УНИВЕРЗИТЕТ „ГОЦЕ ДЕЛЧЕВ” - ШТИП
ФАКУЛТЕТ ЗА ИНФОРМАТИКА

ГОДИШЕН ЗБОРНИК
2012
YEARBOOK
2012

ГОДИНА 1 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

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
Krete Miseirkov 10-A
PO box 201, 2000 Stip,
R. of Macedonia
CONTENTS

DEVELOPING CLOUD COMPUTING’S NOVEL COMPUTATIONAL METHODS FOR IMPROVING LONG-TERM WEATHER GLOBAL FORECAST
Zubov Dmytro .............................................................................................................................. 7

PERVERSIVE ALERT SYSTEM FOR FALL DETECTION BASED ON MOBILE PHONES
Kire Serafimov, Natasa Koceska ........................................................................................................ 17

ESTABLISHEMENT 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
Маријан Стоилов, Василија Шарац .................................................................130

SECURITY IN COMPUTER NETWORKS FROM THE PERSPECTIVE OF ACCESS CONTROL
Saso Gelev, Jasmina Sukarosvka-Kostadinovska ..................................................139

FREQUENCY DISTRIBUTION OF LETTERS, BGRAMS AND TRGRAMS IN THE MACEDONIAN LANGUAGE
Aleksandra Mileva, Stojanche 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
Jaiyéọlá Tèmítọpé Gbọláhàn ................................................................. 264
LINQ TO OBJECTS SUPPORTED JOINING DATA

Mariana Goranova¹,*

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.
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.

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

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.
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

<table>
<thead>
<tr>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>222100</td>
<td>Petia Petrova</td>
<td>2</td>
<td>5</td>
<td>222100</td>
</tr>
<tr>
<td>222101</td>
<td>Julian Emilov</td>
<td>4</td>
<td>6</td>
<td>222101</td>
</tr>
<tr>
<td>222103</td>
<td>Neli Ivanova</td>
<td>6</td>
<td>6</td>
<td>222103</td>
</tr>
<tr>
<td>222104</td>
<td>Ivan Georgiev</td>
<td>9</td>
<td>6</td>
<td>222104</td>
</tr>
</tbody>
</table>

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
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:**

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

**Query expression syntax:**

```csharp
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:**

```csharp
var innerJoin = students.Join(coursesEnrolls,
                               x => x.FN, y => y.FN,
                               (x, y) => new { X = x, Y = y });
```

**Result:**

<table>
<thead>
<tr>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>222100</td>
<td>Petia Petrova</td>
<td>2</td>
<td>5</td>
<td>222100</td>
</tr>
<tr>
<td>222101</td>
<td>Julian Emilov</td>
<td>4</td>
<td>6</td>
<td>222101</td>
</tr>
<tr>
<td>222102</td>
<td>Anely Borisova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222103</td>
<td>Neli Ivanova</td>
<td>6</td>
<td>6</td>
<td>222103</td>
</tr>
<tr>
<td>222104</td>
<td>Ivan Georgiev</td>
<td>9</td>
<td>6</td>
<td>222104</td>
</tr>
<tr>
<td>222105</td>
<td>Mila Ivanova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222107</td>
<td>Kristi Kirilova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222112</td>
<td>Anton Ivanov</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

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

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). It closely likes to a left outer join with reversed sets.

**SQL statement:**

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

**Query expression syntax:**

```csharp
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 };
```

**Lambda expression syntax:**

```csharp
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 });
```

**Result:**

<table>
<thead>
<tr>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>222100</td>
<td>Petia Petrova</td>
<td>2</td>
<td>5</td>
<td>222100</td>
</tr>
<tr>
<td>222101</td>
<td>Julian Emilov</td>
<td>4</td>
<td>6</td>
<td>222101</td>
</tr>
<tr>
<td>222102</td>
<td>Anely Borisova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222103</td>
<td>Neli Ivanova</td>
<td>6</td>
<td>6</td>
<td>222103</td>
</tr>
<tr>
<td>222104</td>
<td>Ivan Georgiev</td>
<td>9</td>
<td>6</td>
<td>222104</td>
</tr>
<tr>
<td>222105</td>
<td>Mila Ivanova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222107</td>
<td>Kristi Kirilova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222112</td>
<td>Anton Ivanov</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

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:**

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

**Query expression syntax:**

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

**Lambda expression syntax:**

```csharp
var rightOuterJoin = coursesEnrolls.GroupJoin(students,
                                              y => y.FN, x => x.FN,
                                              (y, g) => new { X=y, Y=g });
```
from x1 in xG.DefaultIfEmpty() select new { X = x1 == null ? null : x1, Y = y }; x => x.FN, y => y.FN, (x, g) => new { x, g }); .SelectMany(y => y.g.DefaultIfEmpty(), (y, item) => new { X = item, Y = y.x });

<table>
<thead>
<tr>
<th>Result:</th>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>222100</td>
<td>Petia Petrova</td>
<td>2</td>
<td>5</td>
<td>222100</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>3</td>
<td>6</td>
<td>222201</td>
<td></td>
</tr>
<tr>
<td>222101</td>
<td>Julian Emilov</td>
<td>4</td>
<td>6</td>
<td>222101</td>
<td></td>
</tr>
<tr>
<td>222103</td>
<td>Neli Ivanova</td>
<td>6</td>
<td>6</td>
<td>222103</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>7</td>
<td>5</td>
<td>222200</td>
<td></td>
</tr>
<tr>
<td>222104</td>
<td>Ivan Georgiev</td>
<td>9</td>
<td>6</td>
<td>222104</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>10</td>
<td>5</td>
<td>222110</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>11</td>
<td>4</td>
<td>222111</td>
<td></td>
</tr>
</tbody>
</table>

The result set contains all enrolls where four enrolls have no students associated with them – the return objects by the Studentl 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);

<table>
<thead>
<tr>
<th>Result:</th>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>222100</td>
<td>Petia Petrova</td>
<td>2</td>
<td>5</td>
<td>222100</td>
<td></td>
</tr>
<tr>
<td>222101</td>
<td>Julian Emilov</td>
<td>4</td>
<td>6</td>
<td>222101</td>
<td></td>
</tr>
<tr>
<td>222102</td>
<td>Anely Borisova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>222103</td>
<td>Neli Ivanova</td>
<td>6</td>
<td>6</td>
<td>222103</td>
<td></td>
</tr>
</tbody>
</table>
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:**

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

**Query expression syntax:**

```csharp
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:**

```csharp
var leftExcludingJoinl =
    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:**

<table>
<thead>
<tr>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>222102</td>
<td>Anely Borisova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222106</td>
<td>Mila Ivanova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222107</td>
<td>Kristi Kirilova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222112</td>
<td>Anton Ivanov</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>
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`

<table>
<thead>
<tr>
<th>Result</th>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>null</td>
<td>3</td>
<td>6</td>
<td>222201</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>7</td>
<td>5</td>
<td>222200</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>10</td>
<td>5</td>
<td>222110</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>11</td>
<td>4</td>
<td>222111</td>
<td></td>
</tr>
</tbody>
</table>

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);`
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
```

Lambda 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 };
```

```
var rightExludingJoin =
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:
<table>
<thead>
<tr>
<th>FN</th>
<th>Name</th>
<th>EnrollId</th>
<th>Grade</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>222102</td>
<td>Anely Borisova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222105</td>
<td>Mila Ivanova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222107</td>
<td>Kristi Kirilova</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>222112</td>
<td>Anton Ivanov</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>3</td>
<td>6</td>
<td>222201</td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>7</td>
<td>5</td>
<td>222200</td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>10</td>
<td>5</td>
<td>222110</td>
</tr>
<tr>
<td>-1</td>
<td>null</td>
<td>11</td>
<td>4</td>
<td>222111</td>
</tr>
</tbody>
</table>

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);
```

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
```

Lambda expression syntax:
```
var crossJoinl =
    students
        .SelectMany(x=>coursesEnrolls,
            (x,y)=>
                new { X=x, Y=y });
```

```
var crossJoin =
    from x in students
    from y in coursesEnrolls
    select new { X = x, Y = y };
```

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

Appendix
class Course
{  
    public int ID { get; set; }  
    public string Title { get; set; }  
    public Enroll[] enroll;  
    public override string ToString()
    {
        string r = ID + "\t" + Title + "\n";
        foreach (var e in enroll)
        {
            return r + e.ToString();
        }
    }  
    List<Course> courses = new List<Course>()
    {
        new Course {  ID = 1, Title = "Distributed Systems", enroll = new Enroll[]
        {
            new Enroll {EnrollId=2, Grade=5, FN=222100},
        }  
        new Course {  ID = 2, Title = "Computer Graphics", enroll = new Enroll[]
        {
            new Enroll {EnrollId=4, Grade=6, FN=222101},
            new Enroll {EnrollId=6, Grade=6, FN=222103},
            new Enroll {EnrollId=7, Grade=5, FN=222200},
        }  
        new Course {  ID = 3, Title = "Software Engineering", enroll = new Enroll[]
        {
            new Enroll {EnrollId=9, Grade=6, FN=222104},
        }  
    }  
}
public int? FN { get; set; }  
new Enroll {EnrollId=10, Grade=5, 
FN=222110},
new Enroll {EnrollId=11, Grade=4, 
FN=222111}}
};
}
}
class Student{
    public int? FN { get; set; }
    public string Name { get; set; }
    public override string ToString()
    {
        return FN + "\t" + Name;
    }
}
List<Student> students = new List<Student>()
{
    new Student {FN=222100, Name="Petia Petrova"},
    new Student {FN=222101, Name="Julian Emilov"},
    new Student {FN=222102, Name="Anely Borisova"},
    new Student {FN=222103, Name="Neli Ivanova"},
    new Student { FN=222104, Name="Ivan Georgiev"},
    new Student {FN=222105, Name="Mila Ivanova"},
    new Student {FN=222107, Name="Kristi Kirilova"},
    new Student {FN=222112, Name="Anton Ivanov"}
};