Введение в LINQ

- Основы LINQ
- Стандартные операторы запроса
- Запросы-выражения

Основы

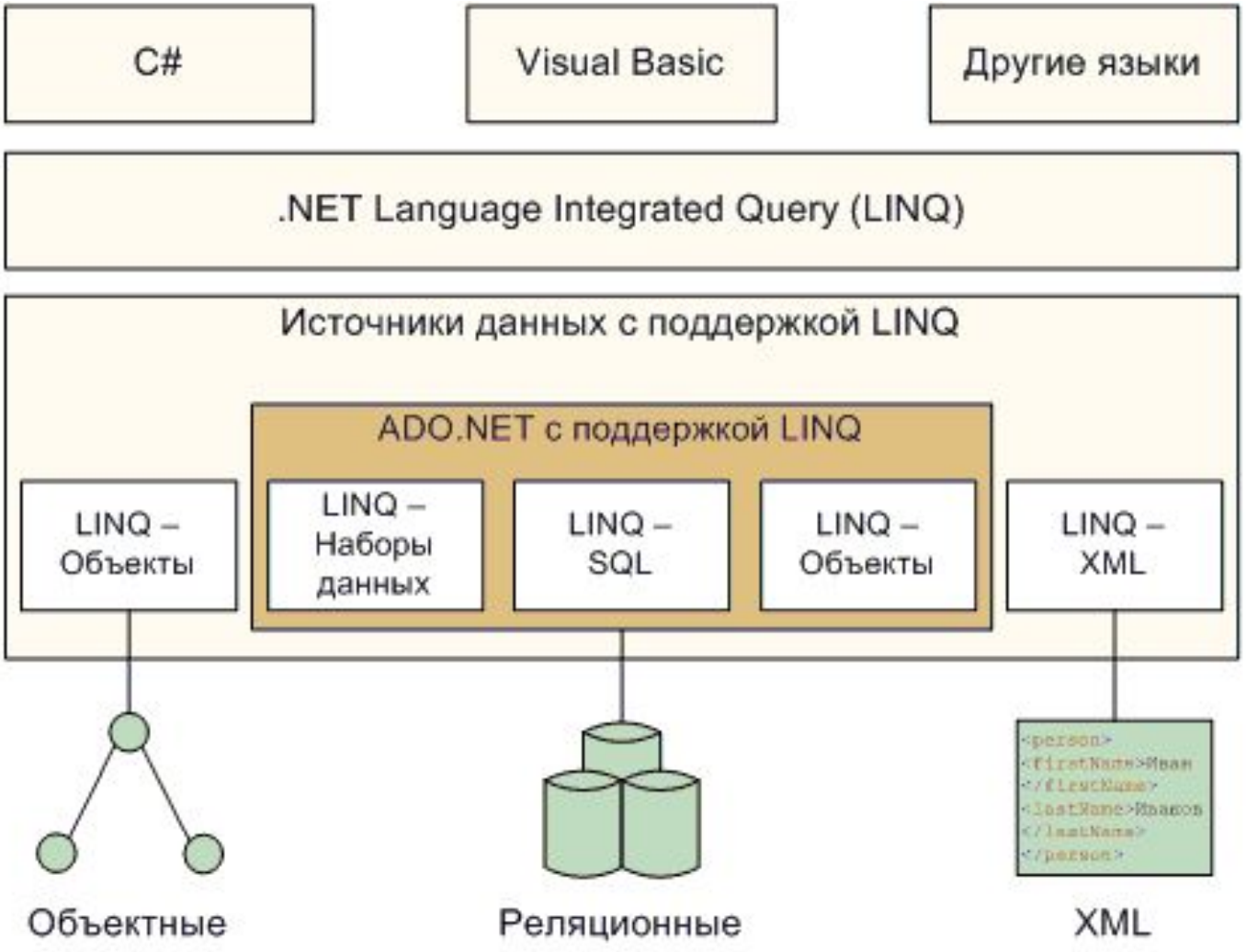
LINQ

LINQ (Language Integrated Query) – технология, представляющая собой набор функций, позволяющих писать структурированные безопасные запросы к локальным объектам-коллекциям и удаленным источникам данных.

LINQ – это язык структурированных запросов к любым массивам и коллекциям объектов, управляемых вашей программой.

Архитектура

LINQ



Запросы

LINQ

SQL-запрос:

```
select * from Student where Stipend > 500
and Kurs > 3
```

Конструкция LINQ:

```
var result =
 from St in Student
 where St.Stipend > 500 and Kurs > 3
 select St;
```

Правила составления запросов LINQ

1. Первый оператор: **from** – это объявление переменной диапазона.
2. Затем идет ограничивающий оператор: **where** – фильтрация данных.
3. **OrderBy, ThenBy** – поле сортировки по одному или нескольким ключам (по возрастанию, убыванию).
4. Оператор проекции: **group** или **select** – значения, полученные в запросе.
5. **join** – связывание элементов из последовательностей.

Запросы

LINQ

Интегрированный запрос:

```
var contacts = from c in customers
               where c.City == "Москва"
               select new { c.City, c.Phone };
```

С использованием лямбда-выражений:

```
var contacts = customers
               .Where (c => c.City == "Москва")
               .Select (c => new { c.Name, c. Phone } );
```

Лямбда выражения

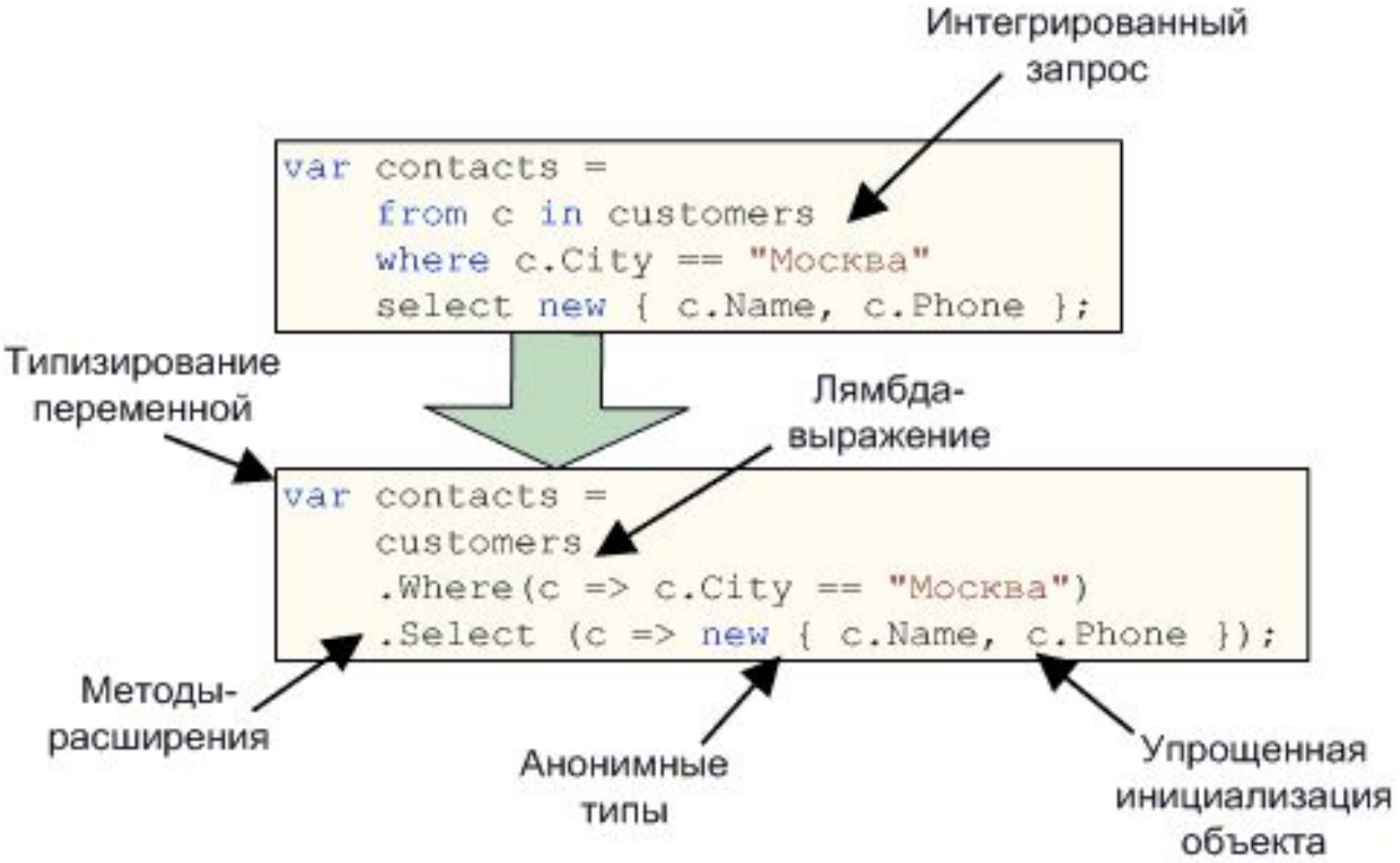
x => x * 5

Лямбда-оператор **=>** читается как «переходит в».

Левая часть лямбда-оператора определяет параметры ввода, а правая часть содержит выражение.

Запросы

LINQ



Интегрированные запросы

Запросы-выражения поддерживают только ограниченный набор операторов:

- Where
- Select
- SelectMany
- OrderBy
- ThenBy
- OrderByDescending
- ThenByDescending
- GroupBy
- Join
- GroupJoin

Интегрированные запросы

Можно использовать *смешанный синтаксис*:

```
string[] names = { "Петр", "Маша", "Марина", "Олег"
};

IEnumerable<string> query =
   from  n  in  names
   where  n.Length == names.Max (m => m.Length)
   select  n;
```

Запросы

LINQ

Ключевое слово var – упрощает синтаксис. Например, заменяет явное задание результата запроса IEnumerable<T>

```
IEnumerable<Students>  studentInMoskow =
   from  st  in  student
   where  st.City == "Москва"
   select  st;

var  studentInMoskow =
   from  st  in  student
   where  st.City == "Москва"
   select  st;
```

Операторы запроса

Оператор запроса (query operator) – метод, преобразующий последовательность.

Обычно операторы запроса принимают *входящую последовательность* и возвращают преобразованную *исходящую последовательность*.

Оператор Where

Извлечь из простого массива все имена, длиной как минимум 6 символов.

```
string[] names = { «Петр», «Николай», «Алексей» };

IEnumerable<string>  query =
    Enumerable.Where(names, n => n.Length > 5);

foreach (string  k  in  query)
    Console.WriteLine (k);
```

Оператор Where

```
IEnumerable<string>  query =
    Enumerable.Where(names, n => n.Length > 5);
```

Можно вызывать метод Where как **экземплярный**:

```
IEnumerable<string>  query =
    names.Where(n => n.Length > 5);
```

Агрегирующие операторы

Агрегирующие операторы **Count, Min, Max, Sum и Average** возвращают скалярное значение числового типа.

```
int[] numbers = { 10, 9, 8, 7, 6 };

int count = numbers.Count(); // 5
int min = numbers.Min(); // 6
int max = numbers.Max(); // 10
double avg = numbers.Average(); // 8
```

Кванторы

Кванторы в LINQ возвращают логическое значение. К ним относятся **Contains, Any, All, SequenceEquals**:

```
int[] numbers = { 10, 9, 8, 7, 6 };

bool  hasTheNum = numbers.Contains (9);
bool  hasMoreThanZeroElements = numbers.Any();
bool  hasOddNum = numbers.Any (n => n % 2 == 1);
bool  allOddNums = numbers.All (n => n % 2 == 1);
```

Операторы комплектовщики

Операторы комплектовщики **Concat и Union** принимают две входящие последовательности одного типа и образуют новую, присоединяя одну последовательность к другой:

```
int[]  seq1 = { 1, 2, 3 },  seq2 = { 3, 4, 5 };

IEnumerable<int>
   concat = seq1.Concat (seq2), // { 1, 2, 3, 3, 4, 5 }
   union = seq1.Union (seq2), // { 1, 2, 3, 4, 5 }
```

Операторы комплектовщики

```
int[]  seq1 = { 1, 2, 3 },  seq2 = { 3, 4, 5 };

IEnumerable<int>
    commonality = seq1.Intersect (seq2), // { 3 }
    difference1 = seq1.Except (seq2), // { 1, 2 }
    difference2 = seq2.Except (seq1); // { 4, 5 }
```

Отложенное выполнение

Операторы запроса выполняются не в момент их создания, а в момент перечисления исходящей последовательности:

```
var numbers = new List<int> { 3 };
numbers.Add(5);

IEnumerable<int> query = numbers.Select(n => n * 10);

numbers.Add(9); // добавляем элемент

foreach (int  k  in  query)
      MessageBox.Show(k.ToString());
```

Стандартные операторы запроса

1. Фильтрующие операторы
2. Проецирующие операторы
3. Объединяющие операторы
4. Упорядочивающие операторы
5. Группирующие операторы
6. Операторы комплектовщики
7. Элементные операторы
8. Агрегирующие операторы
9. Операторы кванторы
10. Преобразующие-импортирующие операторы
11. Преобразующие-экспортирующие операторы
12. Генерирующие операторы

Фильтрующие операторы

Where - возвращает подмножество элементов, удовлетворяющих переданному условию
Take - возвращает первые *n* элементов, отбрасывая остальные
Skip - отбрасывает первые *n* элементов и возвращает оставшиеся
TakeWhile - извлекает элементы из входящей последовательности пока переданный предикат возвращает true
SkipWhile - отбрасывает элементы из входящей последовательности пока переданный предикат возвращает true, затем возвращает остаток
Distinct - возвращает коллекцию с исключенными повторами

Проецирующие операторы

Select - преобразует каждый элемент входящей последовательности с помощью переданного лямбда выражения

SelectMany - преобразует элементы нескольких входящих последовательностей и объединяет получившиеся последовательности в одну (одноуровневую)

Объединяющие операторы

Join - объединяет элементы из двух последовательностей в один одноуровневый набор

GroupJoin - объединяет элементы из двух последовательностей в один иерархический (многоуровневый) набор

Zip - перебирает две последовательности за один проход и возвращает последовательность, содержащую результаты выполнения функции (переданной в качестве аргумента) над парами элементов из двух последовательностей

Операторы запросов

Упорядочивающие операторы:

OrderBy, **ThenBy** - возвращают элементы отсортированные в возрастающем порядке
OrderByDescending, **ThenByDescending** - возвращают элементы отсортированный в убывающем порядке
Reverse - возвращает элементы в обратном порядке

Группирующие операторы:

GroupBy - группирует элементы последовательности в подмножества (подпоследовательности)

Операторы комплектовщики

Concat - объединяет две последовательности

Union - объединяет две последовательности, удаляя повторы

Intersect - возвращает элементы, присутствующие в обеих последовательностях

Except - возвращает элементы первой последовательности, отсутствующие во второй

Элементные операторы

First, **FirstOrDefault** - возвращают первый элемент последовательности или первый элемент, удовлетворяющий переданному предикату
Last, **LastOrDefault** - возвращают последний элемент последовательности или последний элемент, удовлетворяющий переданному предикату
Single, **SingleOrDefault** - эквивалент First/FirstOrDefault, если совпадений больше одного, выбрасывает исключение
ElementAt, **ElementAtOrDefault** - возвращает элемент с указанной позицией
DefaultIfEmpty - возвращает элементы последовательности илиодноэлементную коллекцию со значением по умолчанию - default(TSource), если

Агрегирующие операторы

Count, **LongCount** - возвращает общее число элементов во входящей последовательности или число элементов, удовлетворяющих переданному предикату

Min - возвращает наименьший элемент в последовательности

Max - возвращает наибольший элемент в последовательности

Sum - вычисляет сумму элементов в последовательности

Average - вычисляет среднее значение элементов в последовательности

Aggregate - выполняет пользовательскую агрегацию

Операторы кванторы

Contains - возвращает true если входящая последовательность содержит переданный элемент

Any - возвращает true если хотя бы один элемент последовательности удовлетворяет переданному предикату

All - возвращает true если все элементы последовательности удовлетворяют переданному предикату

SequenceEqual - возвращает true если вторая (переданная в качестве аргумента) последовательность содержит элементы идентичные элементам входящей

Преобразующие операторы

OfType - преобразует IEnumerable в IEnumerable<T>, отбрасывая элементы неподходящего типа

Cast - преобразует IEnumerable в IEnumerable<T>, выбрасывая исключение если встречаются элементы неподходящего типа

ToArray - преобразует IEnumerable<T> в T[]

ToList - преобразует IEnumerable<T> в List<T>

ToDictionary - преобразует IEnumerable<T> в Dictionary<TKey,TValue>

ToLookup - преобразует IEnumerable<T> в ILookup<TKey,TElement>

AsEnumerable - приводит последовательность к IEnumerable<T>

AsQueryable - приводит последовательность к типу IQueryable<T>

Запросы

LINQ

Вывести в алфавитном порядке все имена, в составе букв которых содержится буква «а».

```
string[] names = { "Толя", "Маша", "Даша", "Оля", "Алина" };

IEnumerable<string>  query = names
        .Where(n => n.Contains("а"))
        .OrderBy(n => n)
        .Select(n => n);

foreach (string  name  in  query)
        MessageBox.Show(name);
```

Ключевое слово let

Ключевое слово **let** вводит новую переменную параллельно переменной диапазона:

```
string[]  names = { "Петр", "Маша", "Марина", "Олег"
};

IEnumerable<string>  query =
            from  n  in  names
            let  n_new = n.Replace("a", "")
            orderby  n_new
            select  n + " - " + n_new;
```

Запросы с продолжениями

Пример: Найти в предложении все слова, начинающиеся на букву «м»:

```
IEnumerable<string>  query =
   from  c  in  "Мама мыла раму".Split()
   where  c.StartsWith("М") || c.StartsWith("м")
   select  c;
//С использованием новой переменной диапазона:
IEnumerable<string> query =
   from  c  in  "Мама мыла раму".Split()
   select  c.ToUpper()  into  c_new
   where  c_new.StartsWith("М")
   select  c_new;
//Краткая запись:
IEnumerable<string> query = "Мама мыла раму".Split()
   .Select(c => c.ToUpper())
   .Where(upper => upper.StartsWith("М"));
```

Множественные генераторы

```
int[] arr1 = { 9, 10, 11 };
string[] arr2 = { "а", "б"};

IEnumerable<string>  query =
   from  a1  in  arr1
   from  a2  in  arr2
   select  a1.ToString() + a2;

foreach (string  name  in  query)
   MessageBox.Show(name);

//Краткая запись запроса:
IEnumerable<string>  query =  arr1.SelectMany (
   a1 => arr2,
   (a1, a2) => (a1.ToString() + a2));
```

Операторы объединения

Оператор объединения **Join** поддерживает эквивалентное объединение (т.е. объединяющее условие должно использовать оператор эквивалентности).

Синтаксис:

from <внешн-переменная> **in** <внешн-последовательность>
join <внутр-переменная> **in** <внутр-последовательность>
on <внешн-ключ> **equals** <внутренний-ключ>

Операторы объединения

Вывести список продуктов, купленных детьми.

```
var children = new[]
{
        new { id = 10, name = "Оля" },
        new { id = 20, name = "Паша" }
};

var goods = new[]
{
         new { id = 10, product = "Кораблик" },
         new { id = 20, product = "Мяч" }
};
```

Операторы объединения

```
IEnumerable<string>  query =  from c in children
        join p  in  goods  on  c.id  equals  p.id
        select c.ame + " - " + p.product;

var  query = children
.Join(goods, c => c.id,  g => g.id,  (c, g) => new { c, g });


foreach (var q in query)
     Console.WriteLine(q.c.name + " - " + q.g.product);
```

Операторы объединения

Простейший оператор объединения **Zip** – перебирает две последовательности за один проход и возвращает новую последовательность – результат выполнения функции над соответствующими парами элементов последовательностей:

```
int[]  numbers = { 3, 5 };
string[]  words = {"три", "пять", "семь" };
IEnumerable<string>  query =
    numbers.Zip (words,  (n, w) => n + " = " + w);

// Результат:
3 = three
5 = five
```

Операторы группировки

Оператор GroupBy группирует одноуровневую входящую последовательность в последовательность *групп:*

```
string[] nm = { "Миша", "Маша", "Ольга", "Зоя", "Тим" };
var query =
   from name in nm
   group name by name.Length;

string str = "";
foreach (IGrouping<int, string> grouping in query)
{
   str = " Длина имени = " + grouping.Key + ": ";
   foreach (string name in grouping)
      str += name + ", ";
   MessageBox.Show(str);
}
```

Примеры

Вывести на консоль все четные числа массива.

```
//Источник данных
int[]  intNum = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };

// Создаем запрос
var  queryNumbers  =  from  number  in  intNum
                      where  (number % 2) == 0
                      select  number;

//Выполнение запроса
foreach (int  itemNumber  in  queryNumbers)
            Console.WriteLine(itemNumber);
```

Примеры

Вывести всех студентов, проживающих в Воронеже.

```
var student = new[]
    {
        new {Name = "Нина", Surname = "Иванова", City = "Воронеж"},
        new {Name = "Паша", Surname = "Кузнецов", City = "Воронеж"}
    };

var query =
    from st in student
    where st.City == "Воронеж"
    select st;

foreach (var s in query)
{
    MessageBox.Show(s.Surname + s.Name);
}
```

Лабораторная работа 8

1. Вывести сведения о *студентах*, фамилии, либо имена которых начинаются на «И».
2. Вывести данные *об оценках* студентов второго курса, отсортированные по фамилии.
3. Вывести фамилии *преподавателей* и названия *предметов* которые они ведут.
4. Вывести фамилии и имена *студентов*, обучающихся на курсе не ниже 3 и получающих наибольшую стипендию.
5. Рассчитать среднюю *оценку* студентов, полученную на экзамене по информатике.

Студенты:

	student_id integer	surname text	name text	stipend double precision	kurs integer	city character(50)	birthday date	univ_id integer
1	1	Иванов	Иван	150	1	Орел	1982-03-12	10
2	3	Петров	Петр	200	3	Курск	1980-12-01	10
3	6	Сидоров	Вадим	150	4	Москва	1979-06-07	22
4	55	Белкин	Вадим	250	5	Воронеж	1980-01-07	10
5	12	Зайцева	Ольга	10000	2	Липецк	1981-05-01	10
6	10	Кузнецов	Борис	10000	2	Брянск	1900-01-21	10
7	100	Иванов	Иван		4	Якутск	2000-12-12	12

Экзаменационные оценки:

	exam_id integer	student_id integer	subj_id integer	mark integer	exam_date date
1	145	12	10	5	2000-01-12
2	34	32	10	4	2000-01-23
3	75	55	10	5	2000-01-05
4	238	12	22	3	1999-06-17
5	639	55	22		1999-06-22
6	43	6	22	4	2000-01-18

Преподаватели:

	lecturer_id integer	surname text	name text	city text	subj_id integer
1	24	Колесников	Борис	Воронеж	10
2	46	Никонов	Иван	Воронеж	10
3	276	Николаев	Виктор	Воронеж	10
4	74	Лагутин	Павел	Москва	22
5	108	Струков	Николай	Москва	22
6	328	Сорокин	Павел	Орел	10

Предметы:

	subj_id integer	subj_name text	hour integer	semester integer
1	10	Информатика	56	1
2	22	Физика	34	1
3	43	Математика	56	2
4	94	Английский	56	3
5	73	Физкультура	34	5
6	56	История	34	4