LINQ

Language INtegrated Query

Jak działa LINQ?

- LINQ pozwala programistom na tworzenie kwerend bezpośrednio z poziomu dowolnego języka programowania obsługiwanego na platformie .NET.

Jak działa LINQ?

1. Obiekty implementujące interfejs IEnumerable<T>
2. Język XML
3. Bazy danych
4. Zbiory danych

Hello LINQ – LINQ to Objects

```
string[] greetings = {"hello world", "hello LINQ", "hello Apress" };

var items =
   from s in greetings
   where s.EndsWith("LINQ")
    select s;

foreach (var item in items)
        Console.WriteLine(item);
```

Hello LINQ – LINQ to XML

```
XElement books = XElement.Parse(
    @"<books>
        <book>
          <title>Pro LINQ: Language Integrated Query in C# 2008</title>
          <author>Joe Rattz</author>
        </book>
        <book>
          <title>Pro WF: Windows Workflow in .NET 3.0</title>
          <author>Bruce Bukovics</author>
        </book>
        <book>
          <title>Pro C# 2005 and the .NET 2.0 Platform, Third Edition</title>
          <author>Andrew Troelsen</author>
        </book>
      </books>");

var titles =
  from book in books.Elements("book")
  where (string)book.Element("author") == "Joe Rattz"
  select book.Element("title");

foreach (var title in titles)
  Console.WriteLine(title.Value);
```

Hello LINQ – LINQ to SQL

```
using nwind;

Northwind db = new Northwind(@"Data Source=.\SQLEXPRESS;Initial
    Catalog=Northwind;Integrated Security=SSPI;");

var custs =
    from c in db.Customers
    where c.City == "Rio de Janeiro"
    select c;

foreach (var cust in custs)
      Console.WriteLine("{0}", cust.CompanyName);
```

Korzyści z LINQ

- Prostota
- Wydajność
- Elastyczność

LINQ to Objects

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

var lowNums =
        from n in numbers
        where n < 5
        select n;

Console.WriteLine("Numbers < 5:");

foreach (var x in lowNums) {
        Console.WriteLine(x);
}
```

LINQ to Objects

```
string[] digits = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"

var shortDigits = digits.Where((digit, index) => digit.Length < index);

Console.WriteLine("Short digits:");

foreach (var d in shortDigits) {
            Console.WriteLine("The word {0} is shorter than its value.", d);
        }
```

Odroczone zapytania do chwili odwołania się do zmiennej

```
string[] strings = { "one", "two", null, "three" };

 Console.WriteLine("Before Where() is called.");
 IEnumerable<string> ieStrings = strings.Where(s => s.Length == 3);
 Console.WriteLine("After Where() is called.");

 foreach (string s in ieStrings)
 {
   Console.WriteLine("Processing " + s);
 }
```

Zmiana rezultatów zapytanania pomiędzy wyliczeniami zapytania

```
int[] intArray = new int[] { 1, 2, 3 };

IEnumerable<int> ints = intArray.Select(i => i);

foreach (int i in ints)
    Console.WriteLine(i);

intArray[0] = 5;

Console.WriteLine("---------");

foreach (int i in ints)
    Console.WriteLine(i);
```

Wynik:

```
1
2
3
---------
5
2
3
```

Delegaty

```
//  Create an array of ints.
int[] ints = new int[] { 1, 2, 3, 4, 5, 6 };

//  Declare our delegate.
Func<int, bool> GreaterThanTwo = i => i > 2;

//  Perform the query ... not really.  Don't forget about deferred queries!!!
IEnumerable<int> intsGreaterThanTwo = ints.Where(GreaterThanTwo);

 //  Display the results.
foreach (int i in intsGreaterThanTwo)
        Console.WriteLine(i);
```

Wynik:
3
4
5
6

Dwie notacje

- query expresion syntax
- standard dot notation

```
var oddnum  =  ( from n in nums
                 where n%2 == 1
                 select n)
                 .Reverse()
```

Operatory odroczone

- Where
- Select
- SelectMany

Select

```
string[] words = { "aPPLE", "BlUeBeRrY", "cHeRry" };

var upperLowerWords =
        from w in words
        select new {Upper = w.ToUpper(), Lower = w.ToLower()};

foreach (var ul in upperLowerWords) {
        Console.WriteLine("Uppercase: {0}, Lowercase: {1}", ul.Upper, ul.Lower);
}
```

SelectMany

```
Employee[] employees = Employee.GetEmployeesArray();

EmployeeOptionEntry[] empOptions = EmployeeOptionEntry.GetEmployeeOptionEntries();

var employeeOptions = employees
        .SelectMany(e => empOptions
                        .Where(eo => eo.id == e.id)
                        .Select(eo => new
                        {
                          id = eo.id,
                          optionsCount = eo.optionsCount
                        }));

foreach (var item in employeeOptions)
        Console.WriteLine(item);
```

Operatory odroczone

- Take
- TakeWhile
- Skip
- SkipWhile
- Concat

Sortowanie

- Sortowanie jest stabilne

```
inputSequence
.OrderBy(s => s.LastName)
.ThenBy(s=>s.firstName)....
```

OrderBy, OrderByDescending

```
public static IOrderedEnumerable<T> OrderBy<T,K> (
    this IEnumerable<T> source,
    Func<T,K> keySelector)
where
    K: IComparable<K>;


public static IOrderedEnumerable<T> OrderBy<T,K> (
    this IEnumerable<T> source,
    Func<T,K> keySelector,
    IComparer<K> comparer);


interface IComparer<T>{
    int Compare(T x, T y)
}
```

ThenBy, ThenByDescending

```
public static IOrderedEnumerable<T> OrderBy<T,K> (
    this IOrderedEnumerable<T> source,
    Func<T,K> keySelector)
where
    K: IComparable<K>;


public static IOrderedEnumerable<T> OrderBy<T,K> (
    this IOrderedEnumerable<T> source,
    Func<T,K> keySelector,
    IComparer<K> comparer);
```

Reverse

```
public static IEnumerable<T> Reverse<T>(
    this IEnumerable<T> source);
```

Join

```
Employee[] employees = Employee.GetEmployeesArray();
EmployeeOptionEntry[] empOptions =EmployeeOptionEntry.GetEmployeeOptionEntries();

var employeeOptions = employees
        .Join(
          empOptions,         //  inner sequence
          e => e.id,                          //  outerKeySelector
          o => o.id,                          //  innerKeySelector
          (e, o) => new    //  resultSelector
          {
            id = e.id,
            name = string.Format("{0} {1}", e.firstName, e.lastName),
            options = o.optionsCount
          });

foreach (var item in employeeOptions)
        Console.WriteLine(item);
```

GroupJoin

```
Employee[] employees = Employee.GetEmployeesArray();

EmployeeOptionEntry[] empOptions = EmployeeOptionEntry.GetEmployeeOptionEntries();

var employeeOptions = employees
        .GroupJoin(
          empOptions,
          e => e.id,
          o => o.id,
          (e, os) => new
          {
            id = e.id,
            name = string.Format("{0} {1}", e.firstName, e.lastName),
            options = os.Sum(o => o.optionsCount)
          });

foreach (var item in employeeOptions)
        Console.WriteLine(item);
```

GroupBy

```
public interface IGrouping<K,T>: IEnumerable<T>
{
    K key   {   get;       }
}


public static IEnumerable<IGrouping<K,T>> GroupBy<T, K> {
    this IEnumerable<T> source,
    Func<T,K> keySelector
}
```

GroupBy

```
EmployeeOptionEntry[] empOptions = EmployeeOptionEntry.GetEmployeeOptionEntries();

IEnumerable<IGrouping<int, EmployeeOptionEntry>> outerSequence =
       empOptions.GroupBy(o => o.id);

//  First enumerate through the outer sequence of IGroupings.
foreach (IGrouping<int, EmployeeOptionEntry> keyGroupSequence in outerSequence)
     {
       Console.WriteLine("Option records for employee: " + keyGroupSequence.Key);

       //  Now enumerate through the grouping's sequence of EmployeeOptionEntry
     elements.
       foreach (EmployeeOptionEntry element in keyGroupSequence)
         Console.WriteLine("id={0} : optionsCount={1} : dateAwarded={2:d}",
           element.id, element.optionsCount, element.dateAwarded);
     }
```

Operatory odroczone – działające na multizbiorach

- Distinct
- Union
- Intersect
- Except

Operatory odroczone

- Cast
- OfType
- DefaultIfEmpty
- Range
- Repeat
- Empty

Operatory nieodroczone – operatory konwertujące

- ToArray
- ToList
- ToDictionary
- ToLookup

Operatory nieodroczone – operatory równościowe

- SequenceEqual
- FirstOrDefault
- Last
- LastOrDefault
- Single
- SingleOrDefault
- ElementAt
- ElementAtOrDefault

Operatory nieodroczone - quantifiers

- Any

```
string[] presidents = {
    "Adams", "Arthur", "Buchanan", "Bush", "Carter", "Cleveland",
    "Clinton", "Coolidge", "Eisenhower", "Fillmore", "Ford",  "Garfield",
    "Grant", "Harding", "Harrison", "Hayes", "Hoover", "Jackson",
    "Jefferson", "Johnson", "Kennedy", "Lincoln", "Madison", "McKinley",
    "Monroe", "Nixon", "Pierce", "Polk", "Reagan", "Roosevelt", "Taft",
    "Taylor", "Truman", "Tyler", "Van Buren", "Washington", "Wilson"};

bool any = presidents.Any();
Console.WriteLine(any);
```

- All
- Contains

Operatory nieodroczone - agregaty

- Count
- LongCount
- Sum

```
public static Numeric Sum(
    this IEnumerable<Numeric> source)

public static Numeric Sum<T>(
    this IEnumerable<T> source,
    Func<T, Numeric> selector)
```

- Min
- Max
- Average

Aggregate

```
//  First I need an array of integers from 1 to N where
//  N is the number I want the factorial for.  In this case,
 //  N will be 5.
      int N = 5;
      IEnumerable<int> intSequence = Enumerable.Range(1, N);

 //  I will just ouput the sequence so all can see it.
      foreach (int item in intSequence)
        Console.WriteLine(item);

//  Now calculate the factorial and display it.
//  av == aggregated value, e == element
      int agg = intSequence.Aggregate((av, e) => av * e);
      Console.WriteLine("{0}! = {1}", N, agg);
```

Aggregate

```
//  Create a sequence of ints from 1 to 10.
      IEnumerable<int> intSequence = Enumerable.Range(1, 10);

//  I'll just ouput the sequence so all can see it.
      foreach (int item in intSequence)
        Console.WriteLine(item);
      Console.WriteLine("--");

//  Now calculate the sum and display it.
      int sum = intSequence.Aggregate(0, (s, i) => s + i);
      Console.WriteLine(sum);
```

LINQ to XML

```
<BookParticipants>
  <BookParticipant type="Author">
      <FirstName>Joe</FirstName>
      <LastName>Rattz</LastName>
  </BookParticipant>
  <BookParticipant type="Editor">
      <FirstName>Joe</FirstName>
      <LastName>Rattz</LastName>
  </BookParticipant>
</BookParticipants>
```

XML Document Model API

```
//  I'll declare some variables I will reuse.
    XmlElement xmlBookParticipant;
    XmlAttribute xmlParticipantType;
    XmlElement xmlFirstName;
    XmlElement xmlLastName;

    //  First, I must build an XML document.
    XmlDocument xmlDoc = new XmlDocument();

    //  I'll create the root element and add it to the document.
    XmlElement xmlBookParticipants =
       xmlDoc.CreateElement("BookParticipants");
    xmlDoc.AppendChild(xmlBookParticipants);

    //  I'll create a participant and add it to the book participants list.
    xmlBookParticipant = xmlDoc.CreateElement("BookParticipant");

    xmlParticipantType = xmlDoc.CreateAttribute("type");
    xmlParticipantType.InnerText = "Author";
    xmlBookParticipant.Attributes.Append(xmlParticipantType);

    xmlFirstName = xmlDoc.CreateElement("FirstName");
    xmlFirstName.InnerText = "Joe";
    xmlBookParticipant.AppendChild(xmlFirstName);

    xmlLastName = xmlDoc.CreateElement("LastName");
    xmlLastName.InnerText = "Rattz";
    xmlBookParticipant.AppendChild(xmlLastName);

    xmlBookParticipants.AppendChild(xmlBookParticipant);

//  I'll create another participant and add it to the book participants list.
xmlBookParticipant = xmlDoc.CreateElement("BookParticipant");

xmlParticipantType = xmlDoc.CreateAttribute("type");
xmlParticipantType.InnerText = "Editor";
xmlBookParticipant.Attributes.Append(xmlParticipantType);

xmlFirstName = xmlDoc.CreateElement("FirstName");
xmlFirstName.InnerText = "Ewan";
xmlBookParticipant.AppendChild(xmlFirstName);

xmlLastName = xmlDoc.CreateElement("LastName");
xmlLastName.InnerText = "Buckingham";
xmlBookParticipant.AppendChild(xmlLastName);

xmlBookParticipants.AppendChild(xmlBookParticipant);

//  Now, I'll search for authors and display their first and last name.
XmlNodeList authorsList =

   xmlDoc.SelectNodes("BookParticipants/BookParticipant[@type=\"Author
   \"]");

foreach (XmlNode node in authorsList)
{
  XmlNode firstName = node.SelectSingleNode("FirstName");
  XmlNode lastName = node.SelectSingleNode("LastName");
  Console.WriteLine("{0} {1}", firstName, lastName);
}
```

LINQ to XML

```
XMLOBJECT o =
   new XMLOBJECT(OBJECTNAME,
       XMLOBJECT1,
       XMLOBJECT2,
       ...
       XMLOBJECTN);
```

LINQ to XML

```
XDocument xDocument = new XDocument(
      new XElement("BookParticipants",
        new XElement("BookParticipant",
          new XAttribute("type", "Author"),
          new XElement("FirstName", "Joe"),
          new XElement("LastName", "Rattz")),
        new XElement("BookParticipant",
          new XAttribute("type", "Editor"),
          new XElement("FirstName", "Ewan"),
          new XElement("LastName", "Buckingham"))));
```

Konstruktory

- XElement.XElement(XName name, object content);
- XElement.XElement(XName name, params object content[]);

Rożne zachowania obiektów przy tworzeniu dziecka w obiekcie rodzicu

- string
- XText
- XCData
- XElement
- XAttribute
- XProcessingInstruction
- XComment
- IEnumerable
- null
- inny typ

Konstruktory

```
BokParticipant[] bookParticipants = new[] {
      new BookParticipant {FirstName = "Joe", LastName = "Rattz",
                  ParticipantType = ParticipantTypes.Author},
      new BookParticipant {FirstName = "Ewan", LastName = "Buckingham",
                  ParticipantType = ParticipantTypes.Editor}
    };

    XElement xBookParticipants =
      new XElement("BookParticipants",
            bookParticipants.Select(p =>
              new XElement("BookParticipant",
                new XAttribute("type", p.ParticipantType),
                new XElement("FirstName", p.FirstName),
                new XElement("LastName", p.LastName))));
```

Metody i właściwości

- NextNode
- PreviousNode
- Document
- Parent
- Nodes
- Elements
- Element
- Ancestors
- AncestorsAndSelf
- Descensants
- DescendantsAndSelf
- NodesAfterSelf
- ElementsAfterSelf
- NodesBeforeSelf
- ElementsBeforeSelf

Modyfikacja elementów

- Dodawanie
  - AddFirst
  - AddBeforeSelf
  - AddAfrerSelf
- Usuwanie
  - Remove
  - RemoveAll
- Zmiana
  - Value
  - zmiana własności elementu dokument (Name, SystemId, PublicId)
  - replaceAll
  - SetElementValue

Zdarzenia

- XMLEvents
  - XObject.Changing
  - XObject.Changed

Inne właściwości - zapytania

```
var biddata = from b in bids.Descendants("bid_tuple")
              where ((double)b.Element("bid")) > 50
              join u in users.Descendants("user_tuple")
              on ((string)b.Element("userid")) equals
                ((string)u.Element("userid"))
              join i in items.Descendants("item_tuple")
              on ((string)b.Element("itemno")) equals
                ((string)i.Element("itemno"))
              select new
              {
                Item = ((string)b.Element("itemno")),
                Description = ((string)i.Element("description")),
                User = ((string)u.Element("name")),
                Date = ((string)b.Element("bid_date")),
                Price = ((double)b.Element("bid"))
              };
```

Inne właściwości - transformacje

- Transformacje XSLT
- Transformacje używające konstrukcji funkcyjnych

```
XDocument xTransDocument = new XDocument(
new XElement("MediaParticipants",
 new XAttribute("type", "book"),
  xDocument.Element("BookParticipants")
   .Elements("BookParticipant")
   .Select(e => new XElement("Participant",
    new XAttribute("Role", (string)e.Attribute("type")),
    new XAttribute("Name", (string)e.Element("FirstName") + " " +
     (string)e.Element("LastName")))));
```

Inne właściwości

- Validacja XML Schema
- Zapytania XPath

LINQ to SQL

- Wstawianie
- Zapytania
- Modyfikacja
- Usuwanie

Wstawianie

```
// 1. Create the DataContext.
    Northwind db = new Northwind(@"Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=SSPI;");

// 2. Instantiate an entity object.
    Customer cust =
      new Customer
      {
        CustomerID = "LAWN",
        CompanyName = "Lawn Wranglers",
        ContactName = "Mr. Abe Henry",
        ContactTitle = "Owner",
        Address = "1017 Maple Leaf Way",
        City = "Ft. Worth",
        Region = "TX",
        PostalCode = "76104",
        Country = "USA",
        Phone = "(800) MOW-LAWN",
        Fax = "(800) MOW-LAWO"
      };

// 3. Add the entity object to the Customers table.
    db.Customers.InsertOnSubmit(cust);

// 4. Call the SubmitChanges method.
    db.SubmitChanges();
```

Wstawianie

```
Northwind db = new Northwind(@"Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=SSPI;");

    Customer cust =
      new Customer
      {
        CustomerID = "LAWN",
        CompanyName = "Lawn Wranglers",
        ContactName = "Mr. Abe Henry",
        …
        Fax = "(800) MOW-LAWO",
        Orders = {
          new Order {
            CustomerID = "LAWN",
            EmployeeID = 4,
            OrderDate = DateTime.Now,
            ...
            ShipRegion = "TX",
            ShipPostalCode = "76104",
            ShipCountry = "USA"
          }
        }
      };

    db.Customers.InsertOnSubmit(cust);
    db.SubmitChanges();
```

Zapytania

- Zapytania zwracają sekwencje IQueryable<T>
- Zapytania są tłumaczone do SQL-a
- Zapytania są wykonywane na bazie danych

Zapytania

```
Northwind db = new Northwind(@"Data Source=.\SQLEXPRESS;Initial
Catalog=Northwind;Integrated Security=SSPI;");

IQueryable<Customer> custs = from c in db.Customers
                             where c.Country == "UK" &&
                               c.City == "London"
                             orderby c.CustomerID
                             select c;

foreach (Customer cust in custs)
{
  Console.WriteLine("{0} - {1}", cust.CompanyName, cust.ContactName);
  foreach (Order order in cust.Orders)
  {
    Console.WriteLine("    {0} {1}", order.OrderID, order.OrderDate);
  }
}
```

Zapytania

```
var entities = from s in db.Suppliers
               join c in db.Customers on s.City equals c.City
               select new
               {
                 SupplierName = s.CompanyName,
                 CustomerName = c.CompanyName,
                 City = c.City
               };

foreach (var e in entities)
    {
      Console.WriteLine("{0}: {1} - {2}", e.City, e.SupplierName, e.CustomerName);
    }
```

Modyfikacja

```
Order order = (from o in db.Orders
               where o.EmployeeID == 5
               orderby o.OrderDate descending
               select o).First<Order>();

 Console.WriteLine("Before changing the employee.");
 Console.WriteLine("OrderID = {0} : OrderDate = {1} : EmployeeID = {2}",
       order.OrderID, order.OrderDate, order.Employee.EmployeeID);

Employee emp = (from e in db.Employees
                where e.EmployeeID == 9
                select e).Single<Employee>();

//  Now I will assign the new employee to the order.
order.Employee = emp;

db.SubmitChanges();
```

Modyfikacja

```
Northwind db = new Northwind(@"Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=SSPI;");

Order order = (from o in db.Orders
               where o.EmployeeID == 5
               orderby o.OrderDate descending
               select o).First<Order>();

Console.WriteLine("Before changing the employee.");
Console.WriteLine("OrderID = {0} : OrderDate = {1} : EmployeeID = {2}",
  order.OrderID, order.OrderDate, order.Employee.EmployeeID);

Employee emp = (from e in db.Employees
                where e.EmployeeID == 9
                select e).Single<Employee>();

//  Remove the order from the original employee's Orders.
origEmployee.Orders.Remove(order);

//  Now add it to the new employee's orders.
emp.Orders.Add(order);

db.SubmitChanges();
```

Usuwanie

```
Northwind db = new Northwind(@"Data Source=.\SQLEXPRESS;Initial
    Catalog=Northwind;Integrated Security=SSPI;");

    //  Retrieve a customer to delete.
    Customer customer = (from c in db.Customers
                         where c.CompanyName == "Alfreds Futterkiste"
                         select c).Single<Customer>();

    db.OrderDetails.DeleteAllOnSubmit(
      customer.Orders.SelectMany(o => o.OrderDetails));
    db.Orders.DeleteAllOnSubmit(customer.Orders);
    db.Customers.DeleteOnSubmit(customer);

    db.SubmitChanges();
```

Nadpisywanie wyrażeń modyfikujących bazę danych

partial void Insert[EntityClassName] (T instance)

partial void Update[EntityClassName] (T instance)

partial void Delete[EntityClassName] (T instance)

Nadpisywanie wyrażeń modyfikujących bazę danych

```
partial void InsertShipper(Shipper instance)
  {
    Console.WriteLine("Insert override method was called for shipper {0}.",
      instance.CompanyName);
    //this.ExecuteDynamicInsert(instance);
  }

  partial void UpdateShipper(Shipper instance)
  {
    Console.WriteLine("Update override method was called for shipper {0}.",
      instance.CompanyName);
    //this.ExecuteDynamicUpdate(instance);
  }

  partial void DeleteShipper(Shipper instance)
  {
    Console.WriteLine("Delete override method was called for shipper {0}.",
      instance.CompanyName);
    //this.ExecuteDynamicDelete(instance);
  }
```