

**УНИВЕРЗИТЕТ „ГОЦЕ ДЕЛЧЕВ“ - ШТИП
ФАКУЛТЕТ ЗА ИНФОРМАТИКА**

ISSN 1857- 8691

**ГОДИШЕН ЗБОРНИК
2012
YEARBOOK
2012**

ГОДИНА 1

VOLUME I

**GOCE DELCEV UNIVERSITY - STIP
FACULTY OF COMPUTER SCIENCE**

УНИВЕРЗИТЕТ „ГОЦЕ ДЕЛЧЕВ“ – ШТИП
ФАКУЛТЕТ ЗА ИНФОРМАТИКА



ГОДИШЕН ЗБОРНИК
2012
YEARBOOK
2012

ГОДИНА 1

МАРТ, 2013

VOLUME I

GOCE DELCEV UNIVERSITY – STIP
FACULTY OF COMPUTER SCIENCE

**ГОДИШЕН ЗБОРНИК
ФАКУЛТЕТ ЗА ИНФОРМАТИКА
YEARBOOK
FACULTY OF COMPUTER SCIENCE**

За издавачот:

Проф д-р Владо Гичев

Издавачки совет

Проф. д-р Саша Митрев
Проф. д-р Лилјана Колева - Гудева
Проф. д-р Владо Гичев
Проф. д-р Цвета Мартиновска
Проф. д-р Татајана Атанасова - Пачемска
Доц. д-р Зоран Здравев
Доц. д-р Александра Милева
Доц. д-р Сашо Коцески
Доц. д-р Наташа Коцеска
Доц. д-р Зоран Утковски
Доц. д-р Игор Стојановиќ
Доц. д-р Благој Делипетров

Редакциски одбор

Проф. д-р Цвета Мартиновска
Проф. д-р Татајана Атанасова - Пачемска
Доц. д-р Наташа Коцеска
Доц. д-р Зоран Утковски
Доц. д-р Игор Стојановиќ
Доц. д-р Александра Милева
Доц. д-р Зоран Здравев

Главен и одговорен уредник

Доц. д-р Зоран Здравев

Јазично уредување

Даница Гавриловска - Атанасовска
(македонски јазик)
Павлинка Павлова-Митева
(англиски јазик)

Техничко уредување

Славе Димитров
Благој Михов

Редакција и администрација
Универзитет „Гоце Делчев“ - Штип
Факултет за информатика
ул. „Крсте Мисирков“ 10-А
п. фах 201, 2000 Штип
Р. Македонија

Editorial board

Prof. Saša Mitrev, Ph.D.
Prof. Liljana Koleva - Gudeva, Ph.D.
Prof. Vlado Gicev, Ph.D.
Prof. Cveta Martinovska, Ph.D.
Prof. Tatjana Atanasova - Pacemska, Ph.D.
Ass. Prof. Zoran Zdravev, Ph.D.
Ass. Prof. Aleksandra Mileva, Ph.D.
Ass. Prof. Saso Koceski, Ph.D.
Ass. Prof. Natasa Koceska, Ph.D.
Ass. Prof. Zoran Utkovski, Ph.D.
Ass. Prof. Igor Stojanovik, Ph.D.
Ass. Prof. Blagoj Delipetrov, Ph.D.

Editorial staff

Prof. Cveta Martinovska, Ph.D.
Prof. Tatjana Atanasova - Pacemska, Ph.D.
Ass. Prof. Natasa Koceska, Ph.D.
Ass. Prof. Zoran Utkovski, Ph.D.
Ass. Prof. Igor Stojanovik, Ph.D.
Ass. Prof. Aleksandra Mileva, Ph.D.
Ass. Prof. Zoran Zdravev, Ph.D.

Managing/ Editor in chief

Ass. Prof. Zoran Zdravev, Ph.D.

Language editor

Danica Gavrilovska-Atanasovska
(macedonian language)
Pavlinka Pavlova-Miteva
(english language)

Technical editor

Slave Dimitrov
Blagoj Mihov

Address of the editorial office

Goce Delcev University – Stip
Faculty of Computer Science
Krstе Misirkov 10-A
PO box 201, 2000 Stip,
R. of Macedonia

СОДРЖИНА
CONTENT

DEVELOPING CLOUD COMPUTING’S NOVEL COMPUTATIONAL METHODS FOR IMPROVING LONG-TERM WEATHER GLOBAL FORECAST Zubov Dmytro	7
PERVASIVE ALERT SYSTEM FOR FALL DETECTION BASED ON MOBILE PHONES Kire Serafimov, Natasa Koceska	17
ESTABLISHMENT OF A HEALTHCARE INFORMATION SYSTEM Alexandar Kostadinovski, Drasko Atanasoski	26
TIME COMPLEXITY IMPROVEMENT OF THE FIRST PROCESSING STAGE OF THE INTELLIGENT CLUSTERING Done Stojanov, Cveta Martinovska	36
MOODLE AS A TEACHING TOOLS IN MATHEMATICS-CASE STUDY IN UNIVERSITY “GOCE DELCEV” STIP Tatjana Atanasova-Pacemska, Sanja Pacemska, Biljana Zlatanovska	45
TOURISM RECOMMENDATION SYSTEMS: ANALYTICAL APPROACH Biljana Petrevska, Marija Pupinoska-Gogova, Zoran Stamenov	57
CLOUD COMPUTING APPLICATION FOR WATER RESOURCES MODELING AND OPTIMIZATION Blagoj Delipetrev	66
IMPROVING THE SECURITY OF CLOUD-BASED ERP SYSTEMS Gjorgji Gicev, Ivana Atanasova, Jovan Pehcevski	77
USING OF THE MOORE-PENROSE INVERSE MATRIX IN IMAGE RESTORATION Igor Stojanovic, Predrag Stanimirovic, Marko Miladinovic	88
THE INFLUENCE OF THE BUSINESS INTELLIGENCE ON THE BUSINESS PERFORMANCE MANAGEMENT Ljupco Davcev, Ana Ljubotenska	99
LINQ TO OBJECTS SUPPORTED JOINING DATA Mariana Goranova	109
GLOBALIZATION, INFORMATION TECHNOLOGY AND NEW DIGITAL ECONOMIC LANDSCAPE Riste Temjanovski	120

WEB БАЗИРАН СОФТВЕР ЗА SCADA АПЛИКАЦИИ INTEGRAXOR Марјан Стоилов, Василија Шарац	130
SECURITY IN COMPUTER NETWORKS FROM THE PERSPECTIVE OF ACCESS CONTROL Saso Gelev, Jasminka Sukarovska-Kostadinovska	139
FREQUENCY DISTRIBUTION OF LETTERS, BIGRAMS AND TRIGRAMS IN THE MACEDONIAN LANGUAGE Aleksandra Mileva, Stojanče Panov, Vesna Dimitrova	149
TOWARDS A GENERIC METADATA MODELING Pavel Saratchev	161
ECONOMIC VALUE OF INFORMATION SYSTEMS IN PRODUCTION PROCESSES Aleksandar Krstev, Zoran Zdravev	175
TUNING PID CONTROLLING PARAMETERS FOR DC MOTOR SPEED REGULATION Done Stojanov	185
COMPARISON OF THE PERFORMANCE OF THE ARTIFICIAL BOUNDARIES P3 AND P4 OF STACEY Zoran Zlatev, Vasko Kokalanov, Aleksandra Risteska	192
CORRESPONDENCE BETWEEN ONE-PARAMETER GROUP OF LINEAR TRANSFORMATIONS AND LINEAR DIFFERENTIAL EQUATIONS THAT DESCRIBE DYNAMICAL SYSTEMS Marija Miteva, Limonka Lazarova	200
THE BLACK-SCHOLES MODEL AND VALUATION OF THE EUROPEAN CALL OPTION Limonka Lazarova, Marija Miteva, Natasa Stojkovic	209
BITCOIN SCHEMES- INOVATION OR A THREAT TO FINANCIAL STABILITY? Violeta Madzova	221
JAVA IDEs FOR EASILY LEARNING AND UNDERSTANDING OBJECT ORIENTED PROGRAMMING Aleksandra Stojanova, Natasha Stojkovic, Dusan Bikov	232
STUDENTS' KNOWLEDGE TEST CONTROL – METHODS AND RESULTS' INTERPRETATION Ludmila Stoyanova, Daniela Minkovska	241

WEB SERVICE FOR AMBIGUOUS TRANSLITERATION OF FULL SENTENCES FROM LATIN TO CYRILLIC ALPHABET	
Stojance Spasov, Zoran Zdravev	252
ON THE APPLICATION OF KEEDWELL CROSS INVERSE QUASIGROUP TO CRYPTOGRAPHY	
Jaiyéolá Tèmitopé Gboláhàn	264

LINQ TO OBJECTS SUPPORTED JOINING DATA

Mariana Goranova^{1,*}

^{1,*}*Technical University of Sofia, mgor@tu-sofia.bg*

Abstract: Joins have been studied as a key operations in multiple application domains. This paper focuses on the study of joins as a first-class LINQ operators and their implementation as integrated component of the query processing. The join methods provided in the LINQ perform inner join, left outer join, and cross join. Our goal is to provide join operations, similar to SQL statements in the databases. We describe an efficient implementation of the following join operators: inner join, left outer join, right inner join, full outer join, left excluding join, right excluding join, full outer excluding join, and cross join. A simple example is used to present the potentialities of LINQ technology to solve the problem with the object-relational mapping.

Keywords: LINQ query, inner join, outer join, cross join, C# programming language

19 Introduction

Concept of joining is a key concept of accessing data. From SQL and relational point of view, almost every query requires joining data. SQL languages have powerful join capabilities. They support different types of joins, including inner joins, outer joins, and cross joins [1].

Language Integrated Query (LINQ) is the technology that addresses the problems between programming languages and databases. LINQ provides a uniform, object-oriented way to access data from heterogeneous sources and simplifies the interaction between object-oriented programming and relational data.

In this paper, we implement the set of join operators in LINQ that use syntax similar to SQL [3,4]. The main contributions include the study of joins as a first-class LINQ operators and their implementation as integrated component of the LINQ query processing.

The rest of the paper is organized as follows. Section 2 discusses the background. Section 3 presents the different types of SQL-like joins and syntax using examples in C#. Section 4 presents the conclusions and directions for future research.

20 Background

LINQ is a programming model that introduces queries as a first-class concept into any Microsoft .NET Framework Language [5]. LINQ provides a uniform way to access and manage data in the program, keeping existing heterogeneous data structures, regardless of their physical representation – the data source might be a graph of objects in-memory, relational table or XML file [2]. The software developers use the same query syntax over all different data access models.

The LINQ architecture is shown in Figure 1. The different data sources are as follows: LINQ to Objects, LINQ to Datasets, LINQ to SQL, LINQ to Entities, and LINQ to XML.

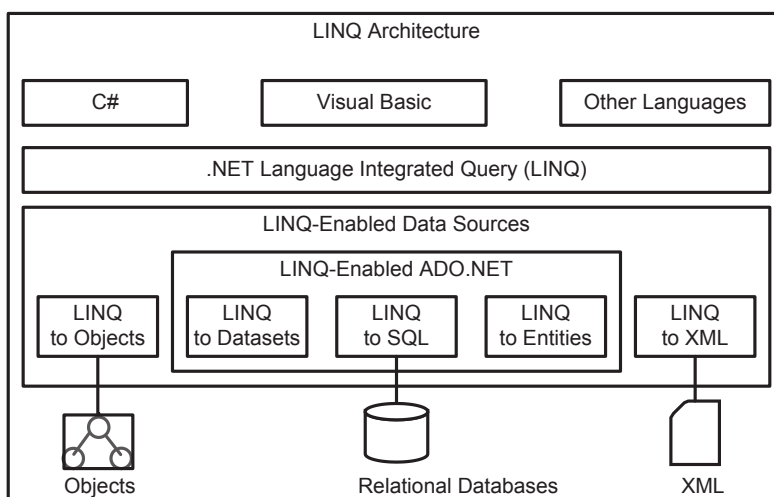


Figure 1 LINQ architecture

LINQ query is a set of operations on instances of some classes. The declarative description of operations on data using syntax very similar to SQL is the most important feature of LINQ because it enhances programmers' productivity.

21 Join operators

Joining data sets is the process of linking two data sources through a common attribute. Joining is an important operation in queries. It models a correlation between objects that is not implemented in the object-oriented programming.

7.3 Class model

In the example we will use a data structure that represents information about courses, with enrolls and students. Each course includes many enrolls. The class **Course** has information about **ID** number, **Title** and a reference to a set of **enrolls**. Class **Enroll** has information about **EnrollId**, **Grade** and faculty number **FN** of a student. Class **Student** has a faculty number **FN** and **Name**. The definitions of these types are represented in the Appendix. The data consists of a set of **courses**, each of which has enrolled **students**. The initialized instances are shown in the Appendix.

We assume we have two sets – **students** is on the left and **coursesEnrolls** is on the right. We can join these sets by their common **FN** attributes.

```
var coursesEnrolls = from course in courses
                    from e in course.enroll
                    select e;
```

7.4 Inner join

The inner join is the most common join operator. It creates a result set of elements of the two sets (**students** and **coursesEnrolls**) that match in both sets. The inner join is implemented in the LINQ base library using the **Join** method.

SQL statement:

```
SELECT * FROM STUDENTS  
INNER JOIN COURSESENROLLS  
ON STUDENTS.FN=COURSESENROLLS.FN
```

Query expression syntax:

```
var innerJoin =from x in students  
join y in coursesEnrolls  
on x.FN equals y.FN  
select new { X = x, Y = y};
```

Lambda expression syntax:

```
var innerJoin = students  
.Join(coursesEnrolls,  
x => x.FN, y => y.FN, (x, y) => new {  
x, y })  
.Select(item => new { item.x, item.y });
```

Result:	FN	Name	EnrollId	Grade	FN
	222100	Petia Petrova	2	5	222100
	222101	Julian Emilov	4	6	222101
	222103	Neli Ivanova	6	6	222103
	222104	Ivan Georgiev	9	6	222104

FN values from **students** match with the FN from **coursesEnrolls**. The inner join only returns elements where the two sets intersect.

7.5 Left outer join

The left outer join creates a result set that includes all elements from the left set (**students**) with the matching elements from the right set (**coursesEnrolls**). The left outer join is implemented using the **GroupJoin** method on the left set and then using **from** operator to get the matching elements from the right set, if any exist. If there is not match the right side will contain null using the **DefaultIfEmpty** extension method.

SQL statement:

```
SELECT * FROM STUDENTS  
LEFT OUTER JOIN COURSESENROLLS  
ON STUDENTS.FN=COURSESENROLLS.FN
```

Query expression syntax:	Lambda expression syntax:
<pre>var leftOuterJoin = from x in students join y in coursesEnrolls on x.FN equals y.FN into yG from y1 in yG.DefaultIfEmpty() select new { X = x, Y = y1 == null ? null : y1 };</pre>	<pre>var leftOuterJoin = students .GroupJoin(coursesEnrolls, x => x.FN, y => y.FN, (x, g) => new { x, g }) .SelectMany(y => y.g.DefaultIfEmpty(), (item, y) => new { X=item.x, Y=y });</pre>

Result:	FN	Name	EnrollId	Grade	FN
	222100	Petia Petrova	2	5	222100
	222101	Julian Emilov	4	6	222101
	222102	Anely Borisova	0	0	-1
	222103	Neli Ivanova	6	6	222103
	222104	Ivan Georgiev	9	6	222104
	222105	Mila Ivanova	0	0	-1
	222107	Kristi Kirilova	0	0	-1
	222112	Anton Ivanov	0	0	-1

The result set contains all students but four students have no enrolls associated with them – the return objects by the **Enroll** class are null and the **NullReferenceException** is controlled.

7.6 Right outer join

The right outer join creates a result set that includes all elements from the right set (**coursesEnrolls**) with the matching elements from the left set (**students**). It closely likes to a left outer join with reversed sets.

SQL statement:

```
SELECT * FROM STUDENTS
RIGHT OUTER JOIN COURSESENROLLS
ON STUDENTS.FN=COURSESENROLLS.FN
```

Query expression syntax:	Lambda expression syntax:
<pre>var rightOuterJoin = from y in coursesEnrolls join x in students on y.FN equals x.FN into xG</pre>	<pre>var rightOuterJoin = coursesEnrolls .GroupJoin(students,</pre>

```

from x1 in xG.DefaultIfEmpty() x => x.FN, y => y.FN, (x, g) => new {
select new { X = x1==null ? null x, g }
: x1, .SelectMany(y =>
        Y = y }; y.g.DefaultIfEmpty(),
        (y, item) => new { X = item, Y = y.x });

```

Result:	FN	Name	EnrollId	Grade	FN
	222100	Petia Petrova	2	5	222100
	-1	null	3	6	222201
	222101	Julian Emilov	4	6	222101
	222103	Neli Ivanova	6	6	222103
	-1	null	7	5	222200
	222104	Ivan Georgiev	9	6	222104
	-1	null	10	5	222110
	-1	null	11	4	222111

The result set contains all enrolls where four enrolls have no students associated with them – the return objects by the **StudentI** class are null and the **NullReferenceException** is controlled.

7.7 Full outer join

The full outer join creates a result set that includes all elements from the left set (**students**) and the right set (**coursesEnrolls**) with the matching elements from the both sets where available. If there is not match the missing left/right side will contain null.

SQL statement:

```

SELECT * FROM STUDENTS
FULL OUTER JOIN COURSESENROLLS
ON STUDENTS.FN=COURSESENROLLS.FN

```

Lambda expression syntax:

```

var fullOuterJoin = leftJoin.Union(rightJoin);

```

Result:	FN	Name	EnrollId	Grade	FN
	222100	Petia Petrova	2	5	222100
	222101	Julian Emilov	4	6	222101
	222102	Anely Borisova	0	0	-1
	222103	Neli Ivanova	6	6	222103

222104	Ivan Georgiev	9	6	222104
222105	Mila Ivanova	0	0	-1
222107	Kristi Kirilova	0	0	-1
222112	Anton Ivanov	0	0	-1
-1	null	3	6	222201
-1	null	7	5	222200
-1	null	10	5	222110
-1	null	11	4	222111

7.8 Left excluding join

The left excluding join creates a result set that includes only elements from the left set (**students**) that are not in the right set (**coursesEnrolls**). It performs the left outer join and then excludes the common elements from the right set.

SQL statement:

```
SELECT * FROM STUDENTS
LEFT OUTER JOIN COURSESENROLLS
ON STUDENTS.FN=COURSESENROLLS.FN
WHERE COURSESENROLLS.FN IS NULL
```

Query expression syntax:

```
var leftExcludingJoin = from x in
students
join y in coursesEnrolls
on x.FN equals y.FN into yG
from y1 in yG.DefaultIfEmpty()
where y1 == null
select new { X = x,
            Y = y1 == null ? null
            : y1 };
```

Lambda expression syntax:

```
var leftExcludingJoin1 = students
.GroupJoin(coursesEnrolls,
x => x.FN, y => y.FN, (x, g) => new {
x, g })
.SelectMany(y=>y.g.DefaultIfEmpty(),
(item, y) => new { X = item.x, Y = y })
.Where(y=>y.Y==null);
```

Result:	FN	Name	EnrollId	Grade	FN
	222102	Anely Borisova	0	0	-1
	222106	Mila Ivanova	0	0	-1
	222107	Kristi Kirilova	0	0	-1
	222112	Anton Ivanov	0	0	-1

7.9 Right excluding join

The right excluding join creates a result set that includes only elements from the right set (**coursesEnrolls**) that are not in left set (**students**). It closely likes to a left excluding join with reversed sets.

SQL statement:

```
SELECT * FROM STUDENTS
RIGHT OUTER JOIN COURSESENROLLS
ON STUDENTS.FN=COURSESENROLLS.FN
WHERE STUDENTS.FN IS NULL
```

Query expression syntax:

```
var rightExcludingJoin =
    from y in coursesEnrolls
    join x in students
    on y.FN equals x.FN into xG
    from x1 in xG.DefaultIfEmpty()
    where x1==null
    select new { X = x1 == null ? null :
x1,
                Y = y };
```

Lambda expression syntax:

```
var rightExcludingJoin =
    coursesEnrolls
    .GroupJoin(students,
    x => x.FN, y => y.FN, (x, g) =>
    new { x, g })
    .SelectMany(y =>
    y.g.DefaultIfEmpty(),
    (y, item) => new { X = item, Y = y.x
    })
    .Where(x=>x.X==null);
```

Result:	FN	Name	EnrollId	Grade	FN
	-1	null	3	6	222201
	-1	null	7	5	222200
	-1	null	10	5	222110
	-1	null	11	4	222111

7.10 Full outer excluding join

The full outer excluding join creates a result set that includes unique elements from the left set (**students**) and the right set (**coursesEnrolls**) that do not match. It performs the full outer join and then excludes the common elements from the both sets.

SQL statement:

```
SELECT * FROM STUDENTS
FULL OUTER JOIN COURSESENROLLS
ON STUDENTS.FN=COURSESENROLLS.FN
WHERE STUDENTS.FN IS NULL OR COURSESENROLLS.FN IS NULL
```

Lambda expression syntax:

```
var fullOuterExcludingJoin = leftExcludingJoin.Union(rightExcludingJoin);
```

Result:	FN	Name	EnrollId	Grade	FN
	222102	Anely Borisova	0	0	-1
	222105	Mila Ivanova	0	0	-1
	222107	Kristi Kirilova	0	0	-1
	222112	Anton Ivanov	0	0	-1
	-1	null	3	6	222201
	-1	null	7	5	222200
	-1	null	10	5	222110
	-1	null	11	4	222111

7.11 Cross join

The cross join or Cartesian product creates a result set that includes all possible ordered pairs whose first component is a member of the left set (**students**) and whose second component is a member of the right set (**coursesEnrolls**).

SQL statement:

```
SELECT * FROM STUDENTS
CROSS JOIN COURSESENROLLS
```

Query expression syntax:	Lambda expression syntax:
<pre>var crossJoin = from x in students from y in coursesEnrolls select new { X = x, Y = y};</pre>	<pre>var crossJoin1 = students .SelectMany(x=>coursesEnrolls, (x,y)=>new { X=x, Y=y});</pre>

The result set contains 64 elements – all of the {student,enroll} pairs, even the combinations are not valid.

8 Conclusion and Future Work

We have proposed implementation of join operations using LINQ. These operations implement the behavior of the SQL join operations. We make attempt to solve the gap between the programming languages and the databases. These operations can be used for the purposes of analysis and visualization.

Plans for future work include the study and integration of join strategies as first-class LINQ operators in approaches that support SOA-based scientific data management.

References

- [1] Jeff Atwood, Coding Horror (2007): *A Visual Explanation of SQL Joins*, Last access 10.11.2012, <http://www.codinghorror.com/blog/2007/10/a-visual-explanation-of-sql-joins.html>.
- [2] M. Goranova and L. Stoyanova (2012): *Effective query implementation of scientific data based on LINQ to XML*. Annual Journal of Electronics 6(2), pp. 125-128.
- [3] Juan Francisco Morales Larios (2012): *LinQ Extended Joins*, Last access: 10.11.2012, <http://www.codeproject.com/Articles/488643/LinQ-Extended-Joins>.
- [4] Scott Mitchell (2009): *An Extensive Examination of LINQ: Grouping and Joining Data*, Last access: 10.11.2012, <http://www.4guysfromrolla.com/articles/080509-1.aspx>.
- [5] P. Pialorsi and M. Russo (2010): *Programming Microsoft LINQ in Microsoft .NET Framework 4*, O'Reilly Media, Inc.

Appendix

```
class Course
{
    List<Course> courses = new List<Course>() {
        new Course { ID = 1, Title = "Distributed
        public int ID { get; set; } Systems",
        public string Title { get; set; enroll = new Enroll[] {
        }
        new Enroll {EnrollId=2, Grade=5,
        public Enroll[] enroll; FN=222100},
        public override string new Enroll {EnrollId=3, Grade=6,
        ToString() FN=222201}}},
        {
            new Course { ID = 2, Title = "Computer
            string r = ID + "\t" + TitleGraphics",
        + "\n";
            enroll = new Enroll[] {
            foreach (var e in enroll) new Enroll {EnrollId=4, Grade=6,
            r += e.ToString(); FN=222101},
            return r;
            new Enroll {EnrollId=6, Grade=6,
            }
            FN=222103},
        }
        new Enroll {EnrollId=7, Grade=5,
        FN=222200}}},
class Enroll
{
    new Course { ID = 3, Title = "Software
    public int? EnrollId { get; enroll = new Enroll[] {
    set; } new Enroll {EnrollId=9, Grade=6,
    public int? Grade { get; set;FN=222104},
    }
}
```

```
public int? FN { get; set; }        new Enroll {EnrollId=10, Grade=5,
public override stringFN=222110},
ToString()                        new Enroll {EnrollId=11, Grade=4,
{                                  FN=222111}}}}
return                             };
EnrollId+"\t"+Grade+
"\t"+FN;                           List<Student> students = new List<Student>()
}                                    {
}                                    new Student {FN=222100, Name="Petia
                                   Petrova"},
class Student                      new Student {FN=222101, Name="Julian
{                                  Emilov"},
public int? FN { get; set; }        new Student {FN=222102, Name="Anely
public string Name { get; set; }    Borisova"},
set; }                              new Student {FN=222103, Name="Neli
public override stringIvanova"},
ToString()                          new Student { FN=222104, Name="Ivan
{                                  Georgiev"},
return FN + "\t" + Name;           new Student {FN=222105, Name="Mila
}                                    Ivanova"},
}                                    new Student {FN=222107, Name="Kristi
                                   Kirilova"},
                                   new Student {FN=222112, Name="Anton
                                   Ivanov"}
};
```