

Thorsten Kansy

LINQ



Thorsten Kansy

# LINQ

Direkte Abfragen mit  
Language Integrated Query

[entwickler.press](http://entwickler.press)

Thorsten Kansy: LINQ  
Direkte Abfragen mit Language Integrated Query  
ISBN: 978-3-86802-006-9

© 2009 entwickler.press  
Ein Imprint der Software & Support Verlag GmbH

Bibliografische Information Der Deutschen Bibliothek  
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen  
Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über  
<http://dnb.ddb.de> abrufbar.

Ihr Kontakt zum Verlag und Lektorat:  
Software & Support Verlag GmbH  
entwickler.press  
Goethestraße 28  
80336 München  
Tel: +49(0) 89 66 55 76 10  
Fax: +49(0) 89 66 55 76 11  
[lektorat@entwickler-press.de](mailto:lektorat@entwickler-press.de)  
<http://www.entwickler-press.de>

Lektorat: Sandra Michel, [smichel@entwickler-press.de](mailto:smichel@entwickler-press.de)  
Fachkorrektorat: Walter Saumweber  
Korrektorat: mediaService, Siegen  
Satz: mediaService, Siegen  
Umschlaggestaltung: Melanie Hahn, Maria Rudi  
Belichtung, Druck & Bindung: M.P. Media-Print Informationstechnologie GmbH,  
Paderborn

Alle Rechte, auch für Übersetzungen, sind vorbehalten. Reproduktion jeglicher Art (Fotokopie, Nachdruck, Mikrofilm, Erfassung auf elektronischen Datenträgern oder andere Verfahren) nur mit schriftlicher Genehmigung des Verlags. Jegliche Haftung für die Richtigkeit des gesamten Werks kann, trotz sorgfältiger Prüfung durch Autor und Verlag, nicht übernommen werden. Die im Buch genannten Produkte, Warenzeichen und Firmennamen sind in der Regel durch deren Inhaber geschützt.

*„You are my inspiration,  
You are my battle cry“*

*Für Ursula*



# Inhaltsverzeichnis

<b>1</b>	<b>Vorwort und Einleitung</b>	<b>13</b>
<b>2</b>	<b>Einführung</b>	<b>17</b>
2.1	Überblick	17
2.2	Die technischen Bausteine von LINQ	19
	Erweiterungsmethoden	19
	Lambda-Funktionen	22
	Lokale Typinferenz	25
	Anonyme Typen	28
	Kompaktere Initialisierung von Objekten	31
2.3	Die LINQ-Schnittstellen	33
2.4	Abfrage-Syntax	34
2.5	Sequenz vs. Auflistung	38
2.6	Verzögerte Ausführung	39
2.7	Sofortige Ausführung	40
2.8	Performance	41
	LINQ to Objects	42
	LINQ to SQL	43
<b>3</b>	<b>Operatoren</b>	<b>45</b>
3.1	Projektionsoperatoren	46
	Select	47
	SelectMany	49
3.2	Filteroperatoren	53
	Where	53
	OfType	56
3.3	Sortierungsoperatoren	56
	OrderBy/OrderBy..Descending	57
	ThenBy/ThenByDescending	60
	Reverse	64
3.4	Gruppierungsoperatoren	65
	GroupBy	65
3.5	Join-Operatoren	71
	Join	71
	GroupJoin	75

3.6	Mengenoperatoren	78
	Distinct	78
	Union	81
	Intersect	84
	Except	87
3.7	Aggregationsoperatoren	90
	Count/LongCount	90
	Sum	92
	Min/Max	95
	Average	100
	Aggregate	103
3.8	Generierungsoperatoren	107
	Range	107
	Repeat	108
	Empty	111
3.9	Quantifizierungsoperatoren	111
	All	111
	Any	113
	Contains	115
3.10	Aufteilungsoperatoren	118
	Take	118
	TakeWhile	119
	Skip	121
	SkipWhile	122
3.11	Elementoperatoren	125
	First/Last	125
	FirstOrDefault/LastOrDefault	128
	ElementAt	130
	ElementAtOrDefault	131
	Single	132
	SingleOrDefault	134
	DefaultIfEmpty	137
3.12	Konvertierungsoperatoren	139
	ToArray	139
	ToList	140
	ToDictionary	141
	ToLookup	144
	AsEnumerable	148
	AsQueryable	149
	Cast	150
3.13	Sonstige Operatoren	151
	Concat	151
	SequenceEqual	152

<b>4</b>	<b>LINQ-Operatoren entwickeln</b>	<b>157</b>
4.1	Grundlagen	157
	Parametername	157
	So generisch wie möglich implementieren	158
	Generische Datentypen	158
	Den Yield-Operator verwenden	159
	Verzögerte Ausführung	160
	Alle Parameter überprüfen	162
	Func, Predicate und Action	162
4.2	CreateSequence-Operator I	165
4.3	CreateSequence-Operator II	167
4.4	Identity-Operator	168
4.5	Standardoperatoren überladen	169
<b>5</b>	<b>LINQ to Objects</b>	<b>173</b>
<b>6</b>	<b>LINQ to SQL</b>	<b>179</b>
6.1	DataContext	183
	Erzeugen	185
	Unabhängige Verbindungen zur Datenbank	186
6.2	Entitäten-Klassen	189
	Database-Attribut	190
	Table-Attribut	190
	Column-Attribut	191
	Association-Attribut	192
	Das InheritanceMapping-Attribut	195
	Die Function- und Parameter-Attribute	196
	Entitäten-Klassen mit Business-Logik erweitern	204
	Datenbankstruktur aus Metadata erstellen/löschen	208
6.3	Entitäten-Klassen generieren	208
	Manuell per Code	209
	SqlMetal-Anwendung	209
	Mit Visual Studio 2005/2008 arbeiten	211
6.4	Parallelitätskonflikte	216
	Pessimistisch	216
	Ignorant	216
	Optimistisch	217
6.5	Transaktionen	218
	ADO.NET-Transaktionen	218
	TransactionScope	219
6.6	Daten abfragen, modifizieren und in die Datenbank schreiben	220
	LINQ-Abfragen	220
	Optionen für das Laden von Daten	220
	Verzögertes Laden umgehen	222

	Daten verändern	223
	Änderungen in Datenbank schreiben	225
	Daten erneut aus Datenbank laden	229
	Abfragen kompilieren	230
	Disconnected Mode	232
	Abfragen protokollieren	233
<b>7</b>	<b>LINQ to DataSets</b>	<b>237</b>
7.1	DataSets – Grundlagen	237
7.2	Übersicht	239
7.3	Untypisierte DataSets	240
	Struktur per Code anlegen	241
	DataSet befüllen	242
	Daten mit LINQ abfragen	244
	Daten ändern	245
	Änderungen speichern	247
7.4	Typisierte DataSets	249
	Struktur anlegen	249
	DataSet befüllen	251
	Daten mit LINQ abfragen	251
	Daten ändern	252
	Änderungen speichern	253
<b>8</b>	<b>LINQ to XML</b>	<b>255</b>
8.1	LINQ to XML im Einsatz	256
	XML abfragen	257
	XML erzeugen	258
8.2	X-Klassen	261
	Die XDocument-Klasse	262
	XDeclaration-Klasse	270
	XDocumentType-Klasse	272
	XComment-Klasse	275
	XElement-Klasse	277
	XAttribute-Klasse	284
	XCDATA-Klasse	287
8.3	Änderungen überwachen	290
<b>9</b>	<b>LINQ to Entities</b>	<b>293</b>
9.1	Entities vs. Entities	293
9.2	Mehr als nur CRUD	295
9.3	Datenmodelle	296
9.4	Der grafische Designer	297
	Entitäten, Zuordnungen, Vererbungen	299
	Gespeicherte Prozeduren	301

9.5	Modellbrowser	301
9.6	Abfragen	302
9.7	Werte ändern	303
9.8	Objekte hinzufügen	303
9.9	Objekte löschen	305
9.10	Änderungen in der Datenbank speichern	305
<b>10</b>	<b>Datenbindung</b>	<b>307</b>
10.1	Windows Forms	307
	Via DataSource-Eigenschaft	307
	Via DataBindings	312
10.2	ASP.NET	313
10.3	WPF	317
	Eigenständige WPF-Anwendung	319
	XBAP-Anwendung	323
<b>A</b>	<b>LINQ-Operatoren</b>	<b>329</b>
	<b>Stichwortverzeichnis</b>	<b>333</b>



# 1

## Vorwort und Einleitung

Willkommen! Schön, dass Sie einen Blick in dieses Buch werfen.

Es hat, wie der Titel schon verrät, die neue Abfragesprache LINQ zum Inhalt und versucht, mit einfachen Worten sowie mit vielen anschaulichen Beispielen dem Leser dieses Thema näher zu bringen. Von mir, dem Autor, wurde alles Wissenswerte mit Sorgfalt zu Papier gebracht. Dabei nutze ich meine langjährige Erfahrung als Entwickler und Softwarearchitekt, um auf die Fragen Antworten zu finden, auf die Sie in Ihrer täglichen Arbeit mit LINQ stoßen können. Hilfreich waren dabei auch die unzähligen Schulungen und Gespräche mit Entwicklern. Es ist also ein Buch von einem Programmierer für Programmierer.

Ich hoffe, dass sich dies alles positiv in diesem Werk bemerkbar macht.

### Für wen ist dieses Buch?

Dieses Entwicklerbuch richtet sich sowohl an den Anfänger, der bereits etwas Erfahrung in der C#-Programmierung mitbringt, als auch an den Profi, der auf der Höhe der Zeit bleiben möchte.

In ein paar Stichworte gefasst, sollte der „optimale“ Leser die folgenden Eigenschaften mitbringen:

- Interesse und Spaß an der Technik und Programmierung
- Neugier und Geduld
- Kenntnisse in C#
- einige Kenntnisse in SQL und den relationalen Datenbanken, vielleicht sogar SQL Server

Aber keine Panik, wenn Ihnen die eine oder andere Grundlage fehlt – dieses Buch lässt Sie an den entscheidenden Stellen nicht im Stich.

### Warum dieses Buch?

Dieses Buch deckt alle wichtigen Aspekte von LINQ ab. Kapitel 2 erläutert die technischen Grundlagen. Kapitel 3 und 4 beschreiben alle Standardoperatoren (LINQ-Befehle) und zeigen Ihnen, wie Sie eigene Operatoren selbst implementieren können. Die anschließenden Kapitel gehen dann auf Details von LINQ für Objekte, LINQ für SQL, LINQ für DataSets, LINQ für XML etc. ein. Das letzte Kapitel – sozusagen der krönende Abschluss – zeigt, wie Daten aus LINQ-Abfragen direkt an Steuerelemente der Oberfläche gebunden werden können.

Das Buch ist also so etwas wie das Rundum-sorglos-Paket für den LINQ-interessierten Entwickler.

Um Ihnen, dem Leser, das Lesen und Suchen nach bestimmten (Kern-)Informationen zu erleichtern, folgt dieses Buch einigen Konventionen bezüglich der verwendeten Schriftarten und Formatierungen.

Quelltexte werden auf diese Weise hervorgehoben.

**Wichtige** Details oder **Änderungen** werden zusätzlich noch **fett** hervorgehoben, damit Sie schneller ins Auge fallen.

Schlüsselwörter, Methoden, Eigenschaften und Programmausgaben werden mit diesem Format versehen.

*Links für Ressourcen im Internet erscheinen in diesem Format.*

*Wird mit Querverweisen auf andere, zum aktuellen Thema passende Inhalte verwiesen, so kommt dieses Format zum Einsatz.*

Tastaturkürzel wie z.B. `STRG+A` sind ebenfalls hervorgehoben.

Besonders wichtige Informationen werden in Kästen wie den folgenden geliefert.

### **Achtung**

Um Sie vor Fehlern oder Trugschlüssen zu bewahren, finden Sie Warnungen in solchen Kästen. Diese sollten Sie auf jeden Fall lesen.

### **Hinweis**

Hinweise, die Ihnen Ihre tägliche Arbeit mit LINQ einfacher machen können, werden auf diese Weise präsentiert.

### **Tipp**

Tipps und Tricks haben in solchen Kästen ihren Platz. Mit diesen Tricks lässt sich LINQ noch einen Tick einfacher und besser verstehen.

Desweiteren wurden alle Anstrengungen unternommen, um dieses Werk möglichst fehlerfrei drucken zu können. Sollten Sie (was ich natürlich nicht hoffe) Fehler/Unstimmigkeiten finden, Anregungen oder Kritik (auch positive) für mich haben oder Unterstützung benötigen, so freue ich mich über eine E-Mail unter [tkansy@dotnetconsulting.eu](mailto:tkansy@dotnetconsulting.eu). Zudem empfängt meine Webseite <http://www.dotnetconsulting.eu> rund um die Uhr Besucher.

---

## Schulung, Consulting & Co

Sie können mich, meine Erfahrung und mein Know-How übrigens auch buchen. Ich biete meinen Kunden weit gefächerte Dienstleistungen rund um .NET und SQL Server an. Dazu gehören auch, jedoch nicht ausschließlich:

- Schulungen rund um .NET (ASP.NET, AJAX, LINQ, WCF, WF, ..) in C# und VB.NET und SQL Server in jeder Version
- Projektbegleitende Unterstützung
  - ▶ Analyse
  - ▶ Softwarearchitektur
  - ▶ Prototyping
  - ▶ Coaching
  - ▶ Hotline
- Übersichtvermittlung
- Vorträge

Sprechen Sie mich einfach an, über E-Mail oder meine Webseite, wie Sie möchten.

## Danksagung

Als Letztes möchte ich mich noch, bevor ich Sie den nächsten Kapiteln überlasse, bei allen bedanken, die mich bei meiner Arbeit unterstützt haben. Allen voran den Mitarbeitern des Software & Support Verlages und meinen Freunden. Ohne sie wäre ich wohl niemals fertig geworden.

Viersen, im August 2008

*Thorsten Kansy*

MCPD MCTS MCAD MCSD MCDBA MCSE+I MCT  
Consultant, Software Designer, Trainer, Fachautor

*tkansy@dotnetconsulting.eu*  
*<http://www.dotnetconsulting.eu>*



# 3 Operatoren

Die vielfältigen Funktionen, die LINQ bereitstellt, werden Operatoren genannt. Diese Operatoren verändern oder erzeugen eine Sequenz und leiten sie oftmals an den nächsten Operator weiter, der möglicherweise wiederum dasselbe tut. Die Operatoren zu kennen, ist daher eine wichtige Voraussetzung, um erfolgreich mit LINQ zu programmieren.

Dieses Kapitel stellt alle Operatoren mit ihren entsprechenden Überladungen vor. Für eine bessere Übersicht sind diese Operatoren in Gruppen eingeteilt. Diese Gruppeneinteilung soll helfen, schnell den gesuchten Operator für eine bestimmte Aufgabe zu finden.

Gruppe	Funktionen
Projektionsoperatoren	Projektion der Elemente in die abschließende Form.
Filteroperatoren	Beliebiges Filtern mit jeglichem C#-Ausdruck, der true oder false liefern kann.
Sortierungsoperatoren	Auf- oder absteigende, mehrfache Sortierung.
Gruppierungsoperatoren	Gruppierung nach gleichen Schlüsselwerten.
Join-Operatoren	Beziehungen zwischen Sequenzen über Schlüsselwerte.
Mengenoperatoren	Operatoren zum Erstellen von Schnitt- und Vereinigungsmengen etc.
Aggregationsoperatoren	Funktionen zum Berechnen von Summen, Minimum und Maximum etc.
Generierungsoperatoren	Operatoren zum Erzeugen von Sequenzen.
Quantifizierungsoperatoren	Bewertung von Sequenzen. Sind z.B. bestimmte Bedingungen erfüllt oder bestimmte Elemente vorhanden?
Aufteilungsoperatoren	Funktionalitäten zum Aufteilen von Sequenzen, also z.B. die ersten oder letzten n Elemente.
Elementoperatoren	Zugriff auf einzelne Elemente einer Sequenz ohne foreach-Schleife (Iterator).
Konvertierungsoperatoren	Operatoren zum Konvertieren von Sequenzen in Auflistungen (oder umgekehrt).
Sonstige Operatoren	Sonstige Operatoren, die in keine der anderen Gruppen passen.

**Tabelle 3.1:** Die Gruppen, in die die LINQ-Operatoren eingeteilt werden

Für alle Operatoren werden deren Signaturen abgedruckt. So ist leicht zu erkennen, ob und wie ein Operator überladen ist. Wie aber sind solche Signaturen zu lesen?

Neben den Aspekten der generischen Programmierung ist zu berücksichtigen, dass alle Operatoren Erweiterungsmethoden sind und somit der erste Parameter nach dem this-

Schlüsselwort die Schnittstelle angibt, die erweitert wird. In den meisten Fällen ist dies die `IEnumerable<>`-Schnittstelle.

```
public static IEnumerable<TResult> Select<TSource, TResult>(
    this IEnumerable<TSource> source, // Diese Schnittstelle wird erweitert
    Func<TSource, TResult> selector); // Func<> ist ein Delegat
```

`Func<>` in der Signatur bezeichnet ein generisches Delegat, dessen letzter Parameter den Typ der Rückgabe angibt. Diese Delegaten können mit gewöhnlichen oder anonymen Methoden und einfachen Lambda-Ausdrücken verwendet werden. Da Lambda-Ausdrücke wohl die häufigste Variante bei LINQ sind, sehen Sie im Folgenden den Ausdruck, der zu der Signatur oben passt:

```
TSource => TResult oder (TSource) => TResult
```

Hat `Func<TResult>` nur einen Parameter, so ist dies ebenfalls wieder der Rückgabewert (da dieser ja auch wieder den letzten Parameter darstellt). Der Lambda-Ausdruck sieht in diesem Fall so aus:

```
() => TResult
```

Das leere Klammernpaar ist hier unabdingbar, da der Compiler sonst einen Fehler erzeugt.

Als letzte Möglichkeit können mehr als drei Parameter vorliegen. In diesem Fall werden die Parameter eines Lambda-Ausdrucks in runde Klammern gefasst. Für `Func<TSource1, TSource2, TSource3, TResult>` sieht der Ausdruck entsprechend so aus:

```
(TSource1, TSource2, TSource3) => TResult
```

Da zu jeder soliden Programmierung auch die Berücksichtigung von möglichen Fehlern in Form einer Ausnahmebehandlung gehört, finden Sie im Folgenden eine Auflistung der möglichen Ausnahmen, die der Operator auslösen kann, zusammen mit einer kurzen Erklärung in welcher Situation er dies tut.

Zur Abrundung werden mittels kleiner Beispiele die letzten Unklarheiten über die sinnvolle Anwendung beseitigt. Dabei werden die wichtigen Stellen fett hervorgehoben. Damit Sie beim Lesen das gute Gefühl bekommen, verstanden zu haben, was der Operator tut, haben die meisten Beispiele eine Ausgabe, die sich im Anschluss des Codes befindet (aus Platzgründen oft in leicht gekürztem Umfang).

## 3.1 Projektionsoperatoren

Dieser Abschnitt beschreibt die so genannten Projektoperatoren, deren Aufgabe es ist, die gewünschten Inhalte aus einer Quellsequenz in eine Ergebnissequenz zu übertragen. Dies geschieht nach Filterung und Sortierung durch andere Operatoren – Projektionsoperatoren werden daher meist abschließend ausgeführt, um zu bestimmen, wie das

Endergebnis aussehen soll. Ihre Ausgabe kann jedoch auch wiederum die Grundlage (Quelle) einer weiteren LINQ-Abfrage darstellen.

Für beide Operatoren gilt, dass sie erst direkt beim Durchlaufen einer `foreach`-Schleife oder beim Aufruf der `GetEnumerator()`-Methode ausgeführt werden und immer nur jeweils ein Element nach dem anderen liefern. Sie folgen damit den Regeln der verzögerten Ausführung und ihr Ergebnis kann sich zwischen zwei Aufrufen verändern, wenn die Quellsequenz verändert wurde.

### Hinweis

Wird eine stabile Auflistung benötigt, die sich nicht bei Veränderungen an der Quellsequenz ändert, so kann dies mit einem Konvertierungsoperator wie `ToArray`, `ToList`, `ToDictionary` oder `ToLookup` realisiert werden. Diese erzeugen eine Auflistung im klassischen Sinne. Für Details siehe Abschnitt 3.12 *Konvertierungsoperatoren*.

## 3.1.1 Select

Der `Select`-Operator projiziert das Ergebnis auf eine Sequenz, die die `IEnumerable<T>`-Schnittstelle implementiert. Dabei wird eine flache Liste erstellt, die z.B. mit einer `foreach`-Schleife durchlaufen werden kann. Daher ist der `Select`-Operator oftmals der letzte in der Reihe, um die gefilterten und aufbereiteten Elemente in die finale Ergebnissequenz zu projizieren. Durch eine zweite Überladung können die Elemente beginnend mit null durchnummeriert werden oder der laufende Index kann für eine Berechnung verwendet werden.

Dieser Operator steht sowohl in der LINQ-Syntax (für die erste Überladung) als auch als Methode zur Verfügung.

### Überladungen

Hier die Signaturen der beiden Überladungen des `Select`-Operators.

Überladung 1:

```
public static IEnumerable<TResult> Select<TSource, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, TResult> selector);
```

Überladung 2:

```
public static IEnumerable<TResult> Select<TSource, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, int, TResult> selector);
```

### Ausnahmen

Es wird eine Ausnahme vom Typ `ArgumentNullException` ausgelöst, wenn `source` oder `selector` lediglich null sind.

## Beispiele

Überladung 1:

Das folgende Beispiel liefert die Namen und die physikalischen Pfade aller Systemordner über die `System.Environment.GetFolderPath()`-Methode.

```
// .NET-Typ der Aufzählung speichern
Type enumType = typeof(Environment.SpecialFolder);

// LINQ-Syntax
var folders1 = from sf in Enum.GetNames(enumType)
    select new {
        Enum = sf,
        Pfad = Environment.GetFolderPath(
            (Environment.SpecialFolder)Enum.Parse(enumType,sf))
    };

// Methode
var folders2 = Enum.GetNames(enumType).Select(sf =>
    new {
        Enum = sf,
        Pfad = GetFolderPath((SpecialFolder)Enum.Parse(enumType, sf))
    });

// Ausgabe
foreach (var folder in folders1)
    Debug.WriteLine(folder);
```

Die Ausgabe ist in beiden Fällen gleich und sieht exemplarisch so aus.

```
{ Enum = Desktop, Pfad = C:\Users\tkansy\Desktop }
{ Enum = Personal, Pfad = C:\Users\tkansy\Documents }
{ Enum = MyDocuments, Pfad = C:\Users\tkansy\Documents }
```

Überladung 2:

In der zweiten Überladung kann zusätzlich auf den laufenden Index für die Erstellung der Zielelemente zugegriffen werden. Dies wird ausgenutzt, um über einen Lambda-Ausdruck die Namen der Sequenz durchzunummerieren.

```
// Array mit Namen erstellen
String[] Namen = { "Julia", "Ursula", "Christiane", "Christine" };
var namen = Namen.Select(
    (name, index) => new { Vorname = String.Format("#{0}, {1}", index, name) }
);
```

```
// Ausgabe
foreach (var name in namen)
    Debug.WriteLine(name);
```

Die Ausgabe besteht also aus nummerierten Namen:

```
{ Vorname = #Julia, 0 }
{ Vorname = #Ursula, 1 }
{ Vorname = #Christiane, 2 }
{ Vorname = #Christine, 3 }
```

### 3.1.2 SelectMany

Während der Select-Operator im Kern aus einer Liste eine andere erzeugt, geht der SelectMany-Operator einen Schritt weiter. Er arbeitet auf Sequenzen, deren einzelne Elemente wiederum ebenfalls Sequenzen darstellen. Er durchläuft diese verschachtelten Sequenzen und erzeugt dabei eine flache Sequenz, die der Reihe nach die Elemente der einzelnen Sequenzen enthält.

Dieser Operator steht nur als Methode zur Verfügung.

#### Überladungen

Für diesen Operator existieren vier Überladungen.

Überladung 1:

```
public static IEnumerable<TResult> SelectMany<TSource, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, IEnumerable<TResult>> selector);
```

Überladung 2:

```
public static IEnumerable<TResult> SelectMany<TSource, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, int, IEnumerable<TResult>> selector);
```

Überladung 3:

```
public static IEnumerable<TResult> SelectMany<TSource, TCollection, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, IEnumerable<TCollection>> collectionSelector,
    Func<TSource, TCollection, TResult> resultSelector);
```

Überladung 4:

```
public static IEnumerable<TResult> SelectMany<TSource, TCollection, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, int, IEnumerable<TCollection>> collectionSelector,
    Func<TSource, TCollection, TResult> resultSelector);
```

## Ausnahmen

Es wird eine Ausnahme vom Typ `ArgumentNullException` ausgelöst, wenn **source**, **selector**, **collectionSelector** oder **resultSelector** lediglich null sind.

## Beispiele

Lassen Sie uns die ersten drei Überladungen jeweils an einem kleinen Beispiel demonstrieren.

Überladung 1:

Das Beispiel für die einfachste Überladung dieses Operators erzeugt aus einer Liste aus Sätzen eine flache Liste mit Wörtern. Alternativ wird die Ausgabe mit zwei verschachtelten `foreach`-Schleifen dargestellt, die ohne den `SelectMany`-Operator notwendig sind.

```
// Eine Liste an fiesen Zungenbrechern
string[] Zungenbrecher = new string[] {
    "Fischers Fritze fischt frische Fische.",
    "Brautkleid bleibt Brautkleid und Blaukraut bleibt Blaukraut.",
    "Der Kaplan klebt klappbare Pappplakate";

// Dieses Array aus Zungenbrechern lässt sich leicht in eine Sequenz
// aus String Arrays aufteilen.
IEnumerable<string[]> Wortlisten = Zungenbrecher.Select(w => w.Split(' '));

// Damit werden zwei verschachtelte foreach-Schleifen benötigt,
// um alle Wörter auszugeben
foreach (string[] Woerter in Wortlisten)
    foreach (string Wort in Woerter)
        Debug.WriteLine(Wort);

// Einfacher geht es mit dem SelectMany-Operator,
// der eine flache Liste erzeugt und nur eine Schleife
// erforderlich macht.
IEnumerable<string> flacheWortliste = Wortlisten.SelectMany( c =>c );

// Alternativ können einzelne Elemente der endgültigen flachen Liste
// z.B. von Punkten am Ende bereinigt werden.
```

```
IEnumerable<string> flacheWortliste2 =  
    Wortlisten.SelectMany( c=>c.Select( d=>d.TrimEnd('.') ) );  
  
// Ausgabe  
foreach (string Wort in flacheWortliste)  
    Debug.WriteLine(Wort);
```

Die Ausgabe der einfachen Schleife besteht aus einer Abfolge von Zeichenketten, welche aus den Wörtern der Zungenbrecher besteht.

```
Fischers  
Fritze  
fischt  
frische  
Fische.
```

Überladung 2:

Im Beispiel für die zweite Überladung kann auf den laufenden Index für die Erstellung der Zielelemente zugegriffen werden. Da sich dieser auf die Quellsequenzen bezieht, wird ein weiterer Select-Operator für den Index des Wortes innerhalb eines Zungenbrechers verwendet.

```
// Eine Liste an fiesen Zungenbrechern  
string[] Zungenbrecher = new string[] {  
    "Fischers Fritze fischt frische Fische.",  
    "Brautkleid bleibt Brautkleid und Blaukraut bleibt Blaukraut.",  
    "Der Kaplan klebt klappbare Pappplakate";  
  
// Dieses Array aus Zungenbrechern lässt sich leicht in eine Sequenz  
// aus String Arrays aufteilen.  
IEnumerable<string[]> Wortlisten = Zungenbrecher.Select(w => w.Split(' '));  
  
// Einfacher geht es mit dem SelectMany-Operator,  
// der eine flache Liste erzeugt und nur eine Schleife  
// erforderlich macht. Durch den zusätzlichen Einsatz des  
// Select-Operators kann neben dem laufenden Index des Zungenbrechers auch  
// auf den laufenden Index der Wörter zugegriffen werden.  
var flacheWortliste =  
    Wortlisten.SelectMany((z,n) =>  
        z.Select((w,m) => string.Format("Brecher# {0}, Wort# {1}: {2}",n,m,w)));  
// Ausgabe  
foreach (var Wort in flacheWortliste)  
    Debug.WriteLine(Wort);
```

In der Ausgabe ist zu erkennen, in welchem Zungenbrecher das Wort an wievielter Stelle steht. Die ersten sieben Zeilen der Ausgabe sehen so aus:

```
Brecher# 0, Wort# 0: Fischers
Brecher# 0, Wort# 1: Fritze
Brecher# 0, Wort# 2: fischt
Brecher# 0, Wort# 3: frische
Brecher# 0, Wort# 4: Fische.
Brecher# 1, Wort# 0: Brautkleid
Brecher# 1, Wort# 1: bleibt
```

#### Überladung 3:

Als Beispiel für die dritte Überladung dient ein kleines Array mit Objekten, die sowohl den Namen als auch eine Liste mit Automarken von berühmten Autobesitzern speichert. Mittels `SelectMany` wird daraus eine flache Liste. Der zweite Lambda-Ausdruck wird dabei verwendet, um Name und Marke miteinander in Beziehung zu setzen.

```
// Ein kleine Liste berühmter Autofahrer
AutoFahrer[] AutoFahrer = {
    new AutoFahrer { Name="Percy Pickwick",
        Autos = new List<string>{ "MG TD" } },
    new AutoFahrer { Name="James Bond",
        Autos = new List<string>{ "BMW Z8", "Lotus Esprit", "AMC Hornet" } }
};

// Mit dem SelectMany-Operator wird daraus eine flache Liste, die
// für jede Fahrer/Auto-Kombination ein anonymes Objekt enthält
var Fahrerliste =
    AutoFahrer.SelectMany(Fahrer => Fahrer.Autos, (Fahrer, Auto) =>
        new { Fahrer.Name, Auto });

// Alternativ kann natürlich auch z.B. die String.Format()-Methode verwendet
// werden, um eine entsprechende Liste von Zeichenketten zu erstellen
IEnumerable<string> Fahrerliste2 =
    AutoFahrer.SelectMany(Fahrer => Fahrer.Autos, (Fahrer, Auto) =>
        string.Format("{0} fährt {1}", Fahrer.Name, Auto ));

// Ausgabe
foreach (var s in Fahrerliste)
    Debug.WriteLine (s);

// Klasse zum Speichern der Fahrer und einer Liste von Autos
```

```
class AutoFahrer
{
    public string Name { get; set; }
    public List<string> Autos { get; set; }
}
```

Die Ausgabe besteht aus vier Zeilen mit dem für anonyme Objekte typischen Aufbau:

```
{ Name = Percy Pickwick, Auto = MG TD }
{ Name = James Bond, Auto = BMW Z8 }
{ Name = James Bond, Auto = Lotus Esprit }
{ Name = James Bond, Auto = AMC Hornet }
```

## 3.2 Filteroperatoren

Sollen nicht alle Elemente einer Sequenz verwendet werden, sondern nur solche, die einem bestimmten Filter entsprechen, so benutzt man Filteroperatoren. Der Kern eines Filters ist das Prädikat, das vom Typ `bool` ist und daher entweder wahr oder falsch sein kann. Für LINQ in Verbindung mit C# bedeutet dies, dass gewöhnliche C#-Bedingungen zum Einsatz kommen, die entweder `true` oder `false` liefern. Aus diesem Grund finden auch die üblichen Regeln über Auswertungsprioritäten Anwendung, wie sie für `if`-Abfragen oder `while/do..while`-Schleifen der Fall sind.

### 3.2.1 Where

Der einzige Standardfilteroperator steht sowohl in der LINQ-Syntax (für die erste Überladung) als auch als Methode zur Verfügung.

#### Überladungen

Für diesen Operator existieren zwei Überladungen.

Überladung 1:

```
public static IEnumerable<TSource> Where<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
```

Überladung 2:

```
public static IEnumerable<TSource> Where<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, int, bool> predicate);
```

## Ausnahmen

Es wird eine Ausnahme vom Typ `ArgumentNullException` ausgelöst, wenn **source** oder **predicate** lediglich null sind.

## Beispiele

Überladung 1:

Als Beispiel für die erste Überladung dient eine Abfrage auf alle laufenden Prozesse, deren Prozessname mit einem `, W'` beginnt und die mehr als fünf Threads verwenden.

```
// Namensraum für Zugriff auf Prozesse einführen
using System.Diagnostics;

// Alle laufenden Prozesse auslesen
Process[] proclList = Process.GetProcesses();

// Es sollen alle Prozesse ausgewählt werden,
// die mit einem 'W' beginnen und die mehr als fünf
// Threads verwenden.
IEnumerable<Process> ProclList1 = from proc in proclList
                                where proc.ProcessName.StartsWith("W") &&
                                      proc.Threads.Count > 5
                                select proc;

// Alternativ kann auch die Methoden-Syntax verwendet werden.
// In diesem Fall kann der Select-Operator entfallen, da hier sowieso
// nur die Process-Objekte unverändert ausgewählt werden.
IEnumerable<Process> ProclList2 = (proclList.Where(proc =>
    proc.ProcessName.StartsWith("W") && proc.Threads.Count > 5));

// Ausgabe
foreach (Process proc in ProclList1)
    // Statt der Ausgabe könnte auch alles andere unternommen werden,
    // was mit einem Process-Objekt möglich ist, z.B. beenden via Kill().
    Debug.Print("{0}: {1} (Threads: {2})",
        proc.Id, proc.ProcessName, proc.Threads.Count);
```

Die Ausgabe hängt natürlich davon ab, was derzeit auf dem Computer ausgeführt wird. Sie könnte in etwa so aussehen:

```
3496: WmiPrvSE (Threads: 6)
5724: WINWORD (Threads: 9)
```

## Überladung 2:

In der zweiten Überladung steht zusätzlich ein laufender, nullbasierter Index zur Verfügung. Dieser wird im Beispiel dafür genutzt, eine Liste der laufenden Prozesse auf maximal fünf zu begrenzen.

**Hinweis**

Der gleiche Effekt kann mittels des Take-Operators erreicht werden. Für Details siehe Abschnitt 3.10.1 *Take*.

```
// Namensraum für Zugriff auf Prozesse einführen
using System.Diagnostics;

// Alle laufenden Prozesse auslesen
Process[] proclList = Process.GetProcesses();

// Hier steht nur die Methoden-Syntax zur Verfügung.
// In diesem Fall kann der Select-Operator entfallen, da hier sowieso
// nur die Process-Objekte unverändert ausgewählt werden.
IEnumerable<Process> = (proclList.Where((proc, index) => index < 5));

// Mit einem Select-Operator könnte die Abfrage alternativ so aussehen.
// Die Sequenz enthält in diesem Fall dann nur noch String-Objekte.
IEnumerable<string> ProclList2 = (proclList.Where((proc, index) => index < 5)).
    Select(proc => string.Format(
        "{0}: {1} (Threads: {2})", proc.Id, proc.ProcessName, proc.Threads.Count));

// Ausgabe
foreach (Process proc in ProclList1)
    // Auch hier kann statt der Ausgabe alles andere unternommen werden,
    // was mit einem Process-Objekt möglich ist, z.B. beenden via Kill().
    Debug.Print(
        "{0}: {1} (Threads: {2})", proc.Id, proc.ProcessName, proc.Threads.Count);
```

Die Ausgabe entspricht der, die wir oben für die erste Überladung gezeigt haben und beschränkt sich im Übrigen auf die ersten fünf Prozesse, welche die `Process.GetProcesses()`-Methode liefert.

```
3740: LINQ.vshost (Threads: 13)
3148: svchost (Threads: 14)
4004: winampa (Threads: 1)
4720: iexplore (Threads: 16)
2748: SearchIndexer (Threads: 22)
```

### 3.2.2 OfType

Dieser Operator filtert die Elemente einer Sequenz nach einem bestimmten Datentyp. Alle Elemente, die von diesem Typ sind oder von diesem abgeleitet wurden, werden übernommen. Alternativ kann statt eines Typs auch eine Schnittstelle angegeben werden, so dass alle Elemente, die diese implementieren, übernommen werden.

#### Signatur

Dieser Operator ist nicht überladen. Es existiert nur eine Signatur:

```
public static IEnumerable<TResult> OfType<TResult>(
    this IEnumerable source);
```

#### Ausnahmen

Es wird eine Ausnahme vom Typ `ArgumentNullException` ausgelöst, wenn **source** lediglich null ist.

#### Beispiel

Das Beispiel zeigt, wie aus einem gemischten Array vom Typ `object` alle Elemente gefiltert werden, die vom Typ `string` sind.

```
// Gemischtes Array erzeugen
object[] DiesUndDas = { 1, "Ronald", true, "Ruck" };

// Sequenz mit allen Zeichenketten erstellen
IEnumerable<string> zeichenketten = DiesUndDas.OfType<string>();

// Ausgabe
foreach (string s in zeichenketten)
    Debug.WriteLine(s);
```

Die Ausgabe besteht aus den zwei Zeichenketten und hat eine auffällige Ähnlichkeit mit einer gewissen Ente:

```
Ronald
Ruck
```

## 3.3 Sortierungsoperatoren

Sollen Sequenzen in einer bestimmten Reihenfolge vorliegen, so müssen sie zuvor sortiert werden. Zu diesem Zweck stehen Sortierungsoperatoren zur Verfügung. Mit ihnen können beliebige Sortierungen durchgeführt oder die Reihenfolge der Elemente in einer Sequenz umgekehrt werden.



# Stichwortverzeichnis

## Numerisch

1:1-Beziehung 296

## A

Abfrage Syntax 34

Abfragen kompilieren 230

AcceptAllChanges() 305

AcceptChanges() 247

Action 164

Add

    XObjectChange 290

Add()

    DataTable 246

    List 174

    XDocument 267

    XElement 279

AddAfterSelf()

    XCData 287

    XComment 275

    XElement 279

AddBeforeSelf()

    XCData 287

    XComment 275

    XElement 279

AddFirst()

    XDocument 267

    XElement 279

AddRange()

    List 175

ADO.NET 186

    Transaktionen 218

Aggregate-Operator 103

Aggregationsoperatoren 90

All-Operator 111

Always 217

Ancestors()

    XCData 287

    XComment 275

    XElement 280

AncestorsAndSelf()

    XElement 280

Anonyme Typen 24, 28

Any-Operator 113

Application.Resources 321

ArgumentNullException 162

ArgumentOutOfRangeException 162, 172

ASCII 255

AsDataView() 240

AsEnumerable() 240

AsEnumerable-Operator 148

ASP.NET 313

AsQueryable-Operator 149

Association-Attribut 193

Attribute()

    XElement 280

Attributes()

    XElement 280

Auflistung 38

Aufteilungsoperatoren 118

AutoSync

    Column-Attribut 191

Average-Operator 100

## B

BaseUri

    XDocument 267

Boxing 25

Business-Logik 204

## C

CanBeNull

    Column-Attribut 191

Cast-Operator 150

ChangeConflictException 227

ChangeConflicts 183, 227

Changed-Ereignis 290

Changing-Ereignis 290

CheckBox-Steuerelement 308

Close() 187

clr-namespace 321

Code

    InheritanceMapping-Attribut 195

collectionSelector siehe selector

- Column-Attribut 191
  - ComboBox-Steuererelement 309
  - CommandTimeout 183
  - comparer 158
  - Compile() 231
  - Complete() 220
  - Concat-Operator 151
  - ConflictMode 225
  - Connection 183, 186
  - ConnectionString 185
  - Constraint 237
  - Contains-Operator 115
  - ContinueOnConflict 226
  - CopyToDataTable() 240
  - Count-Operator 90
  - CreateDatabase() 183
  - CreateReader()
    - XComment 275
    - XDocument 267
    - XElement 280
  - CreateWriter()
    - XDocument 267
    - XElement 280
  - Current() 159
  - CurrentValue 227
- D**
- Database-Attribut 190
  - DatabaseExists() 183
  - DatabaseValue 227
  - DataBindings 312
  - DataColumn 237
  - DataContext 183
  - DataLoadOptions 220
  - DataRelation 237
  - DataSets 237
  - DataSource 307
  - DataTable 237
  - DataTemplate 321
  - DataGridView 237
  - DbConnection 186
  - DBML 212
  - DBML.LAYOUT siehe DBML
  - DbType
    - Column-Attribut 191
    - Parameter-Attribut 196
  - Declaration
    - XDocument 267
  - default 128, 137
  - DefaultIfEmpty-Operator 137
  - DeferredLoadingEnabled 183
  - Delegaten, anonyme 24
  - DeleteDatabase() 183
  - DeleteOnNull
    - Association-Attribut 193
  - DeleteOnSubmit() 225
  - DeleteRule
    - Association-Attribut 193
  - DescendantNodes()
    - XElement 280
  - Descendants()
    - XElement 280
  - Dictionary 141
  - DisableFormatting
    - SaveOptions 265
  - Disconnected Mode 232
  - DisplayMemberBinding 318
  - Distinct-Operator 78
  - Document
    - XAttribute 285
    - XCDATA 287
    - XComment 276
    - XDocumentType 273
    - XElement 280
  - DocumentType
    - XDocument 267
  - DOM 259
  - DSNamespace 321
  - DTD 255
- E**
- EDMX 296
  - Element()
    - XDocument 267
    - XElement 280
  - ElementAt-Operator 130
  - ElementAtOrDefault-Operator 131
  - Elementoperatoren 125
  - Elements()
    - XDocument 267
    - XElement 280
  - ElementsAfterSelf()
    - XCDATA 287
    - XComment 276
    - XElement 280
  - ElementsBeforeSelf()
    - XCDATA 287
    - XComment 276
    - XElement 280

Empty-Operator 111  
 Encoding  
     XDeclaration 271  
 Encoding-Attribut  
     XDeclaration 271  
 Entitäten 179, 208, 293, 299  
 Entities-Datenmodell 299  
 enum 174  
 Enumerable 34, 173  
 Equals() 65, 71, 75, 78, 84, 115, 141, 148, 152  
 Erweiterungsmethoden 19  
 Except-Operator 87  
 EXEC-Anweisung 199  
 ExecuteCommand() 184, 189  
 ExecuteQuery() 199  
 Expression  
     Column-Attribut 191

## F

FailOnFirstConflict 226  
 Fill() 244, 251, 253  
 Filteroperatoren 53  
 Find()  
     DataTable 238  
 FirstAttribute  
     XElement 281  
 FirstNode  
     XDocument 268  
     XElement 281  
 First-Operator 125  
 FirstOrDefault-Operator 128  
 Func 23, 163  
 Function-Attribut 196, 201  
 Funktionen 200

## G

Generierungsoperatoren 107  
 Gespeicherte Prozeduren 196–197, 199  
 GetChanges()  
     DataTable 247  
 GetChangeSet() 184  
 GetEnumerator() 47, 177  
 GetTable() 184  
 GridView-Steuerelement 313  
 GroupBy-Operator 65  
 GroupJoin-Operator 75  
 Gruppierungsoperatoren 65

## I

ICompare 60  
 IEnumerable 19, 33–34, 46, 60, 107,  
     157, 159, 173, 179  
 IEqualityComparer 65, 71, 75, 78, 84,  
     115, 141, 145, 148, 152  
 IGrouping 33, 65  
 ILookup 145  
 InheritanceMapping-Attribut 195  
 Inline-Tabellenwertfunktion 202  
 INotifyPropertyChanged 190, 205  
 INotifyPropertyChanging 190, 205  
 InsertAllOnSubmit() 225  
 InsertOnSubmit() 223  
 InternalSubset  
     XDocumentType 273  
 Intersect-Operator 84  
 InvalidCastException 162  
 InvalidOperationException 162  
 IOrderedEnumerable 33, 60  
 IOrderedQueryable 33  
 IQueryable 33–34, 149, 157, 179  
 IsAfter()  
     XCData 288  
     XComment 276  
     XElement 281  
 IsBefore()  
     XCData 288  
     XComment 276  
     XElement 281  
 IsComposable 201  
     Function-Attribut 196  
 IsDbGenerated  
     Column-Attribut 192  
 IsDefault  
     InheritanceMapping-Attribut 195  
 IsDeleted 227  
 IsDiscriminator  
     Column-Attribut 192  
 IsForeignKey  
     Association-Attribut 193  
 IsModified 227  
 IsNamespaceDeclaration  
     XAttribute 285  
 IsolationLevel siehe Isolationsgrad  
 Isolationsgrad 219  
 IsPrimaryKey  
     Column-Attribut 192  
 IsResolved 227

- IsUnique
  - Association-Attribut 193
- IsVersion
  - Column-Attribut 192
- ItemsSource 318, 322
- ItemTemplate 322
- Iterator 159
- J**
- JOIN
  - INNER 71
  - LEFT OUTER 75
  - RIGHT OUTER 75
- Join-Operator 71
- Join-Operatoren 71
- K**
- KeepChanges 227, 229
- KeepCurrentValues 227, 229
- keySelector siehe selector
- Konvertierungsoperatoren 139
- L**
- Lambda-Ausdruck 46
- Lambda-Funktion 22
  - binäre 24
  - tenäre 24
  - unäre 24
- Verkettung 25
- LastNode
  - XDocument 268
  - XElement 281
- Last-Operator 125
- LastOrDefault-Operator 128
- LDF 208
- LET 35, 168
- LinqDataSource-Steuerelement 314
- LINQ-Operatoren 220
- LINQ-Syntax 34
- List 140, 174
- ListBox-Steuerelement 307
- ListView-Steuerelement 318
- Load() 263
  - XDocument 268
- LoadOptions 184, 220, 263
- LoadWith() 221
- Log 184, 233
- Lokale Typinferenz 25
- LongCount-Operator 90
- Lookup 141, 144
- M**
- Max-Operator 95
- MDF 208
- Member 228
- MemberChangeConflict 227
- MemberConflicts 227
- Mengenoperatoren 78
- Metadaten 208
- Methoden-Syntax 36
- Min-Operator 95
- Modelbrowser 301
- MoveNext() 159
- N**
- Name
  - Association-Attribut 193
  - Column-Attribut 192
  - Function-Attribut 196
  - Parameter-Attribut 196
  - XAttribute 285
  - XDocumentType 273
  - XObjectChange 290
- Never 217
- NextAttribute
  - XAttribute 285
- NextNode
  - XCDATA 288
  - XComment 276
  - XElement 281
- Nodes()
  - XDocument 268
  - XElement 281
- NodesAfterSelf()
  - XCDATA 288
  - XComment 276
  - XElement 281
- NodesBeforeSelf()
  - XCDATA 288
  - XComment 276
  - XElement 281
- NodeType
  - XAttribute 285
  - XCDATA 288
  - XComment 276
  - XDocument 268
  - XDocumentType 273
  - XElement 281
- None
  - LoadOptions 263
  - SaveOptions 265
- Nullable 201

**O**

O/R-Mapping 179, 293  
Object 227  
ObjectContext 296  
ObjectTrackingEnabled 184, 223  
OfType-Operator 56  
Open() 187  
OrderBy...Descending-Operator 57  
OrderBy-Operator 57  
OriginalValue 228  
OtherKey  
    Association-Attribut 193  
OUTPUT-Anweisung 196, 200  
OverwriteCurrentChanges 229  
OverwriteCurrentValues 227

**P**

Paging 182  
Parallelitätskonflikte 216  
Parameter-Attribut 196, 201  
Parent  
    XAttribute 285  
    XCData 288  
    XComment 276  
    XElement 281  
Parse() 264  
partial 206  
Performance 41  
    LINQ to Objects 42  
    LINQ to SQL 43  
Persist Security Info 186  
predicate 158, 163  
PreserveWhitespace  
    LoadOptions 263  
PreviousAttribute  
    XAttribute 285  
PreviousNode  
    XCData 288  
    XComment 276  
    XElement 281  
Primärschlüssel 238  
private 206  
Projektionsoperatoren 46  
PropertyChanged-Ereignis 205  
PropertyChanging-Ereignis 205  
protected 19  
PublicId  
    XDocumentType 273

**Q**

Quantifizierungsoperatoren 111

**R**

Range-Operator 107  
Referenztypen  
    Kompakte Initialisierung 31  
Refresh() 184, 229  
RefreshMode 229  
Rekursion 15  
Remove  
    XObjectChange 290  
Remove() 276  
    DataTable 246  
    XAttribute 285  
    XCData 288  
    XDocumentType 273  
    XElement 281  
RemoveAll()  
    XElement 281  
RemoveAttributes()  
    XElement 281  
RemoveNodes()  
    XDocument 268  
    XElement 281  
Repeat-Operator 108  
ReplaceAll()  
    XElement 281  
ReplaceAttributes()  
    XElement 281  
ReplaceNodes()  
    XDocument 268  
    XElement 282  
ReplaceWith()  
    XCData 288  
    XComment 276  
    XElement 282  
Reset() 159  
Resolve() 227–228  
resultSelector siehe selector  
RETURN 197  
Reverse-Operator 64  
Root  
    XDocument 268  
Rows 238

**S**

Save() 264  
    XDocument 268  
    XElement 282  
SaveChanges() 305  
SaveOptions 264  
sealed 19  
Selbstaufruf siehe Rekursion  
Select()  
    DataTable 238  
SelectMany-Operator 49  
Select-Operator 47  
selector 158  
SequenceEqual-Operator 152  
Sequenz 38  
Service Pack 1 für Visual Studio 2008 293  
SetBaseUri  
    LoadOptions 264  
SetField() 245  
SetLineInfo  
    LoadOptions 264  
SetValue()  
    XAttribute 285  
Sicherheit  
    XBAP 324  
Single-Operator 132  
SingleOrDefault-Operator 134  
Skalar-Funktion 201  
Skip-Operator 121  
SkipWhile-Operator 122  
SOAP 327  
Sortierungsoperatoren 56  
source 158  
sp\_who 199  
SqlCommandBuilder 248  
SqlDataAdapter 242, 244, 248  
SQLException 186  
SQL-Injection 189  
SqlMetal-Anwendung 209  
Standalone  
    XDeclaration 271  
Standalone-Atribut  
    XDeclaration 271  
Standardkonstruktor 32  
Storage  
    Column-Atribut 192  
SubmitChanges() 184, 218–219, 223, 225, 228  
Sum-Operator 92  
System.ComponentModel 190, 205

System.Data-Namensraum 237

SystemId  
    XDocumentType 273  
Systemprozeduren 199

**T**

Tabellenrelationen 295  
Tabellenwertfunktionen mit mehreren  
    Anweisungen 202  
TABLE 202  
Table 223  
Table-Atribut 190  
Take-Operator 118–119  
TCollection 158  
TextWriter 233  
ThenByDescending-Operator 60  
ThenBy-Operator 60  
this 20  
ThisKey  
    Association-Atribut 193  
timestamp 192  
ToArray 40  
ToArray-Operator 139  
ToDictionary-Operator 141  
ToDictionary 40  
ToList 40  
ToList-Operator 140  
ToLookup 40  
ToLookup-Operator 144  
ToString()-Methode 30  
Transaction 184  
TransactionScope 219–220  
Transaktionen 218  
TResult 158  
TSource 158  
Type  
    InheritanceMapping-Atribut 195  
TypedTableBaseExtensions 240  
Typisierte DataSets 249

**U**

Unboxing 25  
Union-Operator 81  
Untypisierte DataSets 240  
UpdateCheck  
    Column-Atribut 192  
using-Blöcke 27

## V

Value

  XAttribute 285

  XCData 288

  XComment 276

  XObjectChange 290

var 26

Version

  XDeclaration 271

Version-Attribut

  XDeclaration 270

Verzögerte Ausführung 39

Verzögertes Laden 222

## W

WebDienst 327

Werttypen

  Kompakte Initialisierung 31

WhenChanged 217

Where-Operator 53

Windows Forms 307

Windows-Authentifizierung 185

WPF 317

WriteTo()

  XDocument 268

  XElement 282

## X

XAML 317

XAttribute-Klasse 284

XCData-Klasse 287

XComment-Klasse 275

XDeclaration-Klasse 270

XDocument 260

XDocument-Klasse 262

XDocumentType-Klasse 272

XElement 260

XElement-Klasse 277

XML 255

  Attribut 284

  CData 287

  Dokument 262

  Dokumenttypdefinition (DTD) 272

  Element 277

  Kommentar 275

  Prolog 270

XmlReader-Klasse 272

XObjectChange 290

XObjectChangeEventArgs 290

XPATH 257

## Y

yield 38, 159

