**Применение вложенных операторов from**

Запрос может состоять из нескольких операторов from, которые оказываются в

этом случае вложенными. Такие операторы from находят применение в тех случаях,

когда по запросу требуется получить данные из двух разных источников. Рассмотрим

простой пример, в котором два вложенных оператора from используются в запросе

для циклического обращения к элементам двух разных массивов символов. В итоге

по такому запросу формируется последовательность результатов, содержащая все возможные

комбинации двух наборов символов.

(***glava19\_10***)

//this class has result of query

class ChrPair

{

public char First;

public char Second;

public ChrPair(char c, char c2)

{

First = c;

Second = c2;

}

}

class MultipleFroms

{

static void Main()

{

char[] chrs = { 'A', 'B', 'C' };

char[] chrs2 = { 'X', 'Y', 'Z' };

//first operator from organize cycle call

//to array chrs, and the second from - cycle call to

//array chrs2

var pairs = from ch1 in chrs

from ch2 in chrs2

select new ChrPair(ch1, ch2);

Console.WriteLine("All pairs ABC and XYZ: ");

foreach (var p in pairs) Console.WriteLine("{0} {1}", p.First, p.Second);

}

}

Этот пример кода начинается с создания класса ChrPair, в котором содержатся

результаты запроса. Затем в нем создаются два массива, chrs и chrs2, и, наконец,

формируется следующий запрос для получения всех возможных комбинаций двух последовательностей

результатов.

var pairs = from ch1 in chrs

from ch2 in chrs2

select new ChrPair(ch1, ch2);

Во вложенных операторах from организуется циклическое обращение к обоим

массивам символов, chrs и chrs2. Сначала из массива chrs получается символ, сохраняемый

в переменной ch1. Затем перечисляется содержимое массива chrs2. На

каждом шаге циклического обращения во внутреннем операторе from символ из

массива chrs2 сохраняется в переменной сh2 и далее выполняется оператор select.

В результате выполнения оператора select создается новый объект типа ChrPair,

содержащий пару символов, которые сохраняются в переменных ch1 и ch2 на каждом

шаге циклического обращения к массиву во внутреннем операторе from. А в конечном

итоге получается объект типа ChrPair, содержащий все возможные сочетания извлекаемых

символов.

Вложенные операторы from применяются также для циклического обращения к

источнику данных, который содержится в другом источнике данных. Соответствующий

пример приведен в разделе *"*Применение оператора let для создания временной

переменной в запросе*"* далее в этой главе.

**Группирование результатов с помощью оператора group**

Одним из самых эффективных средств формирования запроса является оператор

group, поскольку он позволяет группировать полученные результаты по ключам. Используя

последовательность сгруппированных результатов, можно без особого труда

получить доступ ко всем данным, связанным с ключом. Благодаря этому свойству

оператора group доступ к данным, организованным в последовательности связанных

элементов, осуществляется просто и эффективно. Оператор group является одним из

двух операторов, которыми может оканчиваться запрос. (Вторым оператором, завершающим

запрос, является select.) Ниже приведена общая форма оператора group.

group переменная\_диапазона by ключ

Этот оператор возвращает данные, сгруппированные в последовательности, причем

каждую последовательность обозначает общий *ключ.*

Результатом выполнения оператора group является последовательность, состоящая

из элементов типа IGrouping<TKey, TElement>, т.е. обобщенного интерфейса,

объявляемого в пространстве имен System.Linq. В этом интерфейсе определена коллекция

объектов с общим ключом. Типом переменной запроса, возвращающего группу,

является IEnumerable<IGrouping<TKey, TElement>>. ***В интерфейсе IGrouping***

***определено также доступное только для чтения свойство Key, возвращающее ключ,***

***связанный с каждой коллекцией.***

Ниже приведен пример, демонстрирующий применение оператора group. В коде

этого примера сначала объявляется массив, содержащий список веб-сайтов, а затем

формируется запрос, в котором этот список группируется по имени домена самого

верхнего уровня, например .org или .com.

(***glava19\_11***)

class GroupDemo

{

static void Main()

{

string[] websites =

{

"hsNameA.com", "hsNameB.net", "hsNameC.net",

"hsNameD.com", "hsNameE.org", "hsNameF.org",

"hsNameG.tv", "hsNameH.net", "hsNameI.tv"

};

//make query to get list of we-sites

//grouped by name of domain of higher level

var webAddrs = from addr in websites

where addr.LastIndexOf('.') != -1

group addr by addr.Substring(addr.LastIndexOf('.'));

//call query and ouput results

foreach(var sites in webAddrs)

{

Console.WriteLine("Web - sites, grouped "

+ "by name of domain" + sites.Key);

foreach (var site in sites)

Console.WriteLine(" " + site);

Console.WriteLine();

}

}

}

Как следует из приведенного выше результата, данные, получаемые по запросу,

группируются по имени домена самого верхнего уровня в адресе веб-сайта. Обратите

внимание на то, как это делается в операторе group из следующего запроса.

var webAddrs = from addr in websites

where addr.LastIndexOf('.') != -1

group addr by addr.Substring(addr.LastIndexOf('.'));

Ключ в этом операторе создается с помощью методов LastIndexOf()

и Substring(), определенных для данных типа string. (Эти методы упоминаются

в главе 7, посвященной массивам и строкам. Вариант метода Substring(), используемый

в данном примере, возвращает подстроку, начинающуюся с места, обозначаемого

индексом, и продолжающуюся до конца вызывающей строки.) Индекс последней

точки в адресе веб-сайта определяется с помощью метода LastIndexOf().

По этому индексу в методе Substring() создается оставшаяся часть строки, в которой

содержится имя домена самого верхнего уровня. Обратите внимание на то,

что в операторе where отсеиваются все строки, которые не содержат точку. Метод

LastIndexOf() возвращает -1, если указанная подстрока не содержится в вызывающей

строке.

(Ключ создается при использовании этих двух методов класса Стринг, и он привязывается к определенным значеням. Это все можно увидеть в дебагере.)

Последовательность результатов, получаемых при выполнении запроса, хранящегося

в переменной webAddrs, представляет собой список групп, поэтому для доступа

к каждому члену группы требуются два цикла foreach. Доступ к каждой группе осуществляется

во внешнем цикле, а члены внутри группы перечисляются во внутреннем

цикле. Переменная шага внешнего цикла foreach должна быть экземпляром интерфейса

IGrouping, совместимым с ключом и типом элемента данных. В рассматриваемом

здесь примере ключи и элементы данных относятся к типу string. Поэтому

переменная sites шага внешнего цикла имеет тип IGrouping<string, string>,

а переменная site шага внутреннего цикла — тип string. Ради краткости данного

примера обе переменные объявляются неявно, хотя их можно объявить и явным образом,

как показано ниже.

foreach (IGrouping<string, string> sites in webAddrs)

{

Console.WriteLine("Веб-сайты, сгруппированные " +

"по имени домена" + sites.Key);

foreach (string site in sites)

Console.WriteLine(" " + site);

Console.WriteLine();

}

**Продолжение запроса с помощью оператора into**

При использовании в запросе оператора select или group иногда требуется сформировать

временный результат, который будет служить *продолжением запроса* для получения

окончательного результата. Такое продолжение осуществляется с помощью

оператора into в комбинации с оператором select или group. Ниже приведена общая

форма оператора into:

into имя тело\_запроса

где имя обозначает конкретное имя переменной диапазона, используемой для циклического

обращения к временному результату в продолжении запроса, на которое

указывает *тело\_запроса.* Когда оператор into используется вместе с оператором

select или group, то его называют продолжением запроса, поскольку он продолжает

запрос. По существу, продолжение запроса воплощает в себе принцип построения

нового запроса по результатам предыдущего.

Ниже приведен пример программы, в которой оператор into используется вместе

с оператором group. Эта программа является переработанным вариантом предыдущего

примера, в котором список веб-сайтов формируется по имени домена самого

верхнего уровня. А в данном примере первоначальные результаты запроса сохраняются

в переменной диапазона ws и затем отбираются для исключения всех групп, состоящих

менее чем из трех элементов.

(***glava19\_12***)

class GroupDemo

{

static void Main()

{

string[] websites =

{

"hsNameA.com", "hsNameB.net", "hsNameC.net",

"hsNameD.com", "hsNameE.org", "hsNameF.org",

"hsNameG.tv", "hsNameH.net", "hsNameI.tv"

};

//make query to get list of we-sites

//grouped by name of domain of higher level

//but chose groups if there are more then 2

//ws - variable of groups, returned when

//first query is called

var webAddrs = from addr in websites

where addr.LastIndexOf('.') != -1

group addr by addr.Substring(addr.LastIndexOf('.'))

into ws

where ws.Count() > 2

select ws;

//call query and ouput results

foreach (var sites in webAddrs)

{

Console.WriteLine("Web - sites, grouped "

+ "by name of domain" + sites.Key);

foreach (var site in sites)

Console.WriteLine(" " + site);

Console.WriteLine();

}

}

}

Обратите особое внимание в данном примере программы на следующую последовательность

операторов в формируемом запросе.

group addr by addr.Substring(addr.LastIndexOf('.'))

into ws

where ws.Count() > 2

select ws;

Сначала результаты выполнения оператора group сохраняются как временные

для последующей обработки оператором where. В качестве переменной диапазона

в данный момент служит переменная ws. Она охватывает все группы, возвращаемые

оператором group. Затем результаты запроса отбираются в операторе where с таким

расчетом, чтобы в конечном итоге остались только те группы, которые содержат больше

двух членов. Для этой цели вызывается метод Count(), который является *методом*

*расширения* и реализуется для всех объектов типа IEnumerable. Он возвращает

количество элементов в последовательности. (Подробнее о методах расширения речь

пойдет далее в этой главе.) А получающаяся в итоге последовательность групп возвращается

оператором select.

**Применение оператора let для создания**

**временной переменной в запросе**

Иногда возникает потребность временно сохранить некоторое значение в самом

запросе. Допустим, что требуется создать переменную перечислимого типа, которую

можно будет затем запросить, или же сохранить некоторое значение, чтобы в дальнейшем

использовать его в операторе where. Независимо от преследуемой цели, эти

виды функций могут быть осуществлены с помощью оператора let. Ниже приведена

общая форма оператора let:

let имя = выражение

где *имя* обозначает идентификатор, получающий значение, которое дает *выражение.*

Тип имени выводится из типа выражения.

В приведенном ниже примере программы демонстрируется применение оператора

let для создания еще одного перечислимого источника данных. В качестве входных

данных в запрос вводится массив символьных строк, которые затем преобразуются в

массивы типа char. Для этой цели служит еще один метод обработки строк, называемый

ToCharArray() и возвращающий массив, содержащий символы в строке. Полученный

результат присваивается переменной chrArray, которая затем используется

во вложенном операторе from для извлечения отдельных символов из массива. И наконец,

полученные символы сортируются в запросе, и из них формируется результирующая

последовательность.

(***glava19\_13***)

class LetDemo

{

static void Main()

{

string[] strs = { "alpha", "beta", "gama" };

//make query to get symbols, returned from strings

//in sorted order. Take a look at from

var chrs = from str in strs

let chrArray = str.ToCharArray()

from ch in chrArray

orderby ch

select ch;

Console.WriteLine("Sorted symbols: ");

foreach (var c in chrs) Console.Write(c + " ");

Console.WriteLine();

}

}

Обратите внимание в данном примере программы на то, что в операторе let переменной

chrArray присваивается ссылка на массив, возвращаемый методом str.

ToCharArray().

let chrArray = str.ToCharArray()

После оператора let переменная chrArray может использоваться в остальных

операторах, составляющих запрос. А поскольку все массивы в C# преобразуются в тип

IEnumerable<T>, то переменную chrArray можно использовать в качестве источника

данных для запроса во втором, вложенном операторе from. Именно это и происходит

в рассматриваемом здесь примере, где вложенный оператор from служит для перечисления

в массиве отдельных символов, которые затем сортируются по нарастающей

и возвращаются в виде конечного результата.

Оператор let может также использоваться для хранения неперечислимого значения.

В качестве примера ниже приведен более эффективный вариант формирования

запроса в программе IntoDemo из предыдущего раздела.

var webAddrs = from addr in websites

let idx = addr.LastIndexOf('.')

where idx != -1

group addr by addr.Substring(idx)

into ws

where ws.Count() > 2

select ws;

В этом варианте индекс последнего вхождения символа точки в строку присваивается

переменной idx. Данное значение затем используется в методе Substring().

Благодаря этому исключается необходимость дважды искать символ точки в строке.

**Объединение двух последовательностей с помощью оператора join**

Когда приходится иметь дело с базами данных, то зачастую требуется формировать

последовательность, увязывающую данные из разных источников. Например,

в Интернет-магазине может быть организована одна база данных, связывающая наименование

товара с его порядковым номером, и другая база данных, связывающая порядковый

номер товара с состоянием его запасов на складе. В подобной ситуации может

возникнуть потребность составить список, в котором состояние запасов товаров на

складе отображается по их наименованию, а не порядковому номеру. Для этой цели

придется каким-то образом *"*увязать*"* данные из двух разных источников (баз данных).

И это нетрудно сделать с помощью такого средства LINQ, как оператор join.

Ниже приведена общая форма оператора join (совместно с оператором from).

from переменная\_диапазона\_А in источник\_данных\_А

join переменная\_диапазона\_В in источник\_данных\_В

on переменная\_диапазона\_А. свойство equals переменная\_диапазона\_В. свойство

Применяя оператор join, следует иметь в виду, что каждый источник должен содержать

общие данные, которые можно сравнивать. Поэтому в приведенной выше

форме этого оператора *источник\_данных\_А* и *источник\_данных\_В* должны иметь

нечто общее, что подлежит сравнению. Сравниваемые элементы данных указываются

в части on данного оператора. Поэтому если *переменная\_диапазона\_А.свойство*

и *переменная\_диапазона\_А.свойство* равны, то эти элементы данных *"*увязываются*"*

успешно. По существу, оператор join выполняет роль своеобразного фильтра,

отбирая только те элементы данных, которые имеют общее значение.

Как правило, оператор join возвращает последовательность, состоящую из данных,

полученных из двух источников. Следовательно, с помощью оператора join можно

сформировать новый список, состоящий из элементов, полученных из двух разных

источников данных. Это дает возможность организовать данные по-новому.

Ниже приведена программа, в которой создается класс Item, инкапсулирующий

наименование товара и его порядковый номер. Затем в этой программе создается

еще один класс InStockStatus, связывающий порядковый номер товара с булевым

свойством, которое указывает на наличие или отсутствие товара на складе. И наконец,

в данной программе создается класс Temp с двумя полями: строковым (string) и

булевым (bool). В объектах этого класса будут храниться результаты запроса. В этом

запросе оператор join используется для получения списка, в котором наименование

товара связывается с состоянием его запасов на складе.

(***glava19\_14***)

class Item

{

public string Name { get; set; }

public int ItemNumber { get; set; }

public Item(string n, int inum)

{

Name = n;

ItemNumber = inum;

}

}

class InStockStatus

{

public int ItemNumber { get; set; }

public bool InStock { get; set; }

public InStockStatus(int n, bool b)

{

ItemNumber = n;

InStock = b;

}

}

class Temp

{

public string Name { get; set; }

public bool InStock { get; set; }

public Temp(string n, bool b)

{

Name = n;

InStock = b;

}

}

class JoinDemo

{

static void Main()

{

Item[] items =

{

new Item("Kusaki", 1424),

new Item("tiski", 7892),

new Item("Hammer", 8534),

new Item("Saw", 6411)

};

InStockStatus[] statusList =

{

new InStockStatus(1424, true),

new InStockStatus(7892, false),

new InStockStatus(8534, true),

new InStockStatus(6411, true)

};

//make query, it merge objects of class Item

//and InStockStatus for bulding list of goods

//and they qntity on stock. Take a look at forming

//order of Temp objects

var inStockList = from item in items

join entry in statusList

on item.ItemNumber equals entry.ItemNumber

select new Temp(item.Name, entry.InStock);

Console.WriteLine("Goods\t\tOn stock\n");

foreach (Temp t in inStockList)

Console.WriteLine("{0}\t\t{1}", t.Name, t.InStock);

}

}

Для того чтобы стал понятнее принцип действия оператора join, рассмотрим каждую

строку запроса из приведенной выше программы по порядку. Этот запрос начинается,

как обычно, со следующего оператора from.

var inStockList = from item in items

В этом операторе указывается переменная диапазона item для источника данных

items, который представляет собой массив объектов класса Item. В классе Item инкапсулируются

наименование товара и порядковый номер товара, хранящегося на

складе.

Далее следует приведенный ниже оператор join.

join entry in statusList

on item.ItemNumber equals entry.ItemNumber

В этом операторе указывается переменная диапазона entry для источника данных

statusList, который представляет собой массив объектов класса InStockStatus,

связывающего порядковый номер товара с состоянием его запасов на складе. Следовательно,

у массивов items и statusList имеется общее свойство: порядковый номер

товара. Именно это свойство используется в части on/equals оператора join для

описания связи, по которой из двух разных источников данных выбираются наименования

товаров, когда их порядковые номера совпадают.

И наконец, оператор select возвращает объект класса Temp, содержащий наименование

товара и состояние его запасов на складе.

select new Temp(item.Name, entry.InStock);

Таким образом, последовательность результатов, получаемая по данному запросу,

состоит из объектов типа Temp.

Рассмотренный здесь пример применения оператора join довольно прост. Тем не

менее этот оператор поддерживает и более сложные операции с источниками данных.

Например, используя совместно операторы into и join, можно создать *групповое*

*объединение,* чтобы получить результат, состоящий из первой последовательности и

группы всех совпадающих элементов из второй последовательности. (Соответствующий

пример будет приведен далее в этой главе.) Как правило, время и усилия, затраченные

на полное освоение оператора join, окупаются сторицей, поскольку он дает

возможность распознавать данные во время выполнения программы. Это очень ценная

возможность. Но она становится еще ценнее, если используются анонимные типы, о

которых речь пойдет в следующем разделе.

**Анонимные типы**

В C# предоставляется средство, называемое *анонимным типом* и связанное непосредственно

с LINQ. Как подразумевает само название, анонимный тип представляет

собой класс, не имеющий имени. Его основное назначение состоит в создании объекта,

возвращаемого оператором select. Результатом запроса нередко оказывается последовательность

объектов, которые составляются из членов, полученных из двух или

более источников данных (как, например, в операторе join), или же включают в себя

подмножество членов из одного источника данных. Но в любом случае тип возвращаемого

объекта зачастую требуется только в самом запросе и не используется в остальной

части программы. Благодаря анонимному типу в подобных случаях отпадает необходимость

объявлять класс, который предназначается только для хранения результата

запроса.

Анонимный тип объявляется с помощью следующей общей формы:

new { имя\_А = значение\_А, имя\_В = значение\_В, ... }

где имена обозначают идентификаторы, которые преобразуются в свойства, доступные

только для чтения и инициализируемые значениями, как в приведенном ниже

примере.

new { Count = 10, Max = 100, Min = 0 }

данном примере создается класс с тремя открытыми только для чтения свойствами:

Count, Мах и Min, которым присваиваются значения 10, 100 и 0 соответственно.

К этим свойствам можно обращаться по имени из другого кода. Следует заметить, что

в анонимном типе используются инициализаторы объектов для установки их полей

и свойств в исходное состояние.

(Напомним, что у конструкторов такое же имя, как и у их класса. Но у анонимного

класса нет имени, а значит, и нет возможности вызвать его конструктор.)

Итак, у анонимного типа нет имени, и поэтому для обращения к нему приходится

использовать неявно типизированную переменную. Это дает компилятору возможность

вывести надлежащий тип. В приведенном ниже примере объявляется переменная

myOb, которой присваивается ссылка на объект, создаваемый в выражении анонимного

типа.

var myOb = new { Count = 10, Max = 100, Min = 0 }

Напомним, что при создании объекта анонимного типа указываемые идентификаторы

становятся свойствами, открытыми ***только для чтения***. Поэтому их можно использовать

в других частях кода.

Для того чтобы стало более понятным особое назначение анонимных типов, рассмотрим

переделанную версию программы из предыдущего раздела, посвященного

оператору join. Напомним, что в этой программе класс Temp требовался для инкапсуляции

результата, возвращаемого оператором join. Благодаря применению

анонимного типа необходимость в этом классе-заполнителе отпадает, а исходный код

программы становится менее громоздким. Результат выполнения программы при

этом не меняется.

(***glava19\_14.1***)

class Item

{

public string Name { get; set; }

public int ItemNumber { get; set; }

public Item(string n, int inum)

{

Name = n;

ItemNumber = inum;

}

}

class InStockStatus

{

public int ItemNumber { get; set; }

public bool InStock { get; set; }

public InStockStatus(int n, bool b)

{

ItemNumber = n;

InStock = b;

}

}

class AnonTypeDemo

{

static void Main()

{

Item[] items =

{

new Item("Kusaki", 1424),

new Item("tiski", 7892),

new Item("Hammer", 8534),

new Item("Saw", 6411)

};

InStockStatus[] statusList =

{

new InStockStatus(1424, true),

new InStockStatus(7892, false),

new InStockStatus(8534, true),

new InStockStatus(6411, true)

};

//make query, it merge objects of class Item

//and InStockStatus for bulding list of goods

//and they qntity on stock. Take a look at forming

//order of Temp objects

var inStockList = from item in items

join entry in statusList

on item.ItemNumber equals entry.ItemNumber

select new { Name = item.Name, InStock = entry.InStock };

Console.WriteLine("Goods\t\tOn stock\n");

foreach (var t in inStockList)

Console.WriteLine("{0}\t\t{1}", t.Name, t.InStock);

}

}

Обратите особое внимание на следующий оператор select.

select new { Name = item.Name, InStock = entry.InStock };

Он возвращает объект анонимного типа с двумя доступными только для чтения

свойствами: Name и InStock. Этим свойствам присваиваются наименование товара

и состояние его наличия на складе. Благодаря применению анонимного типа необходимость

в упоминавшемся выше классе Temp отпадает.

Прежде чем продолжить изложение, следует отметить еще один заслуживающий

внимания аспект анонимных типов. В некоторых случаях, включая и рассмотренный

выше, синтаксис анонимного типа упрощается благодаря применению *инициализатора*

*проекции.* В данном случае просто указывается имя самого инициализатора. Это имя

автоматически становится именем свойства. В качестве примера ниже приведен другой

вариант оператора select из предыдущей программы.

select new { item.Name, entry.InStock };

В данном примере имена свойств остаются такими же, как и прежде, а компилятор

автоматически *"*проецирует*"* идентификаторы Name и InStock, превращая их в свойства

анонимного типа. Этим свойствам присваиваются прежние значения, обозначаемые

item.Name и entry.InStock соответственно.

**Создание группового объединения**

Как пояснялось ранее, оператор into можно использовать вместе с оператором

join для создания *группового объединения,* образующего последовательность, в которой

каждый результат состоит из элементов данных из первой последовательности и группы

всех совпадающих элементов из второй последовательности. Примеры группового

объединения не приводились выше потому, что в этом объединении нередко применяется

анонимный тип. Но теперь, когда представлены анонимные типы, можно обратиться

к простому примеру группового объединения.

В приведенном ниже примере программы групповое объединение используется

для составления списка, в котором различные транспортные средства (автомашины,

суда и самолеты) организованы по общим для них категориям транспорта: наземного, морского, воздушного и речного. В этой программе сначала создается класс

Transport, связывающий вид транспорта с его классификацией. Затем в методе

Main() формируются две входные последовательности. Первая из них представляет

собой массив символьных строк, содержащих названия общих категорий транспорта:

наземного, морского, воздушного и речного, а вторая — массив объектов типа

Transport, инкапсулирующих различные транспортные средства. Полученное в итоге

групповое объединение используется для составления списка транспортных средств,

организованных по соответствующим категориям

(***glava19\_15***)

//this class connect types of transport

//for example train, with common classification

// of transport: land, air, river, sea

class Transport

{

public string Name { get; set; }

public string How { get; set; }

public Transport(string n, string h)

{

Name = n;

How = h;

}

}

class GroupJoinDemo

{

static void Main()

{

//array of clas. type of transport

string[] travelTypes =

{

"Air", "Land", "Sea", "River"

};

//array types transport

Transport[] transports =

{

new Transport("bycycle", "Land"),

new Transport("aerostat", "Air"),

new Transport("boat", "River"),

new Transport("plane", "Air"),

new Transport("canoe", "River"),

new Transport("biplane", "Air"),

new Transport("Car", "Land"),

new Transport("fary", "Sea"),

new Transport("train", "Land")

};

var byHow = from how in travelTypes

join trans in transports

on how equals trans.How

into lst

select new { How = how, Tlist = lst };

foreach(var t in byHow)

{

Console.WriteLine("In category <{0} transport>: ", t.How);

foreach (var m in t.Tlist) Console.WriteLine(" " + m.Name);

Console.WriteLine();

}

}

}

Главной частью данной программы, безусловно, является следующий запрос.

var byHow = from how in travelTypes

join trans in transports

on how equals trans.How

into lst

select new { How = how, Tlist = lst };

Этот запрос формируется следующим образом. В операторе from используется

переменная диапазона how для охвата всего массива travelTypes. Напомним, что

массив travelTypes содержит названия общих категорий транспорта: воздушного,

наземного, морского и речного. Каждый вид транспорта объединяется в операторе

join со своей категорией. Например, велосипед, автомашина и поезд объединяются

с наземным транспортом. Но благодаря оператору into для каждой категории транспорта

в операторе join составляется список видов транспорта, относящихся к данной

категории. Этот список сохраняется в переменной lst. И наконец, оператор select

возвращает объект анонимного, типа, инкапсулирующий каждое значение переменной how (категории транспорта) вместе со списком видов транспорта.