Methods of Integrating Programming Languages and Databases

Đorđe Stojisavljević, Biljana Radulović, Živoslav Adamović

***Abstract*— The development of software engineering in the last decade has imposed the issue of integrating programming languages and databases, which has opened many discussions that have pointed to a number of problems and limitations of this type of integration. This article describes different methods of integrating programming languages and databases through analysing system performance of an example model.**

***Keywords*— database integration, LINQ, programming languages, system performances**

I. Introduction

The backbone of an information system (IS) is the database. A database is any logically coherent collection of data organized for storage and retrieval by computers, as a single, possibly large, repository of data that can be used simultaneously by multi-users [1].

Users interact with the database through a number of application programs that are used to create and maintain the database and to generate information. The application programs may be written in some programming language (C++, C#, Java) or in some higher-level fourth-generation language.

Programmers forsake type safety and reduce the modularity of their programs for one reason: performance. Network latency so dominates program performance that a programmer must reorganize code manually to communicate with the database as little as possible. The database stores so much data that a programmer must provide it with as much information as possible, to enable the database to organize and execute an efficient search [2].

Software industry struggles to interface programming languages and databases. Applications that access databases are awkward to design and develop. Careful optimizations are often needed to attain good performance, resulting in programs that are difficult to maintain and evolve.

Because of the gap between programming languages and databases, some researchers think that databases are not really necessary, and can be replaced with File System which is one of the earliest ways of managing data. A File System as a collection of raw data files stored in the hard-drive has not provided consistence, durability and security. On the other hand, databases are self-consistent at any instant in time, they provide isolated transactions and security.

In this paper we discuss issues that arise on the boundary between programming languages and databases, and analyse different methods of integrating programming languages and databases from the aspect of system performances.

Manuscript received Oct. 24, 2015. This work was supported in part by the Techincal Faculty "Mihajlo Pupin" Zrenjanin.

B.Sc. Đorđe Stojisavljević is with the Computer Center of University of Banja Luka, 78000 Banja Luka, Republic of Srpska, Bosnia and Herzegovina

Ph.D. Biljana Radulović was with Techical Faculty "Mihajlo Pupin" of University of Novi Sad, 23000 Zrenjanin, Ph.D. Živoslav Adamović was with Techical Faculty "Mihajlo Pupin" of University of Novi Sad, 23000 Zrenjanin, Searbia

II. Related Work

In 1987, Atkinson & Buneman [3] reviewed early work on integrating programming languages and databases; their focus on creating a clean, uniform programming model for persistent data provided a framework for later research.

Same year, Bloom and Zdonik [4] identified cultural and technical differences, including the handling of consistency, triggers, optimization, and data scaling.

Cook & Ibrahim [5] discussed problems of integrating programming languages and databases in their paper. The primary contribution of their paper is a better understanding of impedance mismatch.

Within the past decade significant progress has been made. New solutions are proposed every year, but none has yet proven fully satisfactory.

Benjamin A. Wiedermann in his Ph.D. dissertation [2] discussed integrating programming languagess and databases via program analyses and language design, which was the most important contribution in this field.

III. Problems of Integrating Programming Languages and Databases

A. Typing

Difficulty in aligning types between programming languages and databases is traditionally viewed as a key problem. Primitive types in a programming language typically do not correspond to the types in a database, and usually primitive types differ between databases.

B. Optimization

Most data-intensive applications handle large amounts of data. Therefore, it is important for the application to optimize data access. An appropriate query strategy may often be orders of magnitude faster than a naive query strategy [6].

C. Performance

The goal of performance tuning is to minimize the response time for each query and to maximize the throughput of the entire database server by reducing network traffic, disk I/O, and CPU

time. This goal is achieved through understanding application requirements, the logical and physical structure of the data, and tradeoffs between conflicting uses of the database.

IV. Research model

In our research we used a model of Faculty Information System (FIS) for testing system performance. ER diagram of a FIS model is shown in Fig. 1.

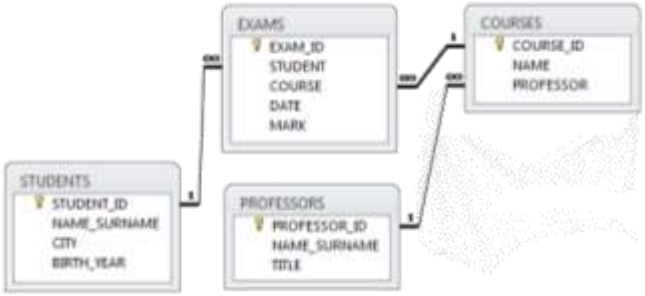

Fig. 1. ER diagram of a Faculty Information System model

The FIS model was designed to record pieces of information about: students, professors, courses and exams. With those pieces of information FIS model can provide four main data operations: reading, writing, searching and deleting records.

For testing system performance purposes, the FIS model was implemented in C# programming language on three different projects:

1. FIS based on File System,
2. FIS based on ADO.NET framework,
3. FIS based on LINQ.

Each project represents one method of integrating programming language and database implementation.

For each data operation, we measured execution time ten times for different number of records in the database/file. Applying statistical tests we wanted to answer these questions:

- Does the number of records in the database/file affect the execution time of an operation?
- Does the number of records highly increase the database/file size?
- Which method gives the best performance results?

V. Methods of Integrating Programming Languages and Databases

A. File System

While a database is generally used for storing related, structured data, with well-defined data formats, a file system is more unstructured data store for storing arbitrary, probably unrelated data.

Conventionally, the data were stored and processed using a traditional file system. In these traditional file systems, each file is independent of other file, and data in different files can be integrated only by writing individual program for each application. The data and the application programs that use the data are so arranged that any change to the data requires modifying all the programs that use the data. This is because each file is hard-coded with specific information like data type, data size etc. Sometimes it is not even possible to identify all the programs using that data [7].

Programming languages have a built-in libraries containing all functions needed for working with files (opening and closing the file, reading from the file, writing to the file, etc). We could say that programming languages and the file system are fully integrated. That is a big advantage of s file system.

In our FIS model based on the file system, we created four binary files. Each file represents an entity: Students, Professors, Courses and Exams. Then we tested this model measuring execution time in milliseconds of each operation for different number of records in a file. After 15 measurements, we found average execution time. Results are shown in Table I.

TABLE I: Execution time of a basic data operations [ms]

Operation	1 record	100 records	500 records	1000 records
Writing	0,3510	0,1470	0,1610	0,2240
Reading	9,7350	9,4170	7,3490	8,2810
Deleting	1,4770	17,581	61,768	86,739
Searching	1,5280	9,1530	37,598	61,765

There was no significant difference in the execution time of writing one record to file (M=1.02, SD=0.92) and writing thousand records to file (M=0.85, SD=0.74) conditions; t(25)=0.50, p=0.62.

There was no significant difference in the execution time of reading one record from file (M=9.44, SD=0.61) and reading thousand records from file (M=8.84, SD=0.37) conditions; t(20)=3.11, p=0.01.

There was a significant difference in the execution time of deleting one record from file (M=2.26, SD=0.52) and deleting thousand records from file (M=82.76, SD=1.12) conditions; t(25)=-211.41, p=3.48E-42.

There was a significant difference in the execution time of searching one record in file (M=1.76, SD=0.43) and searching thousand records in file (M=61.58, SD=0.25) conditions; t(21)=-437.99, p=4.44E-43.

Except the execution time, file size plays an important role in system performance. The size of each file is shown in Table II, as the overall size.

TABLE II: File sizes [KB]

File	1 record	100 records	500 records	1000 records
studens	0,22	21,60	107,00	215,00
professors	0,21	20,70	103,00	202,00
courses	0,35	33,40	167,00	334,00
exams	0,64	63,10	312,00	623,00
Overall	1,42	138,80	689,00	1374,00

There was a significant difference between the size of file with one record (M=0.36, SD=0.20) and size of file with thousand records (M=344, SD=195.57) conditions; t(3)=-3.51, p=0.04.

B. ADO.NET Framework

The .NET Framework has introduced a new set of database classes designed for loosely coupled, distributed architectures. These classes are referred to as ADO.NET [8].

ADO.NET uses the same access mechanisms for local, client-server, and Internet database access. He became very

popular as the first technology for integrating programming languages and databases.

The ADO.NET classes are divided into two components: the Data Providers, which handle communication with a physical data store, and the DataSet, which represents the actual data. Either component can communicate with data consumers such as WebForms and WinForms.

In our FIS model based on the ADO.NET Framework, we created a SQL database with log file. Database contains four tables. Each table represents an entity: Students, Professors, Courses and Exams. We tested this model measuring execution time in milliseconds of each operation for different number of records in a database. After 15 measurements, we found average execution time. Results are shown in Table III.

TABLE III: EXECUTION TIME OF A BASIC DATA OPERATIONS [MS]

Operation	1 record	100 records	500 records	1000 records
Writing	4,3087	0,9847	1,0058	1,0213
Reading	4,2123	5,9704	11,6373	17,3405
Deleting	4,4321	4,4784	4,4953	4,5835
Searching	4,0710	5,2427	8,5788	11,3158

There was no significant difference in the execution time of writing one record to database (M=4.54, SD=0.28) and writing thousand records to database (M=1.48, SD=0.27) conditions; t(28)=30.15, p=6.85E-23.

There was a significant difference in the execution time of reading one record from database (M=4.77, SD=0.52) and reading thousand records from database (M=17.48, SD=0.82) conditions; t(24)=50.61, p=6.64E-26.

There was no significant difference in the execution time of deleting one record from database (M=4.75, SD=0.42) and deleting thousand records from database (M=4.86, SD=0.16) conditions; t(18)=-0.94, p=0.36.

There was a significant difference in the execution time of searching one record in database (M=4.72, SD=0.45) and searching thousand records in database (M=11.71, SD=0.37) conditions; t(27)=-46.84, p=2.13E-27.

The size of a database and log file is shown in Table IV, as the overall size.

TABLE IV: DATABASE AND LOG FILE SIZES [MB]

File	1 record	100 records	500 records	1000 records
Database	3,06	3,06	3,06	3,06
Log File	1,06	1,06	1,06	1,06
Overall	4,12	4,12	4,12	4,12

There was no significant difference between size of database with one record and size of database with thousand records – they are the same size.

C. LINQ

LINQ stands for Language Integrated Query. One of the central, and most important, features of LINQ is its integration of a flexible query syntax into the C# language [9].

Developers have many tools that have been crafted to neatly solve difficult tasks. Yet there are still dark corners in the development landscape. Querying data is one area in which developers frequently encounter problems with no clear resolution. LINQ aims to remove that uncertainty and to show a clearly defined path that is well-lit and easy to follow [10].

LINQ could be considered as the missing link – whether this pun is intended is yet to be discovered – between the data world and the general-purpose programming languages. LINQ unifies data access, whatever the source of data, and allows mixing data from different kind of sources.

In our FIS model based on the LINQ, we created the same SQL database as in the FIS model based on ADO.NET Framework. Instead of using SQL query ADO.NET commands, we used LINQ for creating queries. We tested this model measuring execution time in milliseconds of each operation for different number of records in a database. After 15 measurements, we found the average execution time. Results are shown in Table V.

TABLE V: EXECUTION TIME OF BASIC DATA OPERATIONS [MS]

Operation	1 record	100 records	500 records	1000 records
Writing	1,2454	0,3309	0,5590	1,9905
Reading	9,0357	12,5744	13,3645	15,3456
Deleting	2,9358	2,9933	2,6643	2,3356
Searching	9,1908	11,4545	14,3462	12,2557

There was no significant difference in the execution time of writing one record to database (M=1.84, SD=0.56) and writing thousand records to database (M=1.50, SD=0.61) conditions; t(26)=1.56, p=0.13.

There was a significant difference in the execution time of reading one record from database (M=10.06, SD=0.51) and reading thousand records from database (M=16.94, SD=0.94) conditions; t(20)=-25.70, p=8.62E-17.

There was no significant difference in the execution time of deleting one record from database (M=2.18, SD=0.45) and deleting thousand records from database (M=2.09, SD=0.90) conditions; t(19)=0.39, p=0.70.

There was a significant difference in the execution time of searching one record in database (M=9.82, SD=0.45) and searching thousand records in database (M=12.07, SD=0.45) conditions; t(26)=-14.09, p=1.03E-13.

The size of a database in FIS model based on the LINQ is the same as the size of a database in FIS model based on the ADO.NET Framework.

VI. CONCLUSION

In order to utilise the combined power of programming languages and databases, this research focused on three different methods for integrating them. While the file system has the best performances from the aspect of execution time for basic data operations, it does not have a mechanism for data management (security, integrity, backup, …). With every new record, size of a file rapidly increases.

Accessing databases using ADO.NET Framework is fast and efficient, but it suffers from the impedance mismatch. Data are often returned as different types or custom types, and have to be cast or formatted into usable or .NET or native types. ADO.NET takes more execution time for basic data operations than a file system, but adding a new record does not rapidly increase the database size.

LINQ significantly changed some aspects of handling and manipulating databases with applications and components. It allows for query and set operations, similar to what SQL statements offer for databases. LINQ, though, integrates queries directly within .NET languages like C# and Visual Basic through a set of extensions to these languages.

LINQ is not the overall winner (as expected). While writing and deleting operations are faster in LINQ, reading and searching operations are superior in ADO.

We could say that the best method for integrating programming languages and databases is LINQ because it solves the problem of impedance mismatch, and the query syntax is similar to syntax of programming language.

REFERENCES

[1] N. Gehani, "*The Database Book: Principles and practice using MySQL*". 1st ed., Summit, NJ.: Silicon Press, 2006.

[2] B. A. Wiedermann, "*Integrating Programming Languages and Databases via Program Analysis and Language Design*", Ph.D. dissertation, University of Texas at Austin, U.S., 2009.

[3] M. Atkinson and P. Buneman, „*Types and persistence in database programming languages*", ACM Comput. Surv., 19(2):105-170, 1987. http://dx.doi.org/10.1145/62070.45066

[4] T. Bloom and S. Zdonik, „*Issues in the design of object-oriented database programming languages*", Proc. of ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications, pp. 441-451, ACM Press, 1987. http://dx.doi.org/10.1145/38765.38848 http://dx.doi.org/10.1145/38807.38848

[5] W. Cook, A. Ibrahim, "*Integrating Programming Languages & Databases: What's the problem?",* Expert Article, 2005.

[6] J. Ullman, H. Garcia-Molina, J. Widom, „*Database Systems: The Complete Book*", Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[7] S. Gupta, A. Mittal, "*Introduction to Database Management System*", Laxmi Publications, India, 2009.

[8] R. Riordan, "*Microsoft ADO.NET Step by step",* 1st ed. Microsoft Press, Washington, 2002.

[9] C. Calvert, D. Kulkami, "*The Essence of LINQ*", Addison-Wesley, Indiana, 2009.

[10] F. Marguerie, S. Eichert, J. Wooley, "*LINQ in action",* Manning publications, New York, 2007.