Performance Analysis of .NET Based Object–Relational Mapping Frameworks

Aleksandra Gruca and Przemysław Podsiadło

Institute of Informatics, Silesian University of Technology,
Akademicka 16, 44-100 Gliwice, Poland
aleksandra.gruca@polsl.pl

Abstract. Object-relational mapping is a technology that connects relationships with object-oriented entities, which aims to eliminate duplicate layers together with costs of maintenance and any errors arising from their existence. A lot of tools and technologies were designed in order to support and implement idea of object-relational mapping. In this paper we present the performance comparison of two most common object-relational mapping interfaces for .NET framework: Entity Framework and NHibernate. In the .NET developers community, there is a lot of discussion today about the similarities and differences of the both technologies. To address this issue, we compared the features and performance of both tools. We analysed the performance of Entity Framework and NHibernate for two different databases (MS SQL Server and PostgreSQL), different query languages (lambda expressions and LINQ for Entity Framework and HQL and Critera API for NHibernate) and compared the results with the standard SqlClient queries. The results show that there is no significant difference between these both tools and we proved that common opinion that NHibernate performs better than Entity Framework is incorrect.

Keywords: Object-relational mapping, ORM, .NET framework, Entity Framework, NHibernate, MS SQL Server, PostgreSQL, performance analysis.

1 Introduction

Most of the software applications which are nowadays developed are based on object-oriented programming languages and relational databases. Such approach allows creating applications which are built on the logical elements that interacts with each other, while the relational databases technology is responsible for preserving data structures in tabular forms with established relations. These two technologies, although they seems to be divergent, interact with each other in both theoretical and practical aspects. The object-oriented programming and relational databases are complement on data structures and interface middleware between data and the user. Theoretical background between these two technologies is obvious correspondence between the diagrams of database tables and the structure of the classes defined in the application.

S. Kozielski et al. (Eds.): BDAS 2014, CCIS 424, pp. 40–49, 2014.

Object-relational mapping (ORM) is a technology that connects relationships with object-oriented entities, which aims to eliminate duplicate layers together with costs of maintenance and any errors arising from their existence. A lot of tools and technologies were designed in order to support and implement idea of object-relational mapping but most of them is not suitable for commercial use and there are also some voices in the discussion that undermine the usability of object-relational mapping interfaces in larger projects [7].

The aim of the research presented in this paper is to analyse the performance and to compare two most popular and .NET based ORM interfaces. At the beginning stage of every software application project, when designing data access layer, a selection of the object-relation mapping interface is performed. Such decision must be considered very carefully, as subsequent costs of changing the interface even in partially implemented application, often outweigh the potential gains. Typically such a decision in made on the basis of the skills of a project team members, which often limits the number of choices and may have a bad influence on the process of designing application architecture. Therefore, information on the performance of data interfaces would be helpful in deciding on the appropriate technology.

1.1 Related Work

In the Internet one can find many articles, blogs and discussions comparing different ORM interfaces and various aspects of their practical application. In general, the discussion on ORM interfaces goes in two directions. First problem is how using any ORM interface influences the performance of database query and whole development process. Another discussion is which ORM framework is the best solution in terms of performance and other criteria that may be important for application development and maintaining. Currently, two most popular object oriented technologies are .NET and Java. There are a dozens different ORM tools for both technologies and their comparative list can be found at `http://c2.com/cgi/wiki?ObjectRelationalToolComparison`. Unfortunately, most of the websites that take up this subject is outdated or incomplete and there is a very little number of scientific publications focused on comparative analysis of ORM tools.

To address the above problems we analysed performance of two most popular open-source ORM interfaces based on .NET technology: Entity Framework [6] and NHibernate [1]. In the research performed in 2010 [4] the both tools were analysed, however over last years, both interfaces were extensively developed and improved, and especially Entity Framework got a lot of positive reviews since the previous release. Therefore it is reasonable to analyse if and how their performance has changed with new releases and also in comparison to the performance of the standard SqlClient queries.

The list of basic evaluation criteria for ORM tool suitability and general comparison of several ORM mapping tools based on proposed criteria can be found in [10], however the presented analyses are not focused on the tools performance. The up-to-date comparison for Java based ORM tools can be found in the

article [3]. According to the authors best knowledge there is no such comparison for the current versions of selected .NET based ORM tools. In addition to the work presented in [4] we compared the performance of different query languages for both interfaces, that is lambda expressions and LINQ for Entity Framework and HQL and Critera API for NHibernate. To make our conclusion more general, we used two different relational databases.

The results of the research performed in this paper not only show the differences among analysed technologies: NHibernate and Entity Framework but also analyse whether introduced delay is within acceptable limits, if compared to the standard SqlClient mechanism. Comparative analysis were performed for Microsoft SQL Server [9] and PostgreSQL [11] databases, however the main aim of this work is comparison between interfaces, not the database systems, therefore each database has different structure, schema and data.

2 Relational Databases and Object-Oriented Programming

Persistence is ability of an object to outlive the application process in which it resides. This term is often used in conjunction with the problem of storing objects in databases as to obtain persistence, the object (its current state, properties and relations), needs to be somehow preserved in non-volatile storage such as a hard drive.

The paradigm of object-oriented programming supports the building of applications out of elements that have both data (in a current state) and behaviour. The main issue, when translating the logical representation of the objects into a form that can be stored in database, is the fact that objects do not only contain the data but also have ability to share the data and communicate with other objects. The relational paradigm, however, is based on mathematical principles and the most important issue is the data itself, the relationships and data structure.

The term object-relational impedance mismatch refers to technical and conceptual issues that arise when we try to combine object and relational artefacts. The mismatch occurs, because object-oriented programming paradigm and relational databases have different concepts of data. The impedance mismatch is manifested in several different forms [2]:

- Inheritance and Polymorphism - inheritance is rather unimportant element of the classical relational model, which focuses on the structural relationships between data. There are methods allowing mapping the inheritance hierarchies, but there is no simple method to represent polymorphism, which is crucial for object-oriented programming.
- Associations - classes that exist in object-oriented programming are associated in behavioural way, which is unimportant from the relational point of view. One can easily imagine two classes with sets of mutual references which in relational model would be represented as "many-to-many" relation which would require intermediate table.

- Data types - different database systems and object-oriented programming languages have specific data types and sometimes it is unclear how to map them. The examples of such problematic data types are *DateTime/datetime* type which can be represented in many different ways depending on implementation.
- Granularity - in object-oriented systems classes may exists on different levels of granularity, starting from classes that model business objects and ending on basic data types such as *DateTime* or *Integer*. Therefore, the idea that every class should map to its own table cannot be realized in practice, so the application designer must decide which class should be represented by its own table.
- Identity - identity is one of the most crucial issues in the relational model. In theory, each table should have its own primary key, which allows the record to be unique, even if other attributes have identical values. In case of object-oriented systems, there are two methods of identity verification of two objects. Reference comparison verify whether object references refer to the same area in memory, while value comparison compares objects attribute-by-attribute. Each of these identity types is different.
- Data navigation - naive implementation of an object graph navigation may result in an exponential explosion of application queries to the database. One can easily imagine the problem with "one-to-many" mapping when the function retrieving attribute values creates connection to the database for each item. Therefore, the main problem is how to create the mapping which will minimize number of queries and thus loads several entities via JOINs and selects the targeted entities before starting to walk network of objects.

3 Object-Relational Mapping Tools

Object-relational mapping is a technique for converting data between incompatible object and relational systems. The basic idea standing behind ORM tools is to delegate the management of persistence to the external mechanism. With the use of ORM tools it is possible to work at code-level with objects representing a domain model being separated from the structure of relational database. Therefore, the ORM technology establishes a bidirectional link with the data in a relational database and objects in code, based on a configuration and by executing SQL queries (dynamic most of the time) on the database [8].

The basic features that should be covered by any ORM mapping tool are ability to use inheritance, create hierarchies between entities and use polymorphism. Another important thing is to handle any type of relations (1-1, 1-n, n-m), support for transactions, aggregates (equivalent to SQL's SUM, AVG, MIN, MAX, COUNT) and support for grouping (SQL's GROUP BY). Additional features that should be taken into account when selecting ORM tool are a list of supported databases, ability of use query language, flexibility (customization of queries, SQL joins support, support for the data types specific to the database management system, etc.), optimization and many more numerous criteria. Detailed list and description of features that may be considered when selecting

ORM tool can be found in [8,10]. While the most of these criteria can be verified just by analysing specification and documentation of specific ORM tool, there is still a lot of uncertainty when considering performance of ORM tools. Below, we analyse some features of two selected open-source ORM interfaces for .NET Framework: Entity Framework and NHbiernate.

3.1 Comparison of Entity Framework and NHibernate

Entity Framework. Entity Framework is a set of technologies in ADO.NET supporting object-relational mapping in .NET Framework. The first version of Entity Framework was introduced in 2008 year and received a lot of criticism due to many errors and technical misgivings with design and implementation. Since the first release, Entity Framework has advanced significantly, mainly due to involvement the community feedback during design decisions. Official builds of Entity Framework are available as a fully supported Microsoft products both standalone as well as part of Visual Studio.

NHibernate. Hibernate is one of the most popular object-relational mapping tools. It is non-commercial tool, developed from 2001 and originally designed for Java language. Since 2005 there is also available version for .NET environment, called NHibernate. To achieve persistence, there is no requirement for a .NET objects to implement any NHibernate interface. To communicate with database, NHibernate uses metadata from XML mapping files instead of additional interfaces and attributes.

In Tab. 1 we present comparison of the most important features of both analysed ORM mapping tools [4,5]. The big difference between Entity Framework and NHibernate from a developer perspective is that the former offers an integrated set of services whereas the latter requires the combination of several open-source libraries. This problem can be easily solved by using external tools such as NuGet (`http://www.nuget.org/`), however sometimes these external libraries are not always up-to-date when new NHibernate version is released [5].

4 Experimental Analysis

4.1 Configuration of Testing Environment

Computational analyses were performed on the PC computer with processor Intel Core i5 M 560 2.67GHz, 4GB RAM and operating system Windows 8 64-bits. As testing environment we used .NET Framework, version 4.5.5, Entity Framework version 5.0 [6] and NHibernate, version 3.3.1 [1]. Analysed databases were Microsoft SQL Server 2012, version 11.0.3 with modified Nortwind database [9] and PostgreSQL, version 9.2 with modified Dellstore database [11].

To compare performance between two ORM interfaces we prepared eight different types of queries. For each ORM query the running time was measured and compared with the same operation performed with standard SqlClient method.

Table 1. Comparison of selected features of Entity Framework and NHibernate

	Entity Framework	NHibernate
Programming Language	C# and VB	C# and VB
Code-based mapping	Automated	Automated (from v3.2) or external Fluent NHibernate
Bidirectional relationships support	Yes	Yes
Transaction support	Yes	Yes
Blocking mechanisms	Optimistic and pessimistic	Optimistic and pessimistic
Optimization	Buffering and lazy loading	Buffering and lazy loading
Custom types and collections	No	Yes
Private fields mapping	No	Yes
Automatic migration	build-in Code-First Migrations	external Fluent Migrations
Queries	LINQ to Entities, Entity SQL, SQL	HQL, Criteria API, QueryOver, SQL
Database support	MS SQL, SQL Azure PostgreSQL, MySQL Oracle, SQLite, DB2, Firebird	MS SQL, SQL Azure, PostgreSQL, MySQL Oracle, SQLite, DB2, Firebird

Each query was executed 100 times and the average time was computed. Presented results do not include the data context initialization which is very expensive to create and can be performed in several different ways for each technology.

For testing purposes we selected queries that are the most representative and frequently used in software applications. We analysed basic CRUD (Create, Read, Update, Delete) functionality, as well as more advanced queries using popular operators such as joining, grouping, ordering and queries with subqueries. As we wanted to analyse if and how the performance of ORM tools evolved during the last several years, we compared our work with the results presented in article [4]. Therefore several of the queries and data types used were similar to these presented in [4] in terms of operators used and data types compared.

Each query was prepared in several versions: SQL with the use of SqlClient, HQL with the use of criteria API and with LINQ to Entites in two versions: as standard LINQ query and with lambda expressions. The queries used in our research for MySQL and PostgreSQL databases, are presented in Tab. 2 and Tab. 3 respectively.

4.2 Results for MSSQL Database

Analysis of the results for MSSQL database shows how small is the difference between two analysed ORM interfaces. This is obviously in opposition to the popular theory that performance of NHibernate is better than performance of Entity Framework. We can even notice that in some cases, Entity Framework performs a little better than NHibernate. It is also worth to notice that for a selected interfaces the time of the query execution is almost identical in most of

Table 2. Query list for MSSQL database

Q1	insert into Products values(SqlAddNew2,14,2,1kg per pack,10,null,null,null,0)
Q2	update Products set ProductName=SqlUpd where ProductName=SqlAddNew2
Q3	delete from Products where ProductName=SqlUpd
Q4	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where ProductName =Producut 21099
Q5	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where ProductID =21067
Q6	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where UnitPrice =2000
Q7	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where ProductName like %Producut% and ProductID = 21067 and UnitPrice > 20000 and UnitsOnOrder<1
Q8	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where ReorderLevel!=13
Q9	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where p.ProductID<50
Q10	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where p.ProductID<50 and s.SupplierID <10
Q11	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where ProductID > 21070 order by ProductID desc
Q12	select count(ProductID) from Products group by CategoryID
Q13	select * from Products p join Categories c on p.CategoryID = c.CategoryID join Suppliers s on p.SupplierID = s.SupplierID where s.SupplierID in (select SupplierID from Suppliers where Country = Germany)

Table 3. Query list for PostgreSQL database

Q1	insert into products(category, title, actor, price, special, common_prod_id) VALUES (2,'Sqladd2', 'Przemek Podsiadlo', 100, null, 1)
Q2	update products set title ='Sqlupd' WHERE title='Sqladd2'
Q3	delete from products WHERE title='Sqladd2'
Q4	select * from products p join categories c on p.category =c.category where title ='AIRPLANE BERETS'
Q5	select * from products p join categories c on p.category =c.category where prod_id =5600
Q6	select * from products p join categories c on p.category =c.category where price >20
Q7	select * from products p join categories c on p.category =c.category where title like '%AIRPLANE%' and prod_id = 6066 and price>10
Q8	select * from products p join categories c on p.category =c.category where special <>0
Q9	select * from products p join categories c on p.category =c.category where p.prod_id<50
Q10	select * from products p join categories c on p.category =c.category where p.prod_id<50 and c.category <10
Q11	select * from products p join categories c on p.category =c.category where prod_id > 9990 order by prod_id desc
Q12	select count(prod_id) from products group by category
Q13	select * from products p join categories c on p.category =c.category where c.category in (select category from categories where categoryname = 'Action')

the cases, therefore there is no additional overhead associated with creating the query with the use of lambda expressions for Entity Framework or Criteria API for NHbiernate.

When comparing query execution time to the standard SqlClient, the results are in general in accordance with expectations. The advantage of the standard SqlClient can be seen especially in CRUD queries when operation does not return any data and we take advantage from *ExecuteNonQuery* method. In case of more advanced queries, there are some examples, when the performance of ORM tool

is even better then simple SqlClient, which is explained by better optimization for ORM based queries when referencing to database and mapping the results.

Known disadvantage of SqlClient is shallow mapping, when the mapped object from one-to-many relation have the instance of related objects, but this instance does not have any collection of the objects related to it, until another query to the database is executed. In case of using ORM tools, this problem is solved in automated manner. In addition, the mechanism of lazy loading will not load any object until it is referenced.

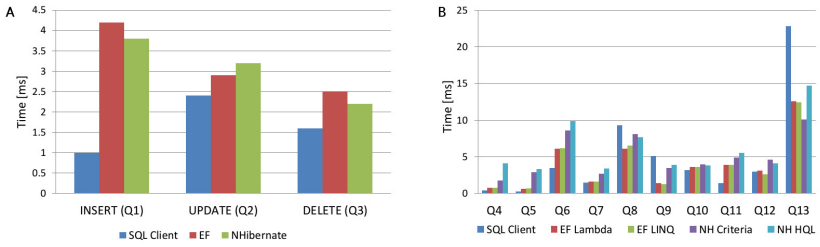

Fig. 1. MS SQL Server results: (A) – CRUD queries, (B) – advanced queries

Detailed comparison of our results with the results presented in [4] was impossible due to different testing environment and database version. However, even in a such condition we could make some general conclusions. In the results presented in [4], there is quite noticeable difference for GROUP BY between Entity Framework and NHibernate due to non-optimal manner in which group-by statements are translated into native SQL form by Entity Framework. This problem is obviously solved in the current version of Entity Framework as we do not observe such differences between both tools.

4.3 Results for PostgreSQL Database

In case of PostgreSQL, when comparing Entity Framework and NHibernate performance, we can see that NHibernate queries perform a little bit faster. This is the result of using external data provider NpgSQL `http://npgsql.projects.pgfoundry.org` which is developed by rather small community and with the limited resources. The influence of using NpgSQL can be also seen when analysing the differences in performance of the both interfaces and standard SQL client which are not big as in a case of other databases. Instead of NpgSQL one can use Devart's dotConnect for PostgreSQL `http://www.devart.com/dotconnect/`, however it is a commercial product.

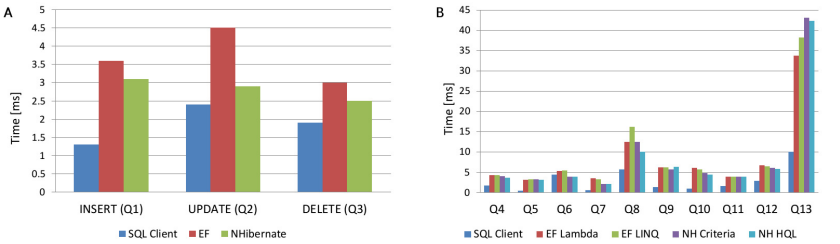

Fig. 2. PostgreSQL results: (A) – CRUD queries, (B) – advanced queries

5 Conclusions

Object-relational mapping interfaces are being more and more popular in the object-oriented applications, allowing to persist model objects to a relational database and to retrieve them. ORM technology provides a clean separation between physical structure of data and its object representation. There is a number of benefits to use ORM tools such as reduction of code, better application design and maintainability, and ability to take a full advantage of object oriented programming paradigm. Therefore it is necessary to measure the performance of ORM tools in order to recommend better and faster solution for developers.

In this paper we analysed the performance of the two most popular object-relational mapping interfaces for .NET platform. Both tools are open-source software, however they are mature and well-established products. We compared performance of the selected tools with standard SqlClient for several different, most frequently used types of queries.

The performance of standard SqlClient was in general better than performance of ORM tools, which is in accordance with intuition, as most ORM tools are designed to handle variety of data-use scenarios, far more than it is need for any single application. However, the differences between simple SqlClient approach and well-designed ORM tools is not a significant one, and the delay in execution time probably would not be noticed by the users. The analysis was performed for two different databases. As the database structures and queries were different, the results cannot be compared directly, however the observed tendency is similar in both cases, regardless of the structure and queries used.

It is also worth to notice, that there is a very small difference for NHibernate between HQL and Criteria API queries. The same observation can be made for Entity Framework when comparing performance of lambda expressions and LINQ. This allows developers to select the more suitable method. In case of using LINQ or HQL we have more readable and understandable code, especially for static and complex queries. However, when there is need to create dynamic queries, it is much easier to do it using Criteria API or lambda expressions.

The results presented in this research clearly shows that the widely held opinion that NHibernate performs better than Entity Framework is incorrect. Such opinion resulted from the drawbacks and errors that were present in the very

first versions of this software, however it seems that nowadays Entity Framework gains more and more popular trust in .NET developers community. The integration of Entity Framework with the .NET Framework and extensive support for it in Visual Studio makes it more and more popular tool. In addition, .NET Framework can automatically generate edmx file, context and models which are related to database structure, while in case of NHibernate there is still need to use external and frequently commercial tools.

In the conclusion, performed analysis does not give the clear answer which ORM interface or query language should be recommended if considering only the tool performance. In the authors opinion this is desirable situation, as it gives more flexibility to the developers team. There is a lot of other criteria that need to be taken into account when recommending object-mapper solution, some of them were mentioned in section 3.

Acknowledgements. The work was supported by Polish Ministry of Science and Higher Education (0161/IP2/2011/71) and partially by Ministry of Science and Higher Education, as a Statutory Research Project (8686/E-367/S/2013).

References

1. NHibernate 3.3.1 (2013), `http://nhforge.org/`
2. Barnes, J.: Object-Relational Mapping as a Persistence Mechanism for Object-Oriented Applications. Master's thesis, Macalester College, Saint Paul, Minnesota, USA (2007)
3. Bhatti, S., Abro, Z., Rufabro, F.: Performance evaluation of java based object relational mapping tool. Mehran University Research Journal of Engineering and Technology 32(2), 159–166 (2013)
4. Cvetković, S., Janković, D.: A comparative study of the features and performance of ORM tools in a .NET environment. In: Dearle, A., Zicari, R.V. (eds.) ICOODB 2010. LNCS, vol. 6348, pp. 147–158. Springer, Heidelberg (2010)
5. Doomen, D.: Entity framework 5/6 vs nhibernate 3 – the state of affairs (2013), `http://www.dennisdoomen.net/2013/03/entity-framework-56-vs-nhibernate-3.html` (accessed December 2013)
6. Entity Framework team: Entity framework 5.0 (2013), `http://entityframework.codeplex.com/`
7. Fowler, M.: Ormhate (2013), `http://martinfowler.com/bliki/OrmHate.html` (accessed December 2013)
8. Marguerie, F.: Choosing an object-relational mapping tool, `http://madgeek.com/Articles/ORMapping/EN/mapping.htm` (accessed December 2013)
9. Microsoft Corporation: Microsoft SQL server 2012 (2013), `http://www.microsoft.com/en-us/sqlserver/`
10. Płuciennik-Psota, E.: Object relational interfaces survey. Studia Informatica 33(2A), 299–310 (2012)
11. The PostgreSQL Global Development Group: PostgreSQL 9.3.2 (2013), `http://www.postgresql.org`