Week 6

Introduction to LINQ

The Relational Model

- Relations provide the sole means for structuring data in the relational model.
- A relation is a homogeneous ***set*** of records, each record itself consisting of a heterogeneous set of uniquely named attributes.
- Relations can be of the following kinds:
  - Base Relations are those which are stored directly
  - Derived Relations (also known as Views)

Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Relational Algebra

- **Restrict** is a unary operation which allows the selection of a subset of the records in a relation according to some desired criteria
- **Project** is a unary operation which creates a new relation corresponding to the old relation with various attributes removed from the records
- **Product** is a binary operation corresponding to the cartesian product of mathematics
- **Union** is a binary operation which creates a relation consisting of all records in either argument relation
- **Intersection** is a binary operation which creates a relation consisting of all records in both argument relations
- **Difference** is a binary operation which creates a relation consisting of all records in the first but not the second argument relation
- **Join** is a binary operation which constructs all possible records that result from matching identical attributes of the records of the argument relations
- **Divide** is a ternary operation which returns all records of the first argument which occur in the second argument associated with each record of the third argument
- One significant benefit of this manipulation language (aside from its simplicity) is that it has the property of ***closure*** — that all operands and results are of the same kind (relations) — hence the operations can be nested in arbitrary ways.

SQL (Structured Query Language)

```
SELECT WORKDEPT, MAX(SALARY)
FROM DSN8A10.EMP Q
GROUP BY WORKDEPT
HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                      FROM DSN8A10.EMP
                      WHERE NOT WORKDEPT = Q.WORKDEPT);
```

Object-Relational Mapping Frameworks

http://en.wikipedia.org/wiki/Object-relational_mapping

Object-relational mapping (**ORM**, **O/RM**, and **O/R mapping**) frameworks convert data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.

Examples:

Java	Hibernate	http://www.hibernate.org
C++	ODB	http://www.codesynthesis.com/products/odb/
Python	SQLAlchemy	http://www.sqlalchemy.org
C#	Entity Framework	http://www.asp.net/entity-framework
PHP	Doctrine	http://www.doctrine-project.org/
Javascript	Bookshelf.js	http://bookshelfjs.org
Ruby	ActiveRecord	http://ar.rubyonrails.org

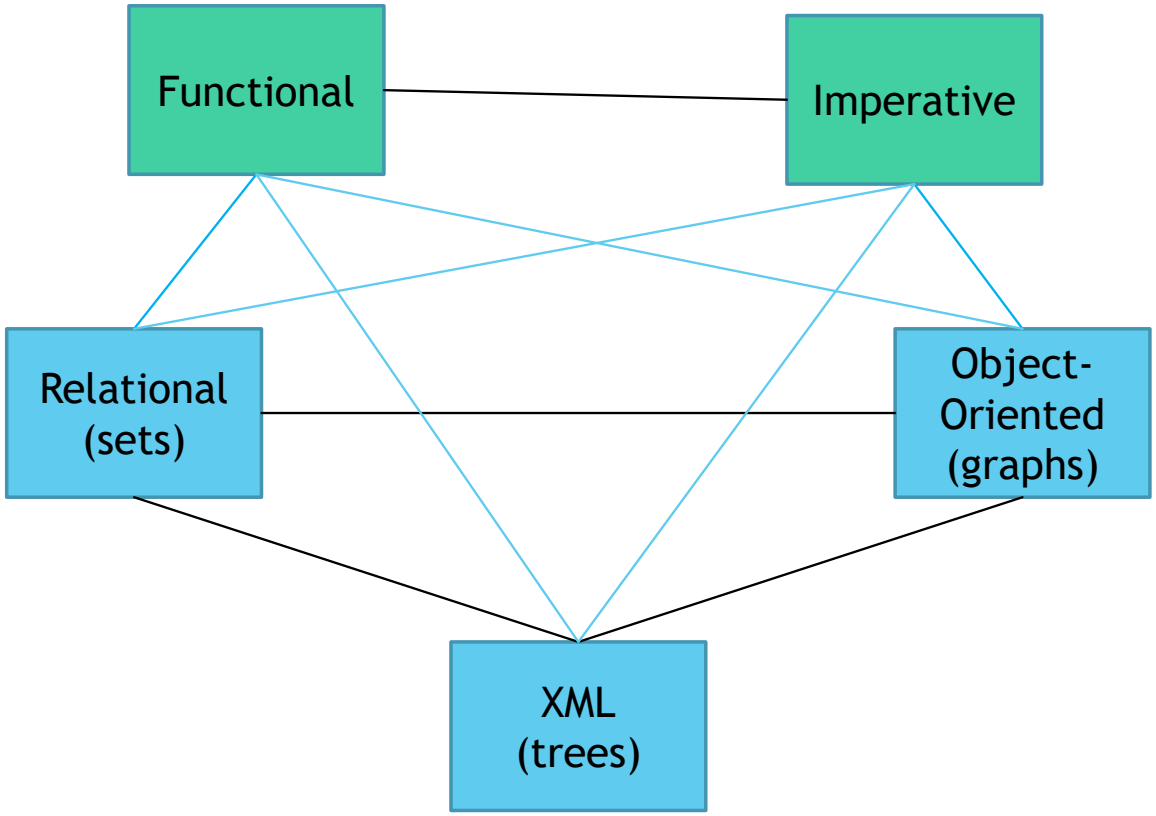
Twenty years of object-relational mapping:

A survey on patterns, solutions, and their implications on application design

Object-Relational Impedance Mismatch

- http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch
- The **object-relational impedance mismatch** is a set of conceptual and technical difficulties that are often encountered when a relational database management system (RDBMS) is being used by a program written in an object-oriented programming language or style; particularly when objects or class definitions are mapped in a straightforward way to database tables or relational schema.
- Do relations represent objects or relationships between objects?
  - In OO, relationships between objects are represented using pointers
  - In relational model, relationships are represented either within relations or as foreign keys.
  - OO = ***graph*** of objects, relational = ***set*** of relations
- Not always a one-to-one relationship between objects and rows.

Impedance Mismatches?



LINQ (Language Integrated Query)

- Native syntax for expressing queries in C# and VB.NET
- All data is object-oriented and strongly typed
- Bridges to Relational and XML data providing a universal query language for all types of data.

LINQ References

- https://msdn.microsoft.com/en-us/library/bb397926.aspx

Data Sources

- Anything that implements `IEnumerable` interface
- Including:
  - Results from other LINQ queries
  - Standard collection classes (System.Collection.List, Array, …)
  - Specialized LINQ providers:
    - LINQ to SQL
    - LINQ to XML
    - LINQ to ADO.NET

LINQ Example

```
using System.Linq;

class Person
{
    public string name { get; set};
    public int age {get set};
}

void main(string[] args)
{
    IEnumerable<Person> people =
        new Person[] { new Person {"Paul", 23},
                       new Person {"Jill", 16}};

    IEnumerable<string> query =
        from p in people
        where p.age > 21
        select p.name.ToUpper();

    foreach (string name in query)
        Console.WriteLine(name);
}
```

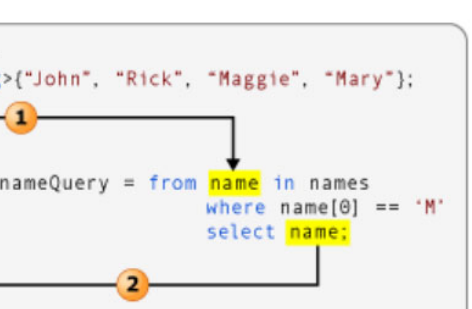
1. Obtain the data source
2. Create the query
3. Execute the query

LINQ with Type Inference

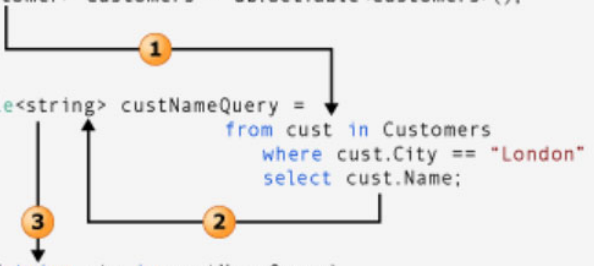
```
using System.Linq;

class Person
{
    public string name { get; set};
    public int age {get set};
}

void main(string[] args)
{
    var people =
        new Person[] { new Person {"Paul", 23},
                       new Person {"Jill", 16}};

    var query = from p in people
                where p.age > 21
                select p.name.ToUpper();

    foreach (var name in query)
        Console.WriteLine(name);
}
```

LINQ Types

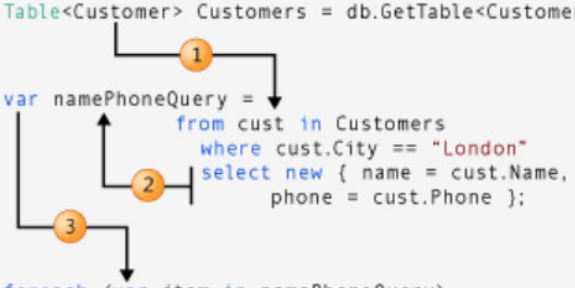
```
List<string> names =
      new List<string>{"John", "Rick", "Maggie", "Mary"};

IEnumerable<string> nameQuery = from name in names
                                where name[0] == 'M'
                                select name;

foreach (string str in nameQuery)
{
      Console.WriteLine(str);
}
```



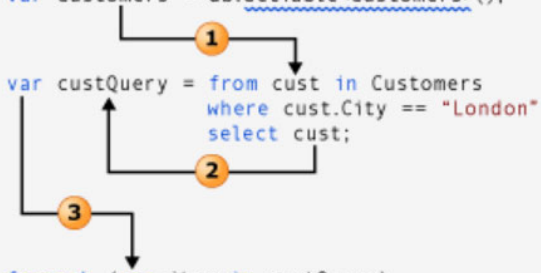

```
Table<Customer> Customers = db.GetTable<Customers>();

IQueryable<string> custNameQuery =
                                from cust in Customers
                                   where cust.City == "London"
                                   select cust.Name;

foreach (string str in custNameQuery)
{
      Console.WriteLine(str);
}
```




```
Table<Customer> Customers = db.GetTable<Customers>();

var namePhoneQuery =
                   from cust in Customers
                     where cust.City == "London"
                   select new { name = cust.Name,
                                phone = cust.Phone };

foreach (var item in namePhoneQuery)
{
      Console.WriteLine(item);
}
```




```
var Customers = db.GetTable<Customers>();

var custQuery = from cust in Customers
                where cust.City == "London"
                select cust;

foreach (var item in custQuery)
{
      Console.WriteLine(item);
}
```



LINQ Projection

```
var query = from cust in Customer
            select new {Name = cust.Name, City = cust.City};
```

```
var studentsToXML = new XElement("Root",
    from student in students
    let x = String.Format("{0},{1},{2},{3}", student.Scores[0],
            student.Scores[1], student.Scores[2], student.Scores[3])
    select new XElement("student",
               new XElement("First", student.First),
               new XElement("Last", student.Last),
               new XElement("Scores", x)
           ) // end "student"
        ); // end "Root"
```

```
IEnumerable<string> query =
    from rad in radii
    select String.Format("Area = {0}", (rad * rad) * 3.14);
```

LINQ Filtering

```
var queryLondonCustomers = from cust in customers
                           where cust.City == "London"
                           select cust;
```

```
where cust.City=="London" && cust.Name == "Devon"
```

```
where cust.City == "London" || cust.City == "Paris"
```

LINQ Ordering

```
var queryLondonCustomers3 =
    from cust in customers
    where cust.City == "London"
    orderby cust.Name ascending
    select cust;
```

LINQ Grouping

```
// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>
  var queryCustomersByCity =
      from cust in customers
      group cust by cust.City;

  // customerGroup is an IGrouping<string, Customer>
  foreach (var customerGroup in queryCustomersByCity)
  {
      Console.WriteLine(customerGroup.Key);
      foreach (Customer customer in customerGroup)
      {
          Console.WriteLine("    {0}", customer.Name);
      }
  }
```

Produces a sequence of groups each consisting of a Key and a list of related records

```
// custQuery is an IEnumerable<IGrouping<string, Customer>>
var custQuery =
    from cust in customers
    group cust by cust.City into custGroup
    where custGroup.Count() > 2
    orderby custGroup.Key
    select custGroup;
```

Give the group a name so its attributes can be conveniently accessed.

LINQ Joining

Join condition must be of the form: … `equals` …

```
var innerJoinQuery =
    from cust in customers
    join dist in distributors on cust.City equals dist.City
    select new { CustomerName = cust.Name, DistributorName = dist.Name };
```

Navigating Relationships without Join

- If the *database* has foreign key constraints then *navigation properties* will be created automatically.
  - Handles both 1-to-1 and 1-to-many relationships.

```
from p in ctx.Persons
where p.ID == personId
join bornIn in ctx.Cities
on p.BornIn equals bornIn.CityID
join livesIn in ctx.Cities
on p.LivesIn equals livesIn.CityID
join s in ctx.Sexes
on p.SexID equals s.ID
select new PersonInfo
{
    Name = p.FirstName + " " + p.LastName,
    BornIn = bornIn.Name,
    LivesIn = livesIn.Name,
    Gender = s.Name,
    CarsOwnedCount = ctx.Cars.Where(c =&gt; c.OwnerID == p.ID).Count()
}
```

```
from p in ctx.Persons
where p.ID == personId
select new PersonInfo
{
    Name = p.FirstName + " " + p.LastName,
    BornIn = p.BornInCity.Name,
    LivesIn = p.LivesInCity.Name,
    Gender = p.Sex.Name,
    CarsOwnedCount = p.Cars.Count(),
}
```

RayTracing Example

- https://github.com/icsharpcode/ILSpy/blob/master/ICSharpCode.Decompiler.Tests/TestCases/Correctness/LINQRaytracer.cs

LINQ: Query Syntax vs Method Syntax

```
//Query syntax:
IEnumerable<int> numQuery1 =
    from num in numbers
    where num % 2 == 0
    orderby num
    select num;
```

```
//Method syntax:
IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 == 0).OrderBy(n => n);
```

```
// Using query expression syntax.
var query = from word in words
            group word.ToUpper() by word.Length into gr
            orderby gr.Key
            select new { Length = gr.Key, Words = gr };
```

```
// Using method-based query syntax.
var query2 = words.
    GroupBy(w => w.Length, w => w.ToUpper()).
    Select(g => new { Length = g.Key, Words = g }).
    OrderBy(o => o.Length);
```

LINQ: Custom Providers

- Simplest scenario: just implement **`IEnumerable`** interface and let LINQ to Object take care of all the other LINQ operations.
- More complex: implement **`IQueryable`** interface and provide custom implementations of various query operations (select, from, where, …).
  - a complex **IQueryable** provider, such as the LINQ to SQL provider, might translate complete LINQ queries to an expressive query language, such as SQL.
  - https://docs.microsoft.com/en-au/archive/blogs/mattwar/linq-building-an-iqueryable-provider-part-i

```
namespace System.Linq
{
    public interface IQueryable<out T> : IEnumerable<T>, IQueryable, IEnumerable
    {
    }
}
```

```
namespace System.Linq
{
    public interface IQueryable : IEnumerable
    {
        Type ElementType { get; }
        Expression Expression { get; }
        IQueryProvider Provider { get; }
    }
}
```

```
namespace System.Linq
{
    public interface IQueryProvider
    {
        IQueryable CreateQuery(Expression expression);
        IQueryable<TElement> CreateQuery<TElement>(Expression expression);
        object Execute(Expression expression);
        TResult Execute<TResult>(Expression expression);
    }
}
```

Parallel LINQ (PLINQ)

- https://msdn.microsoft.com/en-us/library/dd460688(v=vs.110).aspx
- Parallel LINQ (PLINQ) is a parallel implementation of LINQ to Objects.
- PLINQ implements the full set of LINQ standard query operators as extension methods for the System.Linq namespace and has additional operators for parallel operations.

```
// Result sequence might be out of order.
var parallelQuery = from num in source.AsParallel()
                    where num % 10 == 0
                    select num;

// Process result sequence in parallel
parallelQuery.ForAll((e) => DoSomething(e));

// Or use foreach to merge results first.
foreach (var n in parallelQuery) {
   Console.WriteLine(n);
}
```

Java 8 Streams

```
List<Integer> transactionsIds =
    transactions.stream()
                .filter(t -> t.getType() == Transaction.GROCERY)
                .sorted(comparing(Transaction::getValue).reversed())
                .map(Transaction::getId)
                .collect(toList());
```

http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html

https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html