Всяка програма працює з даними:

- в пам'яті (колекції, масиви) // System.Array, System.Collections
- в реляційній БД // System.Data
- в XML-файлі // System.Xml
- метадані програми // System.Reflection

Як до них доступитися? ==> Language Integrated Query (LINQ) – строго типізована мова універсальних запитів, вбудована в граматику C# («випадково» схожа на SQL)

Приклад отримання зі збірки масиву статичних методів, позначених спеціальнгим атрибутом

```
Type sortMethodProviderType = assembly.GetType("SortMethodProvider");
MethodInfo[] methods = (from m in sortMethodProviderType.GetMethods()
                        where (m.IsStatic && m.IsDefined(methodNameAttrType, false))
                        select m).ToArray<MethodInfo>();
```

LINQ to Objects, Parallel LINQ, LINQ to DataSet, LINQ to Entites, LINQ to XML

Що в основі?

- Неявно типізовані локальні змінні (var – обов'язково, якщо тип невідомий до початку компіляції; на етапі компіляції визначається за присвоєнням і фіксується)
- Синтаксис ініціалізації об'єктів і колекцій (явне вказання відкритих властивостей у конструкторах, ініціалізатори колекцій → компактний код, анонімні типи)
- Лямбда-вирази (компактні анонімні функції, тип делегата виводить компілятор)
- Методи розширення (статичні методи зі спеціальним першим параметром)
- Анонімні типи (генеровані компілятором класи)

Універсальна мова доступу до даних

using System.Linq; //(System.Core.dll)

LINQ to Objects – до масивів і колекцій

LINQ to XML – до документів XML (System.Xml.Linq.dll)

LINQ to DataSet – до об'єктів DataSet ADO.NET (System.Data.DataSetExtensions.dll)

LINQ to Entities – всередині Entity Framework

Parallel LINQ (PLINQ) – паралельна обробка даних, отриманих LINQ запитом

Запити строго типізовані, оснащені метаданими, використовують об'єктну модель

Приклади

StartExamples
Наведено приклади LINQ запитів до масиву чисел, масиву рядків, колекції простих об'єктів. У запитах можна побачити: 1) задання джерела даних, 2) операцію обмеження, 3) спосіб впорядкування, 4) спосіб відображення (в тому числі, проекцію). Є приклади, що демонструють «лініву» природу запитів: LINQ запит не є терміновою дією – це об'єкт, який описує алгоритм дії, свого роду конструктор, дія відбудеться тільки тоді, коли в об'єкта-запиту попросять надати дані.
- запит до масиву
- відкладене і термінове виконання: запити мають «лініву» природу
- рефлексія типу результатів запиту: тип результату виводить компілятор, можна з'ясувати, що для різних запитів це – різного роду ітератори.
- застосування до узагальненого контейнера; для звичайного потрібне приведення типу
TypeOf<T> працює як фільтр екземплярів вказаного типу.

LINQSamples:

Клас *Racer* описує гонщика Формули 1, містить його ім'я, країну, кількості стартів і перемог. У проекті екземпляри класу є носіями даних з різними властивостями. Клас *Formula1* є джерелом даних, повідомляє колекцію гонщиків. До цієї колекції і будемо надсилати різноманітні запити.

- Традиційні методи перетворення контейнера
Знаходить чемпіонів з Бразилії, впорядковує їх за спаданням кількості перемог, використовує методи класу.

Наступні приклади показують зсередини, як працює LINQ, переходячи від конкретних інструментів до більш загальних
- Методи розширення + делегати
- Методи розширення + лямбда-вирази
- LINQ → методи розширення працюють «за кулісами»

QueryOperators

Достатньо складні запити. Використано дві пов'язані колекції об'єктів: пілоти і конструктори Формули 1.

- Фільтрування, фільтрування з використанням індекса
- Приведення та фільтрування типу для неузагальнених колекцій
- Складена конструкція from
- Впорядкування за багатьма критеріями
- Термінове отримання частини результатів
- Розбиття на групи. Групування зі створенням вкладених колекцій
- Агрегація результатів
- Поділ результатів на частини
- Операції з множинами. Запит у лямбда-виразі
- Сполучення послідовностей в одну заданим перетворенням
- Побудова відображень
- Генерування діапазонів

Повернення результатів запиту – в строго типізованій змінній

```
public static IEnumerable<TSource> Where<TSource> (
      this IEnumerable<TSource> source,
      Func<TSource, bool> predicate)
{
    foreach (TSource item in source) if (predicate(item)) yield return item;
}
```

Часто вживані операції запитів

Операції	Призначення
from in	Визначають основу кожного запиту
where	Визначає обмеження на елементи колекції, які хочуть отримати
select	Вибирає послідовність з контейнера
join on, equals, into	Виконують сполучення
orderby ascending, descending	Задають спосіб впорядкування
group by	Видають підмножини, згруповані за вказаним значенням

Методи розширення класу System.Linq.Enumerable:

- трансформатори (Reverse<>(), ToArray<>(), ToList<>(), …)
- суматори (Count<>(), Sum<>(),Min<>(), Max<>(), …)
- дії з множинами (Distinct<>(), Union<>(), Intersect<>(), …)

Методи, що реалізують запити LINQ

Назва	Опис
Where, OfType<TResult>	Фільтри: за предикатом, за типом
Select, SelectMany	Визначають проекцію вибірки з одним результатом для елемента чи з колекцією результатів
OrderBy, OrderByDescending ThenBy, ThenByDescending, Reverse	Змінюють порядок елементів результату запиту. ThenBy використовують довільну кількість разів для опису вторинних ключів порівняння
Join, GroupJoin	Комбінують різні колекції в одну
GroupBy ToLookup	Групують дані за певною ознакою. ToLookup створює словник «один до багатьох»
Any All Contains	Квантифікатори перевіряють, чи хоча б один, чи всі задовольняють предикат; чи є заданий елемент
Take, Skip, TakeWhile, SkipWhile	Повертають підмножину колекції
Distinct, Union, Intersect, Except, Zip	Операції з множинами
First FirstOrDefault Last LastOrDefault ElementAt ElementAtOrDefault Single SingleOrDefault	Повертають один елемент, що задовольняє певну умову. Якщо таких є більше, то Single генерує виняток
Count Sum Min Max Average Aggregate	Обчислювачі за елементами колекції
ToArray AsEnumerable ToList ToDictionary Cast<TResult>	Перетворювачі, форсують виконання запитів

Empty Range Repeat	Генератори колекцій

LINQ to DataSet

```
InventoryTableAdapter da = new InventoryTableAdapter();
AutoLotDataSet.InventoryDataTable data = da.GetData();
var cars = from car in data.AsEnumerable()
           where (string)car["Color"] == "Red"
           select new { ID= (int)car["CarID"], Make = (string)car["Make"] };
```

LINQ to Entities

```
using (AutoLotEntities context = new AutoLotEntities())
{   var carToDelete = from c in context.Cars where c.CarID == 2222 select c;
    if (carToDelete != null)
    {       context.DeleteObject(carToDelete); context.SaveChanges();       }
}
```

LINQ to XML

```
XDocument inventoryDoc = XDocument.Load("Inventory.xml");

var makeInfo = from car in inventoryDoc.Descendants("Car")
      where (string)car.Element("Make") == "BMW"
      select car.Element("Color").Value;
```