[목 차]

LINQ(Language Integrated Query)

통합된 데이터 질의 기능으로 실제 그 쿼리 문법은 SQL 쿼리의 SELECT 구문과 유사하다.

데이터를 읽고, 필터링하고, 정렬하는 기능을 쉽게 처리할 수 있음

From : 어떤 데이터 집합에서 찾을 것인가?

Where : 어떤 값의 데이터를 찾을 것인가?

Select : 어떤 항목을 추출할 것인가?

1. FROM, WHERE, ORDERBY, SELECT의 이해

```
static void Main(string[] args)
        {
            int[] numbers = { 9, 2, 6, 4, 5, 3, 7, 8, 1, 10 };
            var result = from n in numbers
                         where n % 2 == 0
                         orderby n   desceding
                         select n;
            foreach (int n in result)
                Console.WriteLine("짝수 : {0}", n);
    }
```

[from]

from<범위 변수> in <데이터 원본> 의 형식으로 사용

[where]

필터 역할을 하는 연산자

from절이 데이터 원본으로 뽑아낸 범위 변수가 가져야 하는 조건을 where 연산자에게 매개 변수로 입력하면 LINQ는 해당 조건에 부합하는 데이터만을 걸러냄

[orderby]

데이터 정렬을 수행하는 연산자

기본적으로 오름차순(ascending)으로 정렬

[select]

최종 결과를 추출하는 연산자

1. 아래 데이터에서 Height가 175미만인 데이터만 추려내려면

Step1) 일반 코드

```
class Profile : IComparable<Profile>      {
        public string Name { get; set; }
        public int Height { get; set; }
        public int CompareTo(Profile other)          {
            return Name.CompareTo(other.Name);
        }
    }
    class MainApp      {
        static void Main(string[] args)          {
            Profile[] arrProfile = {
                new Profile(){Name="정우성", Height=186},
                new Profile(){Name="김태희", Height=158},
                new Profile(){Name="고현정", Height=172},
                new Profile(){Name="이문세", Height=178},
                new Profile(){Name="하동훈", Height=171}
            };
            //필터링
            List<Profile> profiles = new List<Profile>();
            foreach (Profile profile in arrProfile)                {
                if (profile.Height < 175)
                    profiles.Add(profile);
            }
            //정렬
            profiles.Sort();
            //결과출력
            foreach (var profile in arrProfile)
                Console.WriteLine("{0}, {1}",profile.Name, profile.Height);
        }
    }
```

Step2) 일부 코드 Ramda 적용

```
class Profile       // 수정
    {
        public string Name { get; set; }
        public int Height { get; set; }

    }

    class MainApp
    {
        static void Main(string[] args)
        {
            Profile[] arrProfile = {
                new Profile(){Name="정우성", Height=186},
                new Profile(){Name="김태희", Height=158},
                new Profile(){Name="고현정", Height=172},
                new Profile(){Name="이문세", Height=178},
                new Profile(){Name="하동훈", Height=171}
            };

            //필터링
            List<Profile> profiles = new List<Profile>();
            foreach (Profile profile in arrProfile)
            {
                if (profile.Height < 175)
                    profiles.Add(profile);
            }

            //정렬 : Ramda 적용
            profiles.Sort(
                (profile1, profile2) =>
                {
                    return profile1.Height - profile2.Height;
                }
                );

            //결과출력
            foreach (var profile in arrProfile)
                Console.WriteLine("{0}, {1}",profile.Name, profile.Height);

        }
    }
```

Step3) LINQ 적용

```
class Profile      {
          public string Name { get; set; }
          public int Height { get; set; }
     }
class MainApp      {
          static void Main(string[] args)           {
               Profile[] arrProfile = {
                    new Profile(){Name="정우성", Height=186},
                    new Profile(){Name="김태희", Height=158},
                    new Profile(){Name="고현정", Height=172},
                    new Profile(){Name="이문세", Height=178},
                    new Profile(){Name="하동훈", Height=171}
               };

               //필터링 & 정렬
               // var의 형식은 IEnumerable<Profile>
               var profiles = from profile in arrProfile
                              where profile.Height < 175
                              orderby profile.Height
                              select profile;
               //결과출력
               foreach (var profile in profiles)
                    Console.WriteLine("{0}, {1}",profile.Name, profile.Height);
          }
     }
```

```
//필터링 & 정렬
var의 형식은 IEnumerable<string>
var profiles = from profile in arrProfile
               where profile.Height < 175
               orderby profile.Height
               select profile.Name;
```

```
//필터링 & 정렬
var의 형식은 IEnumerable<무명형식>
var profiles = from profile in arrProfile
               where profile.Height < 175
               orderby profile.Height
                select new{ Name = profile.Name,
                                    InchHeight = profile.Height * 0.393 };
//결과출력
               foreach (var profile in profiles)
                    Console.WriteLine("{0}, {1}",profile.Name, profile.InchHeight);
```

2. 여러 개의 데이터 원본에 질의하기

```
class Member
    {
        public string Name { get; set; }
        public int[] Score { get; set; }
    }

    class MainApp
    {
        static void Main(string[] args)
        {
            Member[] arrClass =
            {
                new Member(){Name="연두반", Score=new int[]{99, 59, 70, 24}},
                new Member(){Name="분홍반", Score=new int[]{60, 45, 87, 72}},
                new Member(){Name="파랑반", Score=new int[]{92, 30, 85, 94}},
                new Member(){Name="노랑반", Score=new int[]{90, 88, 0, 17}}
            };

            //60점보다 작은 학생들의 점수 추출
            List<int> scores = new List<int>();
            foreach (Member mem in arrClass)
            {
                foreach(int score in mem.Score)
                if (score < 60)
                        scores.Add(score);
            }

            var classes = from c in arrClass
                          from s in c.Score
                          where s < 60
                          orderby s
                          select new { c.Name, Lowest = s };

            foreach (var c in classes)
                Console.WriteLine("낙제 : {0} ({1})", c.Name, c.Lowest);
        }
    }
```

3. GROUP BY로 데이터 분류하기

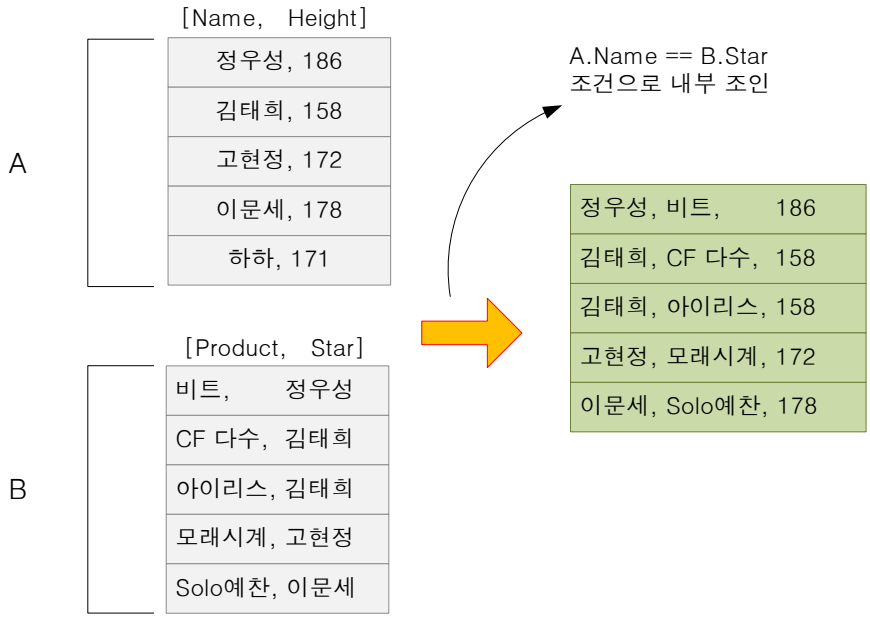
```
class Profile
    {
        public string Name { get; set; }
        public int Height { get; set; }
    }

    class MainApp
    {
        static void Main(string[] args)
        {
            Profile[] arrProfile =
            {
                new Profile(){Name="정우성", Height=186},
                new Profile(){Name="김태희", Height=158},
                new Profile(){Name="고현정", Height=172},
                new Profile(){Name="이문세", Height=178},
                new Profile(){Name="하하", Height=171}
            };

            var listProfile = from profile in arrProfile
                              orderby profile.Height
                              group profile by profile.Height < 175 into g
                              select new { GroupKey = g.Key, Profiles = g };

            foreach (var Group in listProfile)
            {
                Console.WriteLine("- 175cm 미만? : {0}", Group.GroupKey);

                foreach (var profile in Group.Profiles)
                {
                    Console.WriteLine("      {0}, {1}", profile.Name, profile.Height);
                }
            }
        }
    }
```

- **예제 확인(Min, Max, Average)**

```
class Profile
    {
        public string Name { get; set; }
        public int Height { get; set; }
    }

    class MainApp
    {
        static void Main(string[] args)
        {
            Profile[] arrProfile =
            {
                new Profile(){Name="정우성", Height=186},
                new Profile(){Name="김태희", Height=158},
                new Profile(){Name="고현정", Height=172},
                new Profile(){Name="이문세", Height=178},
                new Profile(){Name="하하", Height=171}
            };

            var heightStat = from profile in arrProfile
                             group profile by profile.Height < 175 into g
                             select new
                             {
                                 Group = g.Key == true ? "175미만" : "175이상",
                                 Count = g.Count(),
                                 Max = g.Max(profile => profile.Height),
                                 Min = g.Min(profile => profile.Height),
                                 Average = g.Average(profile => profile.Height)
                             };

            foreach (var stat in heightStat)
            {
                Console.Write("{0} - Count:{1}, Max:{2}, ",
                    stat.Group, stat.Count, stat.Max);
                Console.WriteLine("Min:{0}, Average:{1}",
                    stat.Min, stat.Average);
            }
        }
}
```

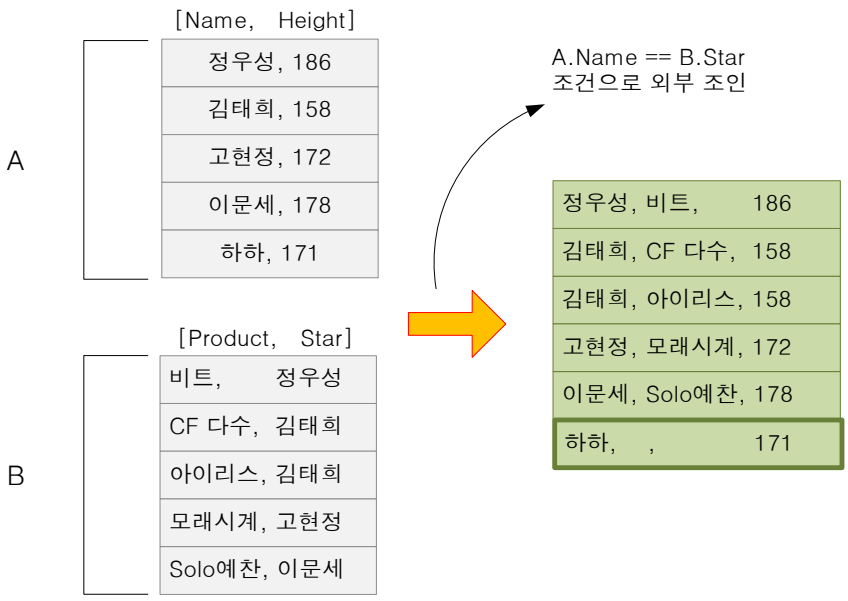
4. Join

JOIN(내부 조인, 두 데이터 원본 사이에서 일치하는 데이터들만 연결한 후 반환)



JOIN(외부조인, 조인 결과에 기준이 되는 데이터 원본은 모두 포함)

-



```
class Profile
    {
        public string Name { get; set; }
        public int Height { get; set; }
    }

    class Product
    {
        public string Title { get; set; }
        public string Star { get; set; }
    }

    class MainApp
    {
        static void Main(string[] args)
        {
            Profile[] arrProfile =
            {
                new Profile(){Name="정우성", Height=186},
                new Profile(){Name="김태희", Height=158},
                new Profile(){Name="고현정", Height=172},
                new Profile(){Name="이문세", Height=178},
                new Profile(){Name="하하", Height=171}
            };

            Product[] arrProduct =
            {
                new Product(){Title="비트",          Star="정우성"},
                new Product(){Title="CF 다수",       Star="김태희"},
                new Product(){Title="아이리스",      Star="김태희"},
                new Product(){Title="모래시계",      Star="고현정"},
                new Product(){Title="Solo 예찬",     Star="이문세"}
            };
```

```
            var listProfile =
                from profile in arrProfile
                join product in arrProduct on profile.Name equals product.Star
                select new
                {
                    Name = profile.Name,
                    Work = product.Title,
                    Height = profile.Height
                };

            Console.WriteLine("--- 내부 조인 결과 ---");
            foreach (var profile in listProfile)
            {
                Console.WriteLine("이름:{0}, 작품:{1}, 키:{2}cm",
                    profile.Name, profile.Work, profile.Height);
            }

            listProfile =
                from profile in arrProfile
                join product in arrProduct on profile.Name equals product.Star into ps
                from product in ps.DefaultIfEmpty(new Product() { Title = "그런거 없음" })
                select new
                {
                    Name = profile.Name,
                    Work = product.Title,
                    Height = profile.Height
                };

            Console.WriteLine();
            Console.WriteLine("--- 외부 조인 결과 ---");
            foreach (var profile in listProfile)
            {
                Console.WriteLine("이름:{0}, 작품:{1}, 키:{2}cm",
                    profile.Name, profile.Work, profile.Height);
            }
        }
    }
```

외부조인을 할 때는 먼저 join 절을 이용해서 조인을 수행한 후 그 결과를 임시 컬렉션에 저장하고,

이 임시 컬렉션에 대해 DefaultIfEmpty 연산을 수행해서 비어 있는 조인 결과에 빈 값을 채워 넣음.

DefaultIfEmpty 연산을 거친 임시 컬렉션에서 from 절을 통해 범위 변수를 뽑아내고,

이 범위 변수와 기준 데이터 원본에서 뽑아낸 범위 변수를 이용해서 결과를 추출해 내면 외부 조인!

5. 예제로 확인하기

- 예제에 사용되는 기본 코드

```
using System;
using System.Linq; // LINQ 쿼리 수행을 위해 반드시 포함해야 하는 네임스페이스
using System.Collections.Generic;
namespace ConsoleApplication1{
    class Person    {
        public string Name { get; set; }
        public int Age { get; set; }
        public string Address { get; set; }
        public override string ToString()        {
            return string.Format("{0}: {1} in {2}", Name, Age, Address);
        }
    }
    class MainLanguage    {
        public string Name { get; set; }
        public string Language { get; set; }
    }
    class Program    {
        static void Main(string[] args)        {
            List<Person> people = new List<Person>                    {
                new Person { Name = "Tom", Age = 63, Address = "Korea" },
                new Person { Name = "Winnie", Age = 40, Address = "Tibet" },
                new Person { Name = "Anders", Age = 47, Address = "Sudan" },
                new Person { Name = "Hans", Age = 25, Address = "Tibet" },
                new Person { Name = "Eureka", Age = 32, Address = "Sudan" },
                new Person { Name = "Hawk", Age = 15, Address = "Korea" },
                };
            List<MainLanguage> languages = new List<MainLanguage>                {
                    new MainLanguage { Name = "Anders", Language = "Delphi" },
                    new MainLanguage { Name = "Anders", Language = "C#" },
                    new MainLanguage { Name = "Tom", Language = "Borland C++" },
                    new MainLanguage { Name = "Hans", Language = "Visual C++" },
                    new MainLanguage { Name = "Winnie", Language = "R" },
                };
            //--- 코드 추가 부분

            //--- 코드 추가 부분
        }
    }
}
```

예제1) 전체 정보 출력(기본 LINQ 사용법)

```
            //LINQ 표현
            {
                var all = from person in people      //foreach(var person in people)
                          select person;             //yield return person;

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

```
// 확장 메서드 표현
            {
                IEnumerable<Person> all = people.Select((elem) => elem);

                foreach (var item in all)
                    Console.WriteLine(item);

            }
```

```
// 일반 메서드 표현
            {
                IEnumerable<Person> all = SelectFunc(people);

                foreach (var item in all)
                    Console.WriteLine(item);
            }
//정적 메서드 추가
static IEnumerable<Person> SelectFunc(List<Person> people)
        {
            foreach (var item in people)
            {
                yield return item;
            }
        }
```

```
Tom: 63 in Korea
Winnie: 40 in Tibet
Anders: 47 in Sudan
Hans: 25 in Tibet
Eureka: 32 in Sudan
Hawk: 15 in Korea
```

예제2) 원하는 필드 정보만 출력(기본 LINQ 사용법)

```
// 방법 1
            {
                // nameList의 타입은 IEnumerable<string>
                var nameList = from person in people
                               select person.Name;

                foreach (var item in nameList)                  {
                    Console.WriteLine(item);          //이름만 출력
                }
            }
            Console.WriteLine();
```

```
// 방법 2
            {
                var nameList = people.Select((elem) => elem.Name);

                foreach (var item in nameList)                  {
                    Console.WriteLine(item);          //이름만 출력
                }
            }
            Console.WriteLine();
```

```
// 익명 타입을 사용한 select
            {
                var dateList = from person in people
                           select new { Name = person.Name, Year = DateTime.Now.AddYears(-person.Age).Year };

                foreach (var item in dateList)                  {
                    Console.WriteLine(string.Format("{0} - {1}", item.Name, item.Year));
                }
            }
            Console.WriteLine();
```

```
// 확장 메서드를 이용한 select
            {
                var dateList = people.Select(
                           (elem) => new { Name = elem.Name, Year = DateTime.Now.AddYears(-elem.Age).Year }
                           );

                foreach (var item in dateList)                  {
                    Console.WriteLine(string.Format("{0} - {1}", item.Name, item.Year));
                }
            }
```

예제3) 부가적 문법 사용(where)

```
var ageOver30 = from person in people
                where person.Age > 30
                select person;
foreach (var item in ageOver30)          {
    Console.WriteLine(item);
}
```

예제4) 부가적 문법 사용(where)

구문을 통해IEnumerable<T> 타입인 people 컬렉션에서 특정 조건을 만족하는 데이터 필터링을 할 수 있다.

Where 조건의 반환타입이 bool형식인 점만 만족한다면 어떤 C# 코드든 자유롭게 사용가능하다.

```
var endWithS = from person in people
               where person.Name.EndsWith("s")
               select person;

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

예제5) 부가적 문법 사용(order by)

Orderby에 올 수 있는 값은 IComparable 인터페이스가 구현된 타입이면 하면 된다.

```
var ageSort = from person in people
              orderby person.Age // 나이순으로 정렬, 기본은 오름차순(ascending)
              select person;

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

// 역순 정렬
var ageSortDesc = from person in people
                  orderby person.Age descending
                  select person;
foreach (var item in ageSortDesc)          {
    Console.WriteLine(item);
}
```

예제6) 부가적 문법 사용(Grouping)

```
var addrGroup = from person in people
                group person by person.Address;

// group by로 묶여진 그룹을 나열하고,
foreach (var itemGroup in addrGroup) {
    Console.WriteLine(string.Format("[{0}]", itemGroup.Key));
    foreach (var item in itemGroup)  {  // 해당 그룹 내에 속한 항목을 나열
        Console.WriteLine(item);
    }
    Console.WriteLine();
}
Console.WriteLine();
Console.WriteLine();


var nameAgeList = from person in people
  group new { Name = person.Name, Age = person.Age } by person.Address;

 foreach (var itemGroup in nameAgeList)          {
    Console.WriteLine(string.Format("[{0}]", itemGroup.Key));
    foreach (var item in itemGroup)
    {
        Console.WriteLine(item);
    }
    Console.WriteLine();
 }
```

```
[Korea]
Tom: 63 in Korea
Hawk: 15 in Korea

[Tibet]
Winnie: 40 in Tibet
Hans: 25 in Tibet

[Sudan]
Anders: 47 in Sudan
Eureka: 32 in Sudan
```

```
[Korea]
{ Name = Tom, Age = 63 }
{ Name = Hawk, Age = 15 }

[Tibet]
{ Name = Winnie, Age = 40 }
{ Name = Hans, Age = 25 }

[Sudan]
{ Name = Anders, Age = 47 }
{ Name = Eureka, Age = 32 }
```

예제7) 부가적 문법 사용(Join)

```
//List<Person> people = new List<Person>
//List<MainLanguage> language = new List<MainLanguage>

// 내부 JOIN
   var nameToLangList = from person in people
      join language in languages on person.Name equals language.Name
            select new { Name = person.Name, Age = person.Age, Language = language.Language };

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

   Console.WriteLine();
   Console.WriteLine();

// 외부 JOIN
var nameToLangAllList = from person in people
            join language in languages on person.Name equals language.Name into lang
                  from language in lang.DefaultIfEmpty(new MainLanguage())
            select new { Name = person.Name, Age = person.Age, Language = language.Language };

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

* 첫 번째 join으로 langauges 컬렉션 내용을 lang이라는 이름의 임시 컬렉션으로 보냄.

  개별 요소에 대해 IEnumerable<T> 타입의 DefaultEmpty 확장 메서드를 호출한다.

- 이 메서드는 요소 중에서 값이 비어 있으면 인자로 전달된 "new MainFanguae()" 값을 사용한다.
- 이 때문에 출력 결과를 보면 내부 조인에 없던 Eureka와 Hawk 레코드가 Language 필드의 값이 "new MainFanguage()" 로 생성된 상태 값을 기반으로 포함돼 있다(아래 참조)

{ Name = Tom, Age = 63, Language = Borland C++ }	{ Name = Tom, Age = 63, Language = Borland C++ }
{ Name = Winnie, Age = 40, Language = R }	{ Name = Winnie, Age = 40, Language = R }
{ Name = Anders, Age = 47, Language = Delphi }	{ Name = Anders, Age = 47, Language = Delphi }
{ Name = Anders, Age = 47, Language = C# }	{ Name = Anders, Age = 47, Language = C# }
{ Name = Hans, Age = 25, Language = Visual C++ }	{ Name = Hans, Age = 25, Language = Visual C++ }
	{ Name = Eureka, Age = 32, Language = }
	{ Name = Hawk, Age = 15, Language = }

예제8) Max와 LINQ 쿼리의 조합

```
// 방법 1
            {
                var all = from person in people
                          where person.Address == "Korea"
                          select person;

                // 주소가 Korea인 사람 가운데 가장 높은 연령을 추출
                var oldestAge = all.Max((elem) => elem.Age);

                Console.WriteLine(oldestAge); // 출력 결과: 63
            }

            Console.WriteLine();

// 방법 2
            {
                var oldestAge = people.Where((elem) => elem.Address == "Korea").Max((elem) => elem.Age);
                Console.WriteLine(oldestAge); // 출력 결과: 63
            }
```

* Max와 같은 단일 값을 반환하는 메서드는 LINQ 쿼리가 수행되자 마자 결과값이 생성된다.

예제9) 지연 테스트 확인

IEnumerable<T>, IOrderedEnumerable<TElement>를 반환하는 메서드를 제외한 다른 모든 것들은 LINQ 식이 평가되면서 곧바로 실행된다.

```
class Program
    {
        static bool IsEqual(string arg1, string arg2)
        {
            Console.WriteLine("Executed");
            return arg1 == arg2;
        }

        static void Main(string[] args)
        {
            List<Person> people = new List<Person>
                {
                new Person { Name = "Tom", Age = 63, Address = "Korea" },
                new Person { Name = "Winnie", Age = 40, Address = "Tibet" },
```

```
            new Person { Name = "Anders", Age = 47, Address = "Sudan" },
            new Person { Name = "Hans", Age = 25, Address = "Tibet" },
            new Person { Name = "Eureka", Age = 32, Address = "Sudan" },
            new Person { Name = "Hawk", Age = 15, Address = "Korea" },
            };

        // LINQ 쿼리가 바로 실행됨
        Console.WriteLine("ToList() executed");
        var inKorea = (from person in people
                       where IsEqual(person.Address, "Korea")
                       select person).ToList();

        // 따라서 IsEqual 메서드의 실행으로 인해 화면에는 "Executed" 문자열이 출력됨
        Console.ReadLine();

        Console.WriteLine("IEnumerable<T> Where/Select evaluated");
        // IEnumerable<T>를 반환하므로 LINQ 쿼리가 평가만 되고 실행되지 않음
        var inKorea2 = from person in people
                       where IsEqual(person.Address, "Korea")
                       select person;

        // 따라서 IsEqual 메서드가 실행되지 않으므로 화면에는 "Executed" 문자열이 출력되지 않음.
        Console.ReadLine();

        // Take 확장 메서드 역시 IEnumerable<T>를 반환하므로 "Executed" 문자열이 출력되지 않음.
        Console.WriteLine("IEnumerable<T> Take evaluated");
        var firstPeople = inKorea2.Take(1);

        Console.ReadLine();

        // 열거를 시작했을 때 LINQ 쿼리가 실제로 실행됨.
        // 이때서야 비로소 "Executed" 문자열이 화면에 출력됨
        foreach (var item in firstPeople)
        {
            Console.WriteLine(item);
        }

        // 단일 값을 반환하는 Single 메서드의 호출은 곧바로 LINQ 쿼리가 실행되게 만듦
        Console.WriteLine(firstPeople.Single());

    }
}
```

예제10) 일관된 데이터 조회

LINQ 기술이 의미 있는 이유는 갖가지 다양한 데이터 원본에 대한 접근법을 단일화 했다는 것이다.

```
using System;
using System.Linq;
using System.Xml.Linq;
using System.IO;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string txt = @"
<people>
<person name='anders' age='47' />
<person name='winnie' age='13' />
</people>
";

            StringReader sr = new StringReader(txt);

            var xml = XElement.Load(sr);

            var query = from person in xml.Elements("person")
                        select person;

            foreach (var item in query)
            {
                Console.WriteLine(item.Attribute("name").Value + ": " + item.Attribute("age").Value);
            }
        }
    }
}
```