Model = "F-250",  
    Year = 2000 };
```

Collection initializer

Zelfde als object initializer maar dan voor collections

Voorbeeld

```
private List<Car> GetCars() {  
    return new List<Car> {  
        new Car {VIN = "ABC123", Make = "Ford", Model = "F-250", Year = 2000},  
        new Car {VIN = "DEF123", Make = "BMW", Model = "Z-3", Year = 2005},  
        new Car {VIN = "ABC456", Make = "Audi", Model = "TT", Year = 2008},  
        new Car {VIN = "HIJ123", Make = "VW", Model = "Bug", Year = 1956},  
        new Car {VIN = "DEF456", Make = "Ford", Model = "F-150", Year = 1998}  
    };  
}
```

Gebruik van object initializers in LINQ?

Projectie (of *shaping*):

- Transformatie van data in een LINQ query om enkel die zaken te krijgen die je nodig hebt (mbv select statement) ipv alles terug te krijgen

Voorbeeld

- Je wilt enkel die kleuren die woordlengte hebben van 5 karakters, gesorteerd op kleur

Voorbeeld van projectie mbv LINQ

```
string[] colors = {  
    "Red",  
    "Brown",  
    "Orange",  
    "Yellow",  
    "Black",  
    "Green",  
    "White",  
    "Violet",  
    "Blue"  
};
```

Voorbeeld van projectie mbv LINQ

```
IEnumerable<Car> fords = from c in colors
                        where c.Length == 5
                        orderby c
                        select new Car() {
                            Make = "Ford",
                            Color = c
                        };

foreach (Car car in fords) {
    Console.WriteLine(String.Format("Car: Make:{0}
                                   Color:{1}", car.Make, car.Color));
}
```

Implicit Typed Local Variable Declarations

```
Car c = new Car();  
List<Car> cars = GetCars();
```

Je kan ook compiler type laten bepalen

```
var cars = GetCars();
```

Regels

- Alleen toepassen op lokale variabelen
- Declaratie statement: gelijkheidsteken + niet null assignment
- Kan niet toegepast worden op parameters van methodes

Impliciete datatypes zijn nodig om anonieme types te ondersteunen => nodig voor LINQ

Anonieme types

Voorbeeld:

- Properties van auto's aan datagrid koppelen
- Maar je wilt niet alle properties zien
- Slechts enkele

Code:

```
var x = new {Make = "VW", Model = "Bug"};  
Console.WriteLine(x.Make + ", " + x.Model);
```

IntelliSense zal in 2^{de} lijn na x. zelf aanvullen met Make of Model

Impliciet typed: je kent naam van anonieme type niet

Anonieme types en LINQ

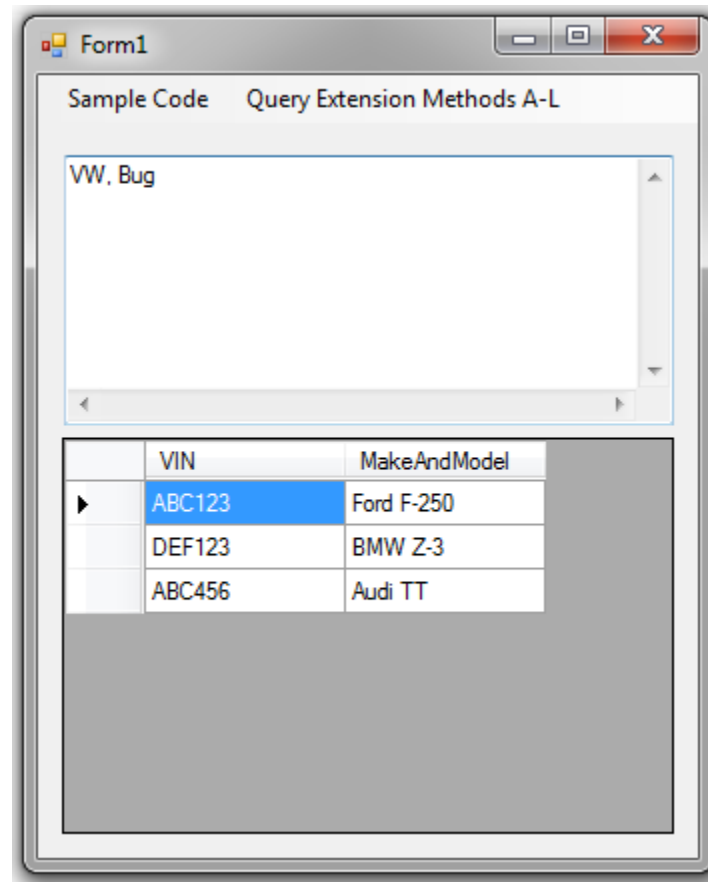
Anonieme types worden gebruikt voor projecties

Voorbeeld:

```
var carData = from c in GetCars()
               where c.Year >= 2000
               orderby c.Year
               select new {
                   c.VIN,
                   MakeAndModel = c.Make + " " + c.Model
               };
dgResults.DataSource = carData.ToList();
```

Anonieme types en LINQ

Resultaat:



Form1

Sample Code Query Extension Methods A-L

VW, Bug

	VIN	MakeAndModel
▶	ABC123	Ford F-250
	DEF123	BMW Z-3
	ABC456	Audi TT

Lambda expressies

Lijken op anonieme methodes, maar hebben een speciale, korte syntax, die ze heel geschikt maken om inline te gebruiken

```
var cars = GetCars();  
var theYear = 2000;  
var found = cars.Find(c => c.Year == theYear);  
txtLog.AppendText(string.Format("Car VIN:{0} Make:{1}  
Year{2}" + Environment.NewLine, found.VIN, found.Make,  
found.Year));
```

Lambda expressies

Lambda expressie: inline methode

Linkse deel:

- Definieert parameters

Na => teken:

- Expressie

Ook mogelijk om meerdere parameters door te geven

```
var found = cars.Find(c=>
{
    int x;
    x = theYear;
    return c.Year == x;
});
```


Extension methods

Deze methodes laten je toe om methodes toe te voegen aan een type, zelfs als je niet over de source code beschikt

Stel:

- Je wilt checken of de waarde van een string variabele numeriek is
- Hoe doe je dat? (klassieke manier)
- Hulpklasse StringHelper maken, met daarin statische methode IsNumeric

StringHelper klasse

```
public static class StringHelper{  
    public static bool IsNumeric(string str){  
        double val;  
        return double.TryParse(str, out val);  
    }  
}
```

Toepassen van statische methode:

```
string s = "abc123";  
txtLog.AppendText(StringHelper.IsNumeric(s) +  
Environment.NewLine);  
s = "123";  
txtLog.AppendText(StringHelper.IsNumeric(s) +  
Environment.NewLine);
```

Dit kan beter!!!

Gebruik maken van extension methodes

```
public static class StringHelper{  
    public static bool IsNumeric(this string str){  
        double val;  
        return double.TryParse(str, out val);  
    }  
}
```

Toepassen van deze extension methodes

```
string s = "abc123";  
txtLog.AppendText(s.IsNumeric() + Environment.NewLine);  
s = "123";  
txtLog.AppendText(s.IsNumeric() + Environment.NewLine);
```

Regels om met extension methodes te werken

- Extension methodes moeten gedefinieerd worden in een C# statische klasse
- De statische klasse moet publiek zijn gedefinieerd
- Als je een extension methode definieert voor een type, maar het type bezit al een methode met dezelfde naam, dan wordt deze laatste methode gebruikt, en wordt de extension methode genegeerd
- Net zoals de klasse is de extension methode statisch

Query extension methodes

MS heeft query extension methodes aan IEnumerable gehangen door de extension methodes te implementeren in een klasse genaamd Enumerable, die zich in System.Linq namespace bevindt

Belangrijkste geïmplementeerde Query Extension Methods

ALL (geeft boolean terug)

```
txtLog.WriteLine(GetCars().All(c => c.Year > 1960));
```

ANY (geeft boolean terug)

```
txtLog.WriteLine(GetCars().Any(c => c.Year <= 1960));
```

AVERAGE

```
var averageYear = GetCars().Average(c => c.Year);  
txtLog.WriteLine(averageYear);
```

CAST

```
IEnumerable<Car> cars = GetCars();  
IEnumerable<Object> objects = cars.Cast<object>();
```

CONCAT

```
int[] lastYearScores = { 88, 56, 23, 99, 65 };  
int[] thisYearScores = { 93, 78, 23, 99, 90 };  
foreach (var item in lastYearScores.Concat(thisYearScores)){  
    txtLog.WriteLine(item);  
}
```

Belangrijkste geïmplementeerde Query Extension Methods

CONTAINS

```
var cars = GetCars();  
Car c1 = cars[2];  
Car c2 = new Car();  
txtLog.WriteLine(cars.Contains(c1));  
txtLog.WriteLine(cars.Contains(c2));
```

COUNT

```
var cars = GetCars();  
txtLog.WriteLine(cars.Count());
```

DISTINCT

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
foreach (var score in scores.Distinct()) {  
    txtLog.WriteLine(score);  
}
```

ELEMENTAT

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
txtLog.WriteLine(scores.ElementAt(4));
```

Belangrijkste geïmplementeerde Query Extension Methods

EXCEPT

```
int[] lastYearScores = { 88, 56, 23, 99, 65 };  
int[] thisYearScores = { 93, 78, 23, 99, 90 };  
foreach (var item in lastYearScores.Except(thisYearScores)){  
    txtLog.WriteLine(item);  
}
```

FIRST

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
txtLog.WriteLine(scores.First());
```

GROUPBY

```
var cars = GetCars();  
var query = cars.GroupBy(c => c.Make);  
foreach (IGrouping<string, Car> group in query) {  
    txtLog.WriteLine("Key:{0}", group.Key);  
    foreach (Car c in group) {  
        txtLog.WriteLine("Car VIN:{0} Make:{1}", c.VIN, c.Make);  
    }  
}
```


Belangrijkste geïmplementeerde Query Extension Methods

INTERSECT

```
int[] lastYearScores = { 88, 56, 23, 99, 65 };  
int[] thisYearScores = { 93, 78, 23, 99, 90 };  
foreach (var item in lastYearScores.Intersect(thisYearScores)){  
    txtLog.WriteLine(item);  
}
```

JOIN

```
var makes = new string[] { "Audi", "BMW", "Ford", "Mazda", "VW" };  
var cars = GetCars();  
var query = makes.Join(cars,  
    make => make, car => car.Make,  
    (make, innerCar) => new { Make = make, Car = innerCar });  
  
foreach (var item in query) {  
    txtLog.WriteLine("Make: {0}, Car:{1} {2} {3}",  
        item.Make, item.Car.VIN, item.Car.Make, item.Car.Model);  
}
```

Belangrijkste geïmplementeerde Query Extension Methods

LAST

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
txtLog.WriteLine(scores.Last());
```

LONGCOUNT

```
var cars = GetCars();  
txtLog.WriteLine(cars.LongCount());
```

MAX

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
txtLog.WriteLine(scores.Max());
```

```
var cars = GetCars();  
txtLog.WriteLine(cars.Max(c => c.Year));
```

MIN

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
txtLog.WriteLine(scores.Min());
```

Belangrijkste geïmplementeerde Query Extension Methods

OFTYPE

```
object[] items = new object[] { 55, "Hello", 22, "Goodbye" };
foreach (var intltem in items.OfType<int>()){
    txtLog.WriteLine(intltem);
}
```

ORDERBY, ORDERBYDESCENDING, THENBY, AND THENBYDESCENDING

```
var cars = GetCars().OrderBy(c => c.Make)
                    .ThenByDescending(c => c.Model)
                    .ThenBy(c => c.Year);
foreach (var item in cars){
    txtLog.WriteLine("Car VIN:{0} Make:{1} Model:{2} Year:{3}",
        item.VIN, item.Make, item.Model, item.Year);
}
```

REVERSE

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };
foreach (var item in scores.Reverse()){
    txtLog.WriteLine(item);
}
```

Belangrijkste geïmplementeerde Query Extension Methods

SELECT

```
var vehicles = new List<Tuple<string, string, int>> {  
    Tuple.Create("123", "VW", 1999),  
    Tuple.Create("234", "Ford", 2009),  
    Tuple.Create("567", "Audi", 2005),  
    Tuple.Create("678", "Ford", 2003),  
    Tuple.Create("789", "Mazda", 2003),  
    Tuple.Create("999", "Ford", 1965) };  
  
var fordCars = vehicles.Where(v => v.Item2 == "Ford")  
    .Select(v => new Car {  
        VIN = v.Item1,  
        Make = v.Item2,  
        Year = v.Item3  
    });  
  
foreach (var item in fordCars) {  
    txtLog.WriteLine("Car VIN:{0} Make:{1} Year:{2}",  
        item.VIN, item.Make, item.Year);  
}
```

Belangrijkste geïmplementeerde Query Extension Methods

SUM

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
txtLog.WriteLine(scores.Sum());
```

TOARRAY

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
var evenScores = scores.Where(s => s % 2 == 0).ToArray();  
scores[2] = 2;  
foreach (var item in evenScores){  
    txtLog.WriteLine(item);  
}
```

Belangrijkste geïmplementeerde Query Extension Methods

TODICTIONARY

```
var cars = GetCars();  
var carsByVin = cars.ToDictionary(c => c.VIN);  
Car myCar = carsByVin["HIJ123"];  
txtLog.WriteLine("Car VIN:{0}, Make:{1}, Model:{2} Year:{3}",  
    myCar.VIN, myCar.Make, myCar.Model, myCar.Year);
```

TOLIST

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
var evenScores = scores.Where(s => s % 2 == 0).ToList();  
scores[2] = 2;  
foreach (var item in evenScores) {  
    txtLog.WriteLine(item);  
}
```

Belangrijkste geïmplementeerde Query Extension Methods

UNION

```
int[] lastYearScores = { 88, 56, 23, 99, 65, 56 };  
int[] thisYearScores = { 93, 78, 23, 99, 90, 99 };  
var allScores = lastYearScores.Union(thisYearScores);  
foreach (var item in allScores.OrderBy(s => s)){  
    txtLog.WriteLine(item);  
}
```

WHERE

```
var cars = GetCars();  
foreach (var myCar in cars.Where(c => c.Make == "Ford")) {  
    txtLog.WriteLine("Car VIN:{0}, Make:{1}, Model:{2}, Year:{3}",  
        myCar.VIN, myCar.Make, myCar.Model, myCar.Year);  
}
```

LINQ EN .NET OBJECTEN

LINQ en Objecten

Deze provider laat toe om queries uit te voeren over .NET arrays, collecties, generische collecties, en alles dat de IEnumerable of IQueryable interface implementeert

Voorbeelddata

```
var transport = new[] {  
    new { Name = "Car", Wheels = 4, SpeedClass = 3 },  
    new { Name = "Motorcycle", Wheels = 2, SpeedClass = 3 },  
    new { Name = "Bike", Wheels = 2, SpeedClass = 2 },  
    new { Name = "Unicycle", Wheels = 1, SpeedClass = 1 },  
    new { Name = "Tricycle", Wheels = 3, SpeedClass = 1 },  
    new { Name = "Semi", Wheels = 18, SpeedClass = 3 } };  
  
var speed = new[] {  
    new { ClassID = 1, Name = "Low", LowMaxSpeed = 1,  
          HighMaxSpeed = 10 },  
    new { ClassID = 2, Name = "Medium", LowMaxSpeed = 11,  
          HighMaxSpeed = 50 },  
    new { ClassID = 3, Name = "High", LowMaxSpeed = 51,  
          HighMaxSpeed = 150 } };
```

Resultaten projecteren mbv Select

Projectie:

- Transformatie van originele kolommen in een nieuwe subset van kolommen
- In de output kolommen kunnen ook berekeningen zitten (avg, sum, ...)

```
var results = from sp in speed
               select new { Name = sp.Name.ToUpper(),
                           sp.LowMaxSpeed, sp.HighMaxSpeed,
                           SpeedRange = (sp.HighMaxSpeed -
                                           sp.LowMaxSpeed + 1)
               };
```

Alternatieve schrijfwijze m.b.v. lambda expressies

```
var results = speed.Select(sp => new { Name = sp.Name.ToUpper(),  
                                     sp.LowMaxSpeed, sp.HighMaxSpeed,  
                                     SpeedRange = sp.HighMaxSpeed -  
                                     sp.LowMaxSpeed + 1 } );
```

**Combinatie van lambda expressies en extension
methodes**

Output van een specifiek type met projectie

```
IEnumerable<SimpleClass> results =  
    from tr in transport  
    select new SimpleClass {  
        Name = tr.Name,  
        NumValue = tr.SpeedClass  
    };
```

Output van projectie direct gebruiken

```
foreach (var oneVehicle in (from tr in transport select tr)) {  
    ...  
}
```

Filteren van resultaten m.b.v. where clause

Where clause (zie ook SQL):

- Filtering van originele collectie
- Gebruik makend van filter

```
var results = from tr in transport
               where tr.SpeedClass == 1
               select tr;
```

Alternatieve schrijfwijze (mbv lambda expressies en extension methode)

```
var results = transport.Where(tr => tr.SpeedClass == 1);
```

Filters combineren

```
var results = from tr in transport
               where tr.SpeedClass == 1 && tr.Name.EndsWith("cycle")
               select tr;
```

LINQ: Where

```
class WhereSample2 {  
    static void Main() {  
        // Data source.  
        int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
        // Create the query with two predicates in where clause.  
        var queryLowNums2 = from num in numbers  
                             where num < 5 && num % 2 == 0  
                             select num;  
  
        // Execute the query  
        foreach (var s in queryLowNums2) {  
            Console.Write(s.ToString() + " ");  
        }  
    }  
}  
  
// Output: 4 2 0
```


where (2)

```
class WhereSample3 {  
    static void Main() {  
        // Data source  
        int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
        // Create the query with a method call in the where  
        clause.  
        var queryEvenNums = from num in numbers  
                             where IsEven(num)  
                             select num;  
  
        // Execute the query.  
        foreach (var s in queryEvenNums) {  
            Console.Write(s.ToString() + " ");  
        }  
    }  
}
```

where (3)

```
// Method may be instance method or static method.  
static bool IsEven(int i) {  
    return i % 2 == 0;  
}  
}  
//Output: 4 8 6 2 0
```

Sorteren van resultaten met OrderBy

OrderBy sorteert de gefilterde en geprojecteerde resultaten

```
var results = from tr in transport
               orderby tr.SpeedClass descending, tr.Name
               select tr;
```

Alternatieve schrijfwijze met lambda expressies

```
var results = transport.OrderByDescending(
    tr => tr.SpeedClass).ThenBy(tr => tr.Name);
```

LINQ en joins

Let op: volgende code is geen inner join, maar een cross join

```
var results = from tr in transport
               from sp in speed
               select new { tr.Name, tr.SpeedClass,
                           SpeedName = sp.Name };
```

Dit is wel een inner join:

```
var results = from tr in transport
               from sp in speed
               where tr.SpeedClass == sp.ClassID
               select new { tr.Name, tr.SpeedClass,
                           SpeedName = sp.Name };
```

Alternatieve schrijfwijze met Join

```
var results = from tr in transport
               join sp in speed
               on tr.SpeedClass equals sp.ClassID
               select new { tr.Name, tr.SpeedClass,
                           SpeedName = sp.Name };

```

Limiet op inhoud van queryresultaat

```
var results = (from tr in transport
               orderby tr.Wheels
               select tr.Wheels).Distinct();

// ----- Returns just the first result, not a
// collection.
var result = (from tr in transport select tr).First();

// ----- Counts the returned records.
int result = (from tr in transport select tr).Count();
```

Aggregatiefuncties

```
// ----- What is the maximum wheel count on any vehicle?  
int result = transport.Max(tr => tr.Wheels);  
  
// ----- Do any vehicles have three wheels?  
bool result = transport.Any(tr => tr.Wheels == 3);
```

group

group clause geeft een sequentie van de volgende objecten terug:

- `IGrouping<TKey, TElement>`

Voorbeeld

```
// Query variable is an IEnumerable<IGrouping<char, Student>>  
var studentQuery1 = from student in students  
                    group student by student.Last[0];
```


group (2)

```
// Group students by the first letter of their last name
// Query variable is an IEnumerable<IGrouping<char, Student>>
var studentQuery2 = from student in students
                    group student by student.Last[0] into g
                    orderby g.Key
                    select g;

// Iterate group items with a nested foreach. This IGrouping
// encapsulates a sequence of Student objects, and a Key of type char.
// For convenience, var can also be used in the foreach statement.
foreach (IGrouping<char, Student> studentGroup in studentQuery2) {
    Console.WriteLine(studentGroup.Key);
    // Explicit type for student could also be used here.
    foreach (var student in studentGroup) {
        Console.WriteLine("    {0}, {1}", student.Last, student.First);
    }
}
```

group (3)

```
public void Linq77() {  
    List<Product> products = GetProductList();  
    var categoryCounts = from p in products  
                          group p by p.Category into g  
                          select new {  
                              Category = g.Key,  
                              ProductCount = g.Count()  
                          };  
}
```

Category=Beverages	ProductCount=12
Category=Condiments	ProductCount=12
Category=Produce	ProductCount=5
Category=Meat/Poultry	ProductCount=6
Category=Seafood	ProductCount=12
Category=Dairy Products	ProductCount=10

into

Tijdelijke variabele om resultaten van een group, select of join in op te slaan

```
class IntoSample1 {  
    static void Main() {  
        // Create a data source.  
        string[] words = { "apples", "blueberries", "oranges",  
                           "bananas", "apricots"};  
  
        // Create the query.  
        var wordGroups1 =  
            from w in words  
            group w by w[0] into fruitGroup  
            where fruitGroup.Count() >= 2  
            select new { FirstLetter = fruitGroup.Key,  
                        Words = fruitGroup.Count() };  
    }  
}
```

Into (2)

```
// Execute the query. Note that we only iterate over the
// groups, not the items in each group
foreach (var item in wordGroups1) {
    Console.WriteLine(" {0} has {1} elements.",
                      item.FirstLetter, item.Words);
}

// Keep the console window open in debug mode
Console.WriteLine("Press any key to exit.");
Console.ReadKey();
}
}

/* Output:
  a has 2 elements.
  b has 2 elements.
*/
```

Codevoorbeeld: orderby en group by gecombineerd

```
class OrderbySample2 {  
  
    // The element type of the data source.  
    public class Student {  
        public string First { get; set; }  
        public string Last { get; set; }  
        public int ID { get; set; }  
    }  
}
```

Codevoorbeeld: orderby en group by gecombineerd

```
public static List<Student> GetStudents() {  
    // Use a collection initializer to create the data source. Note  
    // that each element in the list contains an inner sequence of  
    // scores.  
    List<Student> students = new List<Student> {  
        new Student {First="Svetlana", Last="Omelchenko", ID=111},  
        new Student {First="Claire", Last="O'Donnell", ID=112},  
        new Student {First="Sven", Last="Mortensen", ID=113},  
        new Student {First="Cesar", Last="Garcia", ID=114},  
        new Student {First="Debra", Last="Garcia", ID=115}  
    };  
    return students;  
}
```

Codevoorbeeld: orderby en group by gecombineerd

```
static void Main(string[] args){  
    // Create the data source.  
    List<Student> students = GetStudents();  
    // Create the query.  
    IEnumerable<Student> sortedStudents =  
        from student in students  
        orderby student.Last ascending, student.First ascending  
        select student;  
    // Execute the query.  
    Console.WriteLine("sortedStudents:");  
    foreach (Student student in sortedStudents)  
        Console.WriteLine(student.Last + " " + student.First);  
}
```

Codevoorbeeld: orderby en group by gecombineerd

```
// Now create groups and sort the groups. The query first sorts the  
// names of all students so that they will be in alphabetical order  
// after they are grouped. The second orderby sorts the group keys in  
// alpha order.
```

```
var sortedGroups = from student in students  
                   orderby student.Last, student.First  
                   group student by student.Last[0] into newGroup  
                   orderby newGroup.Key  
                   select newGroup;
```


Codevoorbeeld: orderby en group by gecombineerd

```
// Execute the query.
Console.WriteLine(Environment.NewLine + "sortedGroups:");
foreach (var studentGroup in sortedGroups) {
    Console.WriteLine(studentGroup.Key);
    foreach (var student in studentGroup) {
        Console.WriteLine("    {0}, {1}", student.Last, student.First);
    }
}

// Keep the console window open in debug mode
Console.WriteLine("Press any key to exit.");
Console.ReadKey();
}
}
```

Codevoorbeeld: orderby en group by gecombineerd

sortedStudents:

Garcia Cesar

Garcia Debra

Mortensen Sven

O'Donnell Claire

Omelchenko Svetlana

sortedGroups:

G

Garcia, Cesar

Garcia, Debra

M

Mortensen, Sven

O

O'Donnell, Claire

Omelchenko, Svetlana

LINQ

**Bekijk de voorbeelden die je kan vinden op 101
LINQ Samples**

(<http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>)

Probeer voorbeelden te begrijpen

LINQ TO DATASET

LINQ to DataSet provider

Om LINQ te kunnen gebruiken op een datasource

- Nood aan: Collection die IEnumerable of IQueryable interfaces implementeert
- **Niet het geval bij DataTables**

LINQ zal ook niet automatisch de datatypes achterhalen van een gegeven kolom (dwz expliciet casten)

LINQ to DataSet provider voorziet

- Nieuwe extension methodes voor DataTable en DataRow

Queries schrijven met LINQ to DataSet

Gebruik hiervoor de extension methode
AsEnumerable

```
// Customer is an existing DataTable instance.  
var results = from cu in Customer.AsEnumerable()  
              select cu;
```

Focus ligt eigenlijk op DataTable ipv DataSet

LINQ houdt geen rekening met relatie(s) tussen
verschillende DataTables

Zelf joins schrijven indien nodig!

Joins tussen DataTables

```
// Explicit join.  
var results = from cu in Customer.AsEnumerable()  
              join ord in Order.AsEnumerable()  
              on cu.ID equals ord.CustomerID  
              select...  
  
// Implicit join  
var results = from cu in Customer.AsEnumerable()  
              from ord in Order.AsEnumerable()  
              where cu.ID == ord.CustomerID  
              select...
```

Casten van een veld in een record

Casten gebeurt door de Field extension methode

```
var results = from cu in Customer.AsEnumerable()  
              orderby cu.Field<string>("FullName")  
              select new { CustomerName =  
                           cu.Field<string>("FullName") };
```


LINQ to Objects en DataSet

```
// Build an ad hoc collection, although you could also include a
// fully realized class.
var statusTable[] = {
    new { Code = "P", Description = "Active Order" },
    new { Code = "C", Description = "Completed / Shipped" },
    new { Code = "X", Description = "Canceled" }
};

// Link ADO.NET and Object collections in one query.
var results = from ord in Order.AsEnumerable()
               join sts in statusTable on
               ord.Field<string>("StatusCode") equals sts.Code
               orderby ord.Field<long>("ID")
               select new { OrderID = ord.Field<long>("ID"),
                           CurrentStatus = sts.Description };
```

LINQ TO SQL

LINQ to SQL

Voor SQL Server

- 3 mogelijke LINQ providers
 - LINQ to DataSet
 - LINQ to Entities (zie later)
 - LINQ to SQL
 - Speciaal gebouwd om met SQL Server tabellen te interageren, en de queries tonen die dichte verwantschap mooi aan
 - Voorziet een raamwerk om relationele data als objecten te beheren, terwijl je nog queries kan uitvoeren op de data

LINQ to SQL

Object-relational mapping (ORM) tool

- Enerzijds: objecten en klassen (applicatie)
- Anderzijds: data die opgeslagen is als records in tabellen
- Mapping tussen beiden vinden
 - Niet evident!
 - Structuur van gegevens in objecten wijkt af van de structuur van gegevens in databank
 - Gegevens uit 1 object kunnen verspreid zijn over meerdere tabellen

Datacentrisch vs objectcentrisch

Data modelleren

**LINQ to SQL model genereren vanuit een
bestaande databank**

Demo

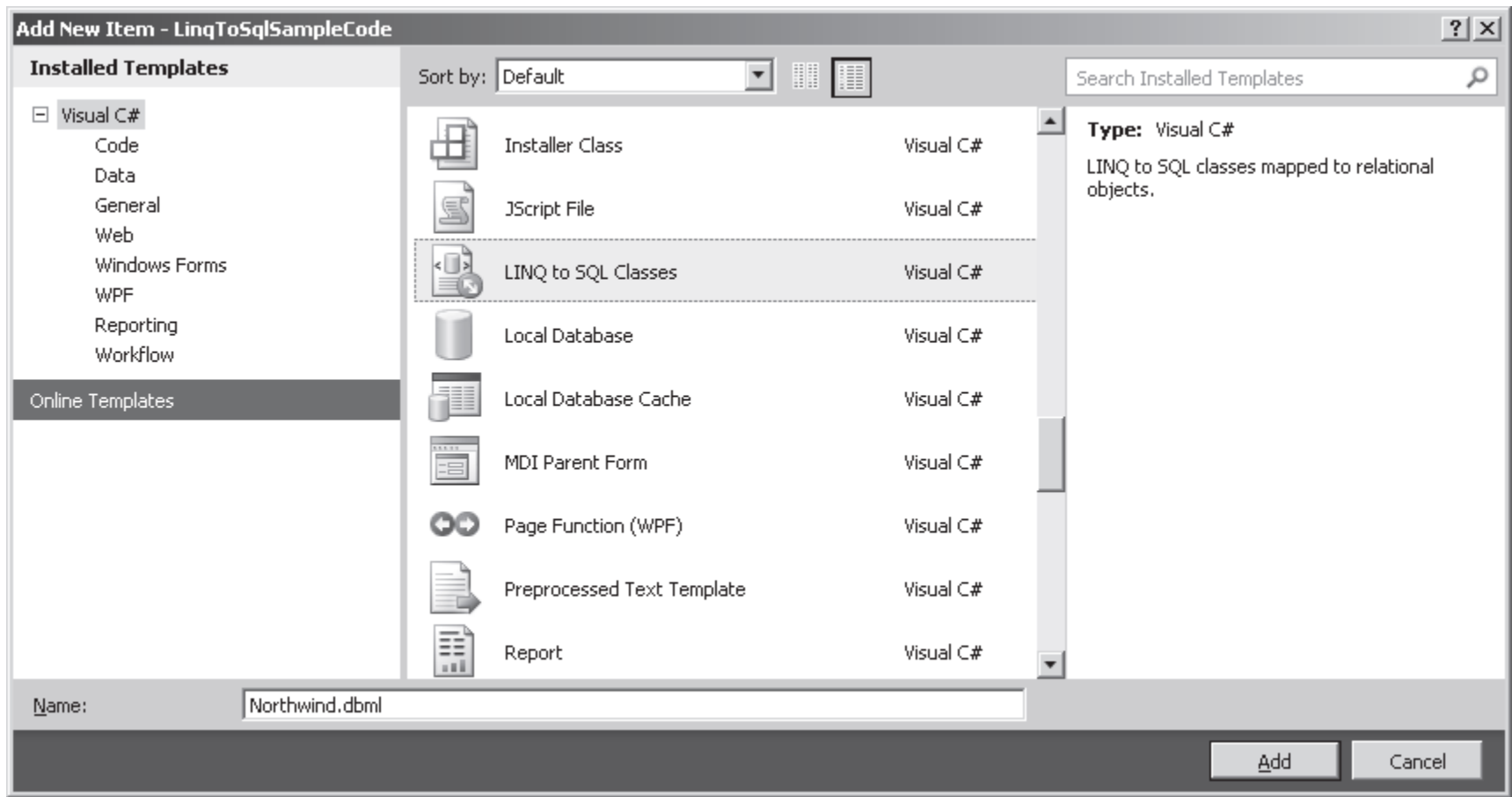
Stappen

Nieuw project maken

Kies Add > New Item > LINQ to SQL Classes

Noem het NorthWind.dbml

Stappen



Stappen

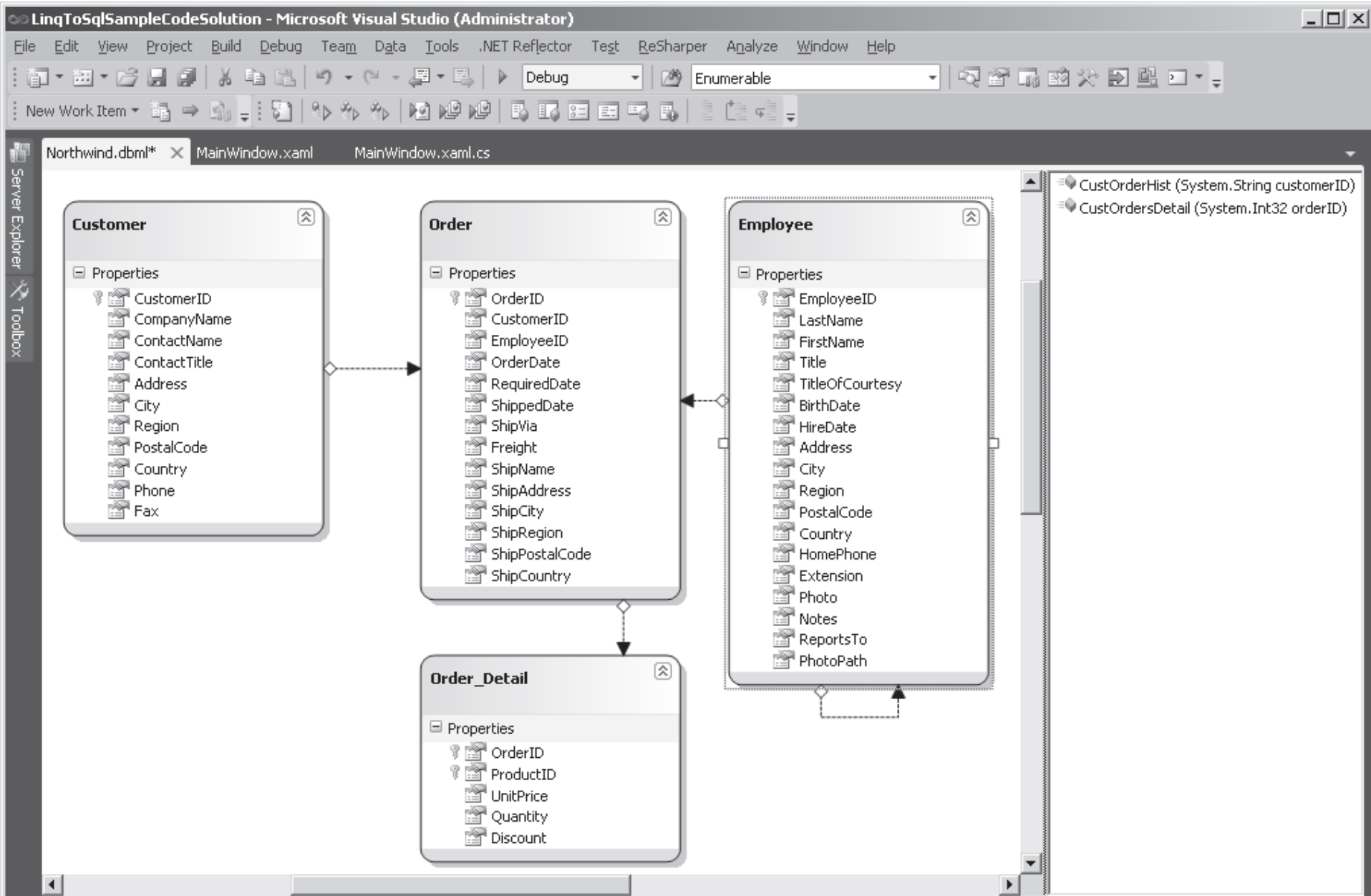
Open in Server Explorer databank die je wilt gebruiken

Kies bijv. Northwind databank en sleep de volgende tabellen op linkergedeelte van het venster

- Customers, Orders, Order Details, Employees

En sleep de volgende stored procedures op het rechter deel van het venster:

- CustOrderHist, CustOrderDetail



NorthWind.designer.cs

Deze file bevat alle informatie over de mapping tussen databank en objecten in C#

Klasse:

- Customer: velden + constructoren
- Order: idem
- Order_Detail: idem
- Employee: idem

Relaties tussen de klassen

DataContext klasse

Belangrijkste klasse in dit bestand

- NorthWindDataContext klasse
- Zorgt voor het versturen van data van en naar SQL Server databank
- Connection string bevindt zich in app.config file

Basis query met filter en sorteren

```
var ctx = new NorthwindDataContext();  
var customers = from c in ctx.Customers  
                where c.CompanyName.Contains("Restaurant")  
                orderby c.PostalCode  
                select c;  
dg.DataSource = customers;
```

Projectie

In vorige query werden alle kolommen teruggegeven:

- Stel dat we enkel CustomerID, CompanyName, en PostalCode willen
- Projectie toepassen

Projectie

```
var ctx = new NorthwindDataContext();  
var customers = from c in ctx.Customers  
                where c.CompanyName.Contains("Restaurant")  
                orderby c.PostalCode  
                select new {  
                    c.CustomerID,  
                    c.CompanyName,  
                    c.PostalCode  
                };  
dg.DataSource = customers;
```

Inner Join mbv query extension methode

```
var ctx = new NorthwindDataContext();
var customers = ctx.Customers.Join(ctx.Orders,
    c => c.CustomerID,
    o => o.CustomerID,
    (c, o) => new {
        c.CustomerID,
        c.CompanyName,
        o.OrderID,
        o.OrderDate
    })
    .OrderBy(r => r.CustomerID)
    .ThenBy((r => r.OrderID));
dg.DataSource = customers;
```

Alternatief mbv LINQ query

```
var ctx = new NorthwindDataContext();  
var customers = from c in ctx.Customers  
                join o in ctx.Orders  
                on c.CustomerID equals o.CustomerID  
                orderby c.CustomerID, o.OrderID  
                select new {  
                    c.CustomerID,  
                    c.CompanyName,  
                    o.OrderID,  
                    o.OrderDate  
                };  
dg.DataSource = customers;
```


Grouping en aggregatie

```
var ctx = new NorthwindDataContext();  
var orders = from o in ctx.Order_Details  
              group o by o.OrderID  
              into grouped  
              select new {  
                  OrderID = grouped.Key,  
                  Total = grouped.Sum(  
                      line => line.Quantity * line.UnitPrice *  
                          (1 - (decimal)line.Discount))  
              };  
dg.DataSource = orders;
```

LINQ TO SQL: VERANDERINGEN DOORGEVEN AAN SQL SERVER

LINQ to SQL

Voorbeelden tot nu toe:

- Enkel data ophalen uit databank

LINQ is enkel geschikt om data op te halen

BEHALVE LINQ to SQL

- Voorziet manieren om INSERT, DELETE en UPDATE operaties uit te voeren

DataContext

Bezit identity table

- Houdt bij welke records reeds uit databank gehaald zijn (via PK van record)
- Indien zelfde record nog eens gevraagd wordt via DataContext object, wordt niet meer naar databank gegaan, maar wordt record uit cache gehaald
- Toestand van objecten wordt bijgehouden a.d.h.v. zijn status:
 - Untracked, Unchanged, PossiblyModified, ToBeInserted, ToBeUpdated, ToBeDeleted, Deleted

Een voorbeeld om te starten

```
private Customer customer;  
private NorthwindDataContext ctx = new NorthwindDataContext();  
customer = (from c in ctx.Customers  
            where c.CustomerID == "ALFKI"  
            select c).First();
```

Wijzigen van customer object

```
customer.ContactName = "Marty " + DateTime.Now.ToLongTimeString();  
ctx.SubmitChanges();
```

Door uitvoeren van SubmitChanges worden veranderingen terug naar databank geschreven

Nieuw object toevoegen aan DataContext

```
var ctx = new NorthwindDataContext();  
var employee = new Employee {  
    FirstName = "John",  
    LastName = "Smith"  
};  
ctx.Employees.InsertOnSubmit(employee);  
ctx.SubmitChanges();
```

Met InsertOnSubmit om object (met status: Untracked) kenbaar te maken aan DataContext

Object verwijderen

```
var ctx = new NorthwindDataContext();  
var employee = (from emp in ctx.Employees  
                 where emp.EmployeeID == 10  
                 select emp).First();  
ctx.Employees.DeleteOnSubmit(employee);  
ctx.SubmitChanges();
```


Wat met stored procedures?

Hoe gebruiken we geïmporteerde stored procedures?

Als een methode van DataContext

```
var ctx = new NorthwindDataContext();  
dg.DataSource = ctx.CustOrderHist("ALFKI");
```

LINQ TO XML

XElement

Opening- en sluittag

- `<test>` en `</test>`

Tags kunnen genest worden

Er is een speciale tag

- `<test/>`

C# voorbeelden

Root node

```
XElement root = new XElement("Record");
```

Child node

```
XElement child1 = new XElement("Name");  
root.Add(child1);
```

Als je dit laat uitschrijven naar console:

```
<Record>  
  <Name />  
</Record>
```

C# voorbeelden

```
XElement root=new XElement("Record");  
XElement child1=new XElement("Name");  
root.Add(child1);  
XElement child2=new XElement("First");  
XElement child3=new XElement("Second");  
child1.Add(child2);  
child1.Add(child3);  
XElement child4=new XElement("Address");  
root.Add(child4);
```



`child1.Add(child2,child3);`

Corresponderende XML file

<Record>

<Name>

<First />

<Second />

</Name>

<Address />

</Record>

Alternatief

```
XElement root = new XElement("Record",  
    new XElement("Name",  
        new XElement("First"),  
        new XElement("Second")),  
    new XElement("Address"));
```

Load & Parse methode

XElement heeft 2 interessante methodes

- **Load**: zal een XML file inlezen en omzetten naar XElement'en
- **Parse**: vertrekt van een string die wordt omgezet naar XElement'en

Voorbeeld van Parse methode

```
string XML = @"  
    <Name>  
        <First />  
        <Second />  
    </Name>  
    <Address />  
</Record>";  
XElement root= XElement.Parse(XML) ;
```

Nu nog inhoud...

Voor het ogenblik enkel structuur en nog geen inhoud tussen de tags

In een XElement boom wordt de tekst tussen tags bewaard als een XText node

Voorbeeld

```
XElement root = new  
    XElement("Record", "Address Record",  
        new XElement("Name",  
            new XElement("First", "Mike"),  
            new XElement("Second")),  
        new XElement("Address"));
```

Attributen

2 manieren om een attribuut toe te voegen

```
1. XAttribute Att1=new XAttribute("Epoc",2000);  
root.Add(Att1);
```

- Dit resulteert in:

```
<Record Epoc="2000"> ...<Record>
```

```
2. XElement root = new XElement("Record",  
    new XAttribute("Epoc", 2000),  
    new XElement("Name",  
        new XElement("First","Mike"),  
        new XElement("Second")),  
    new XElement("Address"));
```

Boom-manipulatie

In de boom kan je allerlei zaken toevoegen, aanpassen of verwijderen

```
root.Add(child4);  
child4.Remove();
```

Aanpassen van waarden:

```
root.SetElementValue("Tel", "123");
```

- Dit zal aan een bestaande xml kind van de root met de naam Tel de waarde 123 toevoegen. Als dit element niet bestaat zal het eerst gemaakt worden.

Analoog voor attributen:

```
root.SetAttributeValue("Epoc", "2008");
```

Linq To XML

- Linq queries worden mogelijk gemaakt door extension methods die toegepast worden op objecten die IEnumerable interfaces implementeren
- In het geval van XML implementeren de objecten deze interface niet, maar sommige van hun methodes retourneren objecten die dat wel doen
- Vb: Elements methode geeft IEnumerable collectie terug van alle child elementen van het object

Voorbeeld

```
foreach( XElement ele in root.Elements())  
{  
    textBox1.Text += ele.ToString();  
}
```

Voorbeeld: resultaat

```
<Record>  
  <Name>  
    <First />  
    <Second />  
  </Name>  
  <Address />  
</Record>
```


LINQ

```
var q = root.Elements().  
    Where<XElement>(E=>E.Name=="Address");  
foreach( XElement ele in q){  
    textBox1.Text += ele.ToString();  
}
```

of

```
var q = from E in root.Elements()  
        where E.Name == "Address"  
        select E;
```

of

```
var q=root.Elements("Address");
```

BRONNEN

Referenties

Microsoft ADO.NET 4 Step by Step

MCTS Self-Paced Training Kit (Exam 70-516):

Accessing Data with Microsoft .NET Framework

4

[http://www.i-](http://www.i-programmer.info/programming/c/1641-linq-and-xml.html?start=1)

[programmer.info/programming/c/1641-linq-](http://www.i-programmer.info/programming/c/1641-linq-and-xml.html?start=1)

[and-xml.html?start=1](http://www.i-programmer.info/programming/c/1641-linq-and-xml.html?start=1)

