Abstract

Stored procedures are used as the current database logic for SAAB’s data model of the fighter aircraft JAS-39 Gripen electrical schemas. Since the database model was developed in 2000, a research and tests needed to be carried out to decide on whether updating the database to today’s technology is applicable. Therefore, Object-Relational Mapping (ORM) is to be researched, tested and compared to stored procedures using test-driven development (TDD) concerning an important factor, that Stored procedures are well-known for, which is querying performance of the database. Moreover, how maintainability and flexibility [1] can affect decision between Stored procedures or migrating to ORM based on our subjective experience. NHibernate and Entity Framework are the two ORM solutions considered since SAAB uses C# in this project. The process of this project is run using scrum of the agile software development to maintain an iterative progress throughout the project timeline. In this paper, the process and methodology are covered in details and also the comparison with the test results. These results eventually lead us to the answer that ORM is not a suitable technology, and stored procedures still dominate the querying performance for SAAB’s current database.
Acknowledgements

First of all we would like to thank our supervisor, head of Computer Science Department, Jesper Andersson who helped us in writing this report. We especially thank him for his accurate comments which were critical to this report. Our sincere thanks to Natan Gelber, our supervisor at SAAB, for accepting us to perform our thesis work at SAAB and for his support and great assistance with his experience to accomplish the results in this report.

We would also place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.
# Table of contents

Abstract .................................................. 2
Acknowledgements ....................................... 3
Table of contents ....................................... 4

1 Introduction ........................................... 5
   1.1 Background ...................................... 6
   1.2 Previous research ................................ 6
   1.3 Problem formulation ............................. 6
   1.4 Motivation ....................................... 7
   1.5 Research Questions .............................. 7
   1.6 Scope/Limitation ................................. 7
   1.7 Target group ................................... 8
   1.8 Outline ......................................... 8

2 Method .................................................. 10
   2.1 Scientific Approach .............................. 10
   2.2 Method Description ............................. 11
   2.3 Reliability and Validity ........................ 12
   2.4 Ethical Considerations ........................ 12

3 Implementation ....................................... 14

4 Results .................................................. 18
   4.1 Basic Model Test ................................ 18
   4.2 ELDIS Model Test ............................... 19

5 Analysis ............................................... 24

6 Discussion ............................................ 26

7 Conclusion ............................................ 29
   7.1 Future Research ................................ 29

References .............................................. 30
1 Introduction

In computer science and software development, data storage and persistence is a very important aspect that keeps data and information across reboots and between subroutines. Data storage is also the most important part in information handling in every field imaginable because of its simple nature and high value. Depending on the type of data, databases and storage can be very different in structure and operation. Therefore, various database structures have been introduced in order to suite all cases. There is the relatively new and modern Document-Oriented-Model such as MongoDB where the data objects are stored as JSON formatted documents. In other words, the database is structured as collections where document objects are stored. This flexible yet efficient database structure enables storage of any type of document shape because saved items do not need to follow a certain structure or model as long as it conforms to JSON formatting. An older and more widely used structure is the table based SQL format where data is divided into tables, columns, and rows. In this case, data is stored in a certain format and order which structures data in a very organized fashion.

In order to retrieve information, these two data storage approaches have different syntax and operators. In the older SQL, the data is fetched using queries via a sentence-like language that fetches data based on the commands in the sentence-string. Within this string, there can also be table operators such as JOINs, ANDs, ORs, SELECT, etc. These operators can join tables on certain conditions or columns, after which a certain row is selected on a condition and so on. This querying language works well on structured data and can pick out data pieces individually very well. It is also very efficient because of its organized nature, however, it lacks the flexibility and the ease of working with it because of how precise and thorough its implementation and queries have to be.

On the other hand, the newer MongoDB design, ODM, uses a simplified design that functions for document based databases. It uses a more programming-like query language commonly using dot-formatting where dots separate operators on the database. For example, DB.Collection.find(), as opposed to the more lingual SQL queries that look something like: SELECT * FROM DB WHERE item = “data”. The benefits of using an ODM is that maintainability and flexibility increases while sacrificing efficiency and performance. The two database structures are completely different in design but their data-fetching querying is quite similar. The main difference being that an ODM takes the more programming-like query and converts it into a more SQL-like query as a middle step.

There is another database design called ORM (Object-Relational-Mapping) based on a semi-merge between the ODM and Relational database design. The data is stored in tables, columns, and rows, but the querying is done in a similar way that ODMs query. This means that data is structured and organized very well and follows a certain model. However, ORM gains the ODM querying methodology of dot-formatting the queries which gives the design easy maintainability and flexibility aspects.

The ORM database thus gains the benefits of having a more object-oriented way of querying data from the database and is simultaneously able to keep its current relational database structure and table relations. However, the question still remains whether a
migrants to an ORM implementation is worth it which is the very question this thesis research is about.

1.1 Background
SAAB is a global defense and Security Company active in air, land, and naval-based defense systems. SAAB is also involved in civil security and commercial airline technology. There are about 15,000 employees and are involved in activities on all continents. SAAB is on the leading edge within many areas and a fifth of their income is distributed to the sciences and development.

Because of SAAB's involvement in the defense forces, they have numerous databases containing various information. However, some of the database presentation tools are not quite up to today's standards. As such, their current database presenter for the electrical schemas of JAS-39 Gripen fighter aircraft is outdated and they are now developing a new presenter called ELDIS to accommodate the next iteration of the aircraft.

SAAB now requests assistance in investigating whether it is applicable to migrate their current database logic of stored procedures (Relational Database) to an Object Relational Mapping system.

The main focus of the research is to evaluate the benefits and setbacks of migrating the current relational database model to the ORM model. However, before taking on the evaluation one must understand the differences between the different available database models and the interaction layer between the database and the main system. On top of which, how querying data from these databases is performed.

1.2 Previous research
A paper called Object Relational Mapping was useful during writing this paper that was studying ORM in a general manner with different approaches of setting up the mapping between relational and object models. Some of the contents of the Object Relational Mapping paper are mentioned later on in this paper.

1.3 Problem formulation
Should SAAB migrate their database model from Stored Procedures to an ORM implementation and if so, what are the benefits and drawbacks?

The main task is to compare the current data model design which consists of a logistical storage of pre-stored procedures to an ORM technique and decide which technique is best applicable for SAAB. When comparing between those two different techniques in general cases, stored procedures are well-known to provide more control over the data retrieved from the database and higher queries performance. The reason behind this is that queries are written in SQL, which are specifically tailored to the required data. On the other hand, querying in SQL is a double-edged weapon since this affects the variability of data retrieval because the queries are fixed. However, ORM is a technique that can provide extra flexibility [1] which can be optimal in maintaining [1] the database model design with the drawbacks of adding another layer that handles the SQL querying.
1.4 Motivation
Each technique, whether it is ORM or Stored Procedures, provides different perks and naturally has its own advantages and disadvantages. By carrying out the methodology, which is mentioned later on in this document, it will be possible to identify if an ORM improves SAAB’s needs for the database design. Thus, such a result would benefit the company and subsequently the industry because more knowledge and information have been acquired on the available options for various techniques for representing their data model. Also, if the migration to ORM is more rewarding than Stored Procedures, SAAB would then have an easier approach to the possibility of migrating to a document-oriented database (e.g. MongoDB) for future variations and improvements. In short, one of the goals of this project is to provide SAAB with the information and analysis needed for a wider access to a diversity of database techniques.

1.5 Research Questions
The table 1.1 contains the research questions for this thesis:

<table>
<thead>
<tr>
<th>ID</th>
<th>Research Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1.</td>
<td>What are the advantages and disadvantages of the Stored Procedures compared to ORM with the current database design model in terms of queries performance?</td>
</tr>
<tr>
<td>RQ2.</td>
<td>Concluding from the previous research question while considering flexibility and maintainability factors, where and when to use stored procedures or ORMs?</td>
</tr>
</tbody>
</table>

Table 1.1: Research Questions

1.6 Scope/Limitation
The project can become overwhelming and too wide to cover that we cannot reach a scientific-based decision at the end of the project. Therefore, the extent of the goals has to be limited and boundaries needed to be put so that the project is feasible. As mentioned earlier in the Abstract section of this paper, ORM has many frameworks for different programming languages (e.g. LiteSQL for C++, Hibernate for Java, Django for Python), hence the research only covers two frameworks for .NET which are NHibernate and Entity Framework.

The stored procedure database is to be implemented in Microsoft SQL Server RDBMS. The frameworks and the stored procedure RDBMS are chosen based on the fact that the programming language used for this project is C# which constrains the scope of our research and narrows down the frameworks alternatives.

The comparison points between ORMs and stored procedure also have to be scoped, thus not all the aspects of both technologies are to be covered. This being said, one main aspect is to be analyzed in addition of two other aspects which are:

- Queries Performance (main aspect)
- Maintainability
- Flexibility.
According to the goals of this project, these qualities will suffice to present a feasible analysis upon which a decision can be made on whether to migrate from Stored Procedures to ORM. The emphasis is on whether or not the benefits and setbacks of each implementation are worth either staying with Stored Procedures or Migrating.

The project will cover only one of SAAB’s data model with all of its relations. Hence, the size of the project will be more manageable that would provide more accurate results. On the other hand, the project will not cover all types of data model operations, which will compact the scope and also set limits of what the implementations and research will cover in terms of testing results and data. The goal here is to compare queries that grasp as many relations in the data model as possible and more likely to be used in context to SAAB’s needs.

Regarding the querying performance testing, not all types of performance testing shall be tested. The main task of this testing is to find how fast the Stored Procedures queries used by SAAB are when translated to ORM. Therefore, other performance tests such as Stress testing, load testing or endurance testing are not be entirely covered in this report.

Software flexibility is to be tested in the sense of how different the code is written in ORMs in order to execute various queries compared to Stored Procedure queries. Also, the flexibility testing covers how NHibernate and Entity Framework adapts to manipulating the relationships between the database tables and how much change is needed in the mapping configuration in order to connect those relationships among the tables.

1.7 Target group
The main target of this report and research is the SAAB group. Though the question of where and when it is applicable to use either Stored Procedures or an ORM implementation is an open and general question. Thus, the target group is software developers and other individuals and organizations that have a need for database interaction and data models. The information contained in this report will hopefully aid these entities in deciding which implementation to choose for their data model.

1.8 Outline
The parts of this report are listed here with a brief description of what they contain.

2. Method
This chapter will focus on describing the scientific approach that will be used to answer our research problem and question. It will contain the methods used and the reliability and validity of our approaches. This chapter will also include ethical considerations of the area of study.

3. Implementation
This section is meant to showcase our implementations done for this research. Generally, methods, tactics, and approaches that have been implemented to get the research results. Here we also describe the software, other libraries and frameworks used.

4. Results
The results section will contain the results and their descriptions, mainly the performance of the querying implementations implemented for this research. This part will not cover any speculations or conclusions but rather be a section of solid test data. This chapter will visually display the test results in the form of graphs.
5. Analysis
This chapter will focus on giving meaning to the results and contain speculations and conclusions regarding the test result.

6. Discussion
This part will mainly be directed at discussing the findings of the research and whether or not the test results and analysis give an answer to the research questions. A discussion of previous research compared to the results of this report will also be mentioned. Are our results close or similar to our hypothesis?

7. Conclusion
The final section of this report will consist of summarizing what has been determined, how it has been shown and the relevancy of the findings for computer science and the industry in general. This section will also touch on how the research results can be applied in other areas of computer science. Lastly, this chapter will mention ways that our approaches and implementations could have been improved to obtain more accurate readings or results.
2 Method
The methodology applied for this report is based on the research questions. Milestones are used to define the checkpoints of the methodology plan so that we can keep track of the thesis progress. The process for the milestones is iterative so working on each milestone is in parallel to the other milestones. The milestone descriptions in chapter 2.1 are directly tied to the research questions and are thus elaborated on here. This is also the methodology that will be applied when answering the research questions and how data will be gathered.

2.1 Scientific Approach
Milestones
Table 2.1 illustrates the goals of each milestone for this thesis which is described thoroughly below the table:

<table>
<thead>
<tr>
<th>ID</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ms1</td>
<td>Analyze the current data model, software design, and source code. Test the stored-procedure implementation.</td>
</tr>
<tr>
<td>Ms2</td>
<td>Implement the ORM DAO layer for the database.</td>
</tr>
<tr>
<td>Ms3</td>
<td>Test the ORM DAO implementation.</td>
</tr>
<tr>
<td>Ms4</td>
<td>Compare test result data between ORM and SP.</td>
</tr>
<tr>
<td>Ms5</td>
<td>Discuss results of test data. Identify pros and cons of ORMs &amp; SPs. Identify best usage situations and conditions for ORMs and SPs</td>
</tr>
</tbody>
</table>

Table 2.1: Milestones Goals

1. Ms1 is to review and study the database design model and source code by measuring the querying performance coupled with the flexibility and maintainability aspects. Afterwards, sufficient data is collected to make the analysis utilized in the forthcoming comparison between stored procedures and ORM. Also, Ms1 is fundamental to the rest of the project because it will decide how the structure of the ORM implementation will be implemented.

**Ms1 will yield:**
Performance test data on the stored-procedure-implementation and a decision on which ORM to implement.

2. Ms2 is the implementation of the new DAO layer following the suitable ORM that was elected in Ms1.

**Ms2 will yield:**
The new DAO layer implementation.

3. Ms3 will be an analytical phase where the ORM implementation is tested in the same way that the stored-procedure implementation was tested. This will be done through measuring the query performance in relation to the flexibility and maintainability aspects.
Ms3 will yield:
Performance test data on the ORM implementation.

4. Ms4 will contain a comparative analysis on the differences between the two implementations based on the performance test data acquired in Ms1 and Ms3.

Ms4 will yield:
A side-by-side comparison of numbers and figures based on the test data.

5. Ms5 will be a discussion-analysis where the comparative data from Ms4 will be discussed. There will also be a pro and con comparison between ORMs and SPs. Ms5 will also contain a section describing preliminary usage areas and conditions where ORMs or Stored-Procedure implementations are more suitable.

Ms5 will yield:
A discussion on the result sets in Ms4.
It will contain a list of pros and cons of ORMs/SPs.
A suggestion on usage, based on the test data, of ORMs/Stored-procedures in relation to what data they are connected to.

2.2 Method Description
Approaches
The implementation of the new database interaction layer will be done in TDD (test-driven development). This means that the test cases and test suites will be built where each unit test will be written before the actual queries and methods.

This approach will ensure increased quality for the implementation [2] since it will give near maximum test coverage and thus ensure correct and quality code. Because of the methodology behind TDD, the implementation will yield greater variability, modularity, high cohesion and lower coupling than a normal non-TDD approach.

The project progress is handled in a SCRUM approach in order to maintain the updates with the project members and supervisors. The sprints will be on a weekly basis where every Friday is a SCRUM meeting where the appointed SAAB supervisor is the SCRUM-master. The weekly meeting will summarize the progress made during the week and preliminary goals and planning for the following week.

System Analysis
The first step to answer any of the research questions at all is to analyze the current implementation to gain insight as to how the system and data model is structured. This includes reviewing the query structure and database table relations. With this information, a smaller and simpler data model will be constructed first. This is intended to familiarize with Entity Framework and NHibernate together with working with these in visual studio. Later on, the system analysis is to be extended to cover all the required outcomes of this project as mentioned in section 1.6.
Implementation
Before the data model of SAAB (ELDIS data model) is implemented and analyzed, a simpler data model will be constructed and analyzed for a small scale performance and analysis. This sample data model will serve as an analysis example of how the performance differences show in a small relational database. Later on, database scaling will show if the performance is affected by the size and complexity of the data model.

The ORM implementation will be iterative because SAAB does not want to release their entire data model due to both security and workload reasons. Thus, the development team will receive portions of the data model and the stored procedures. When the received portion is completed, a new portion will be given. This will hopefully yield the most important queries success whereas the less used or simpler ones are of a lower priority.

Comparison
After the ORM queries are implemented, these queries will be compared to the Stored Procedures. They will be compared mainly in terms of querying performance. They will also be analyzed in the sense of scalability in regards to the data model and relational complexity. In addition, flexibility and maintainability will be shed upon subjectively based on our experience throughout the project. Therefore, the primary comparison aspect that is experimentally shown in this paper is the query performance.

Query time performance:
When comparing query time, the main focus is the execution time from when a request is made to the data being returned to the method requesting it. Queries will vary from simple fetch to a more complicated join queries from multiple tables. This variance in the queries complexity provides better analyses information about the ORMs and Stored Procedures querying performance.

2.3 Reliability and Validity
The main validity and reliability concern in this project is that the research area is quite specific and does not contain or cover the entire domain of either Stored Procedures or ORMs. This means that the research on the topic of Stored Procedures vs ORM will be directed to a specific area of research, study, and implementation.

Nonetheless, the main goal of this study is to compare an ORM framework and implementation to a non-ORM implementation that uses Stored Procedures. The problem is general and not too specific in nature means that others can benefit from this information in other areas too.

2.4 Ethical Considerations
The main concern regarding this thesis project is that SAAB is a manufacturer and seller of the military, defensive equipment, machinery and services and thus, working for SAAB means being a part of such an organization.

Before even being allowed to review the details of the project and access their current data model, the authors of this report were required to sign an NDA agreement regarding data
and information acquired, analyzed and handled during the course of the project and cooperation with SAAB.

The supervisors of this project have been asked for permission to have their names tied to this report, which was granted.
3 Implementation

The first part to implement was the basic data model and relational model that all experiments are going to be based on since the ORMs to be built using the Database First development. As stated in section 2.2, Implementation, the first basic data model will be a simple test data model to ensure query functionality and performance in a small and manageable scale in order to get some sort of baseline test results that can later be compared to the ELDIS data model.

Basic Data Model

![Figure 3.1: Basic Data Model](image)

The Basic Data Model in figure 3.1 is, as previously mentioned, used as an example database model with a very small scale. The example is based on a simple Student-Course situation, but the Student entity is only used in this case to initiate simple testing and get an approximate overview on the difference between Stored Procedures and ORMs query performance. Notice that the Course entity exists in the model only to test the foreign key relationship performance. Therefore, simple CRUD (Create, Retrieve, Update and Delete) operations were used to test the query performance of Stored Procedures, NHibernate and Entity Framework. The testing operations are further explained along with the results and analysis phase later on in sections Results and Analysis.

ELDIS Data Model

Since ELDIS is the actual data model of SAAB, the information that ELDIS holds is sensitive and cannot be published. Therefore, due to security and privacy reasons signed under the NDA to protect SAAB’s data model, this paper does not include the actual model diagram or information that would expose SAAB. However, the characteristics of the database model is to be mentioned for the reader to grasp the scale and properties of the database.

ELDIS database holds information with the purpose to identify the wiring and connection types of cables within the aircraft. The database is rather complex with more than 18 tables, and table can have more than 30 columns. What makes this database a complex one is how the tables are connected together, foreign keys are connecting a table to at least another table. In other words, all tables are connected to each other, some of these tables have a one-to-one relationships, while other tables have a one-to-many relationships.
Stored Procedures

Stored procedures are commonly used in relational database management systems (RDBMS) to manipulate or access data through the use of prewritten strings that are instructions on what and where to query. Microsoft SQL Server is chosen to be the RDBMS for this project since the programming language used is C#.

The version required to be used for Microsoft SQL Server is 2014 which NHibernate and Entity Framework is built upon. However, NHibernate does not provide supportability to the SQL Server 2014, the latest supported SQL Server is 2012 [3]. Therefore, NHibernate database module is based on SQL Server 2012 database, but Entity Framework database module is based on SQL Server 2014 database.

For constructing the basic and the ELDIS databases based on the mentioned data models, Microsoft SQL Server Management Studio is used as an aiding tool to implement the database and analyze it. Of course, the management studio of version 2012 is used because NHibernate only supports SQL Server 2012.

NHibernate

NHibernate is an open source ORM for the .NET platform that allows the developer to communicate with a relational database, basically CRUD operations, in an object-oriented manner. In other words, NHibernate is an abstraction of the relational database SQL where the developer does not need to worry about the SQL syntax, and dealing with the data present in the database as objects. This framework is a port of Hibernate for Java and was possibly the most mature and capable ORM among the other existed frameworks in 2011 [4].

In the process of building the environment for NHibernate in Visual Studio 2017, the required packages were downloaded and installed using NuGet Package Manager. Mapping is required between the object-oriented classes and their equivalent relational database tables, therefore XML files are used for configuration and mapping. Unlike Entity Framework, NHibernate has relatively more complicated processes in order to get started using the ORM framework properly due to the need of XML files for translating the objects.

There exists another method to configure the mapping for NHibernate called Fluent NHibernate which is easier than XML-based files in the sense that the developer only writes the mapping in C# and never use XML. Of course, this provides a less error-prone, concise and more readable code. On the other hand, the disadvantage of Fluent NHibernate is more code written since for every entity, an extra class needs to be created for mapping purposes. Also, performance is affected by Fluent NHibernate which hinders the query process since the mapping is in C# and not XML. Last but not least, a rule of thumb taught to developers is to always divide between areas that have different purposes, therefore dividing between XML documents for mapping and C# classes for entity objects is more organized than having all together in C# classes. [4]

One of the benefits of NHibernate is Lazy Loading, and also stands true for Entity Framework. When an object is required to be retrieved from the database, other data related to this object is retrieved as well. This approach minimizes the access times of the ORM to the database and results in higher performance of the ORM queries execution. However, if the object to be retrieved is related to too many other objects, then loading all of those objects would hinder the query performance, especially if the acquired objects are not needed. Therefore, lazy loading has a limit of data fetching and uses a marker on the object structure to use the data only when needed. [8]
Entity Framework

Entity Framework is an open source Object-Relational-Mapping mapper for ADO.NET applications and used to be a part of the .NET framework. However, it was separated as of version 6. According to the NuGet package manager website statistics, Entity Framework is the most used ORM framework.[5]

Entity framework has the ability to allow complex database queries to be created without needing to manually write the SQL queries by hand.[5] Entity Framework uses something called LINQ which is a sort of syntax that acts as a middle hand between the application code and the database. LINQ acts as a converter of object-oriented commands and orders where a query can be written in a similar way of object-oriented dot-format. This is then converted into an SQL-string that is executed as normal.

This automation makes database interaction and querying simpler from a developer’s point of view since it allows a more streamlined approach to query data. However, this automation in LINQ can severely hamper the performance of the application where poor queries in relation to the data model are major bottlenecks.

The main reason for bottlenecking is the fact that Entity Framework is based on the idea that tables are Entities. When something needs to be queried, the object-oriented approach is to load the entity as a variable in memory and then perform operations on it. This means that the size of data in the entity directly affects performance because the data handling is carried out in memory. In turn, this creates a lot of arbitrary and redundant information.

Because of this, there are some rules that should be followed in order to retain as much performance as possible. [5]

1. Only put data in memory that is actually needed.
2. Avoid querying the database inside loops.
3. Try to write a long query instead of several short ones.
4. Do not query too deeply.
5. Use JOINs to attain maximum performance.

These basic steps are common mistakes that should be avoided when querying with LINQ.

EF performance can be improved by how queries are written and how they are ordered. It is not uncommon for EF users to have dedicated, short, initialize method(s) that force EF to compile tables and entities, meaning a start-up time before the database can be accessed. This can be vital for the performance of the ORM. Another way to maximize performance is to limit the queries to only get specific and small pieces of data from the database. It is also possible to use a cached instantiated database to query to and from. This means that the data model is quickly accessible in memory and can yield very quick execution times. However, caching databases or tables can be resource intensive and should in most cases be evaluated if it is worth to perform or not. A cached and instanced database can later be merged into the old existing one and thus achieve eventual consistency for the data. It could also be executed as an asynchronous task in which the system continuously merges the real database with the cached one.

Entity framework also allows pure SQL queries to be written manually and executed, however, this would defeat the purpose of having Entity Framework since there already are stored procedures in place in the already tested implementation of SAAB’s application.
There are three main scenarios that are useful for entity framework which figure 3.5 illustrates:

1. There already exists a database or the database design is desired to be created before the other parts of the application.
2. The main focus is to create domain classes and then create the database from these classes.
3. The database schema can be designed from the visual designer element and then create the classes and databases.

Figure 3.5: The three uses of Entity Framework.[6]

In the case of SAAB, they already have a database, domain classes and an application that uses them. Therefore, entity framework implementation and integration will work well because of its ability to mold into an already existing data model or system.

Creating and mapping SAABs ELDIS database model took a few clicks on a step-by-step Visual Studio setup, after which Visual Studio created the entities and relations from the data model in SQL Server. The only complaint from Visual Studio was that a few tables didn’t have a primary key, which they shouldn’t have because the tables are supplementary tables to other tables.
4 Results

Test-Bed specifications are important to consider when observing the acquired results. Hence, table 4.6 states the Test-Bed specifications on which the experiments of this thesis were carried out:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Windows 7 Enterprise 2017, Service Pack 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel(R) Core™ i3-6100U CPU 2.30 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>8 GB</td>
</tr>
<tr>
<td>System Type</td>
<td>64-bit Operating System</td>
</tr>
<tr>
<td>Database (For NHibernate)</td>
<td>Microsoft SQL Server 2012 (64-bit)</td>
</tr>
<tr>
<td>SQL Server Developing Tool (For NHibernate)</td>
<td>Microsoft SQL Server Management Studio 2012</td>
</tr>
<tr>
<td>Database (For Entity Framework)</td>
<td>Microsoft SQL Server 2014 (64-bit)</td>
</tr>
<tr>
<td>SQL Server Developing Tool (For Entity Framework)</td>
<td>Microsoft SQL Server Management Studio 2014</td>
</tr>
<tr>
<td>.NET version</td>
<td>Microsoft .NET Framework 4.6.01055</td>
</tr>
<tr>
<td>IDE</td>
<td>Microsoft Visual Studio Community 2017</td>
</tr>
</tbody>
</table>

Table 4.6: Test-Bed Specifications

4.1 Basic Model Test

The first tests executed were carried out on the Student database entity using unit tests. As mentioned in section 3, simple CRUD operations are executed and query performance is observed for Stored Procedures, NHibernate and Entity Framework. Each method is carried out on one object, whether this object created, retrieved, updated or deleted. By using Visual Studio unit test project, the figure 4.7 illustrates the execution performance of each query.

![Figure 4.7: Querying Performance for the Basic Model](image)
Note that the configuration setup for NHibernate and Entity Framework is done only once in the [TestInitialize] before starting the querying and is not included in the time measurement of queries. The code snippet 4.8 is the configuration setup in the test class for NHibernate.

```csharp
[TestInitialize]
public void can_generate_schemeHBM()
{
    cfg = new Configuration();
    cfg.Configure();
    new SchemaExport(cfg).Execute(false, true, false);
}
```

Code 4.8: Configuration step for ORMs in TestInitialize

Another point to address in Entity Framework, the execution time for the Retrieve operation has two values: 120 ms (cold) and 29 ms (hot). Hot and Cold refer to whether entity framework has or has not loaded the data model. Hot meaning that data model is loaded and Cold that it is not.

4.2 ELDIS Model Test

The Stored Procedure query carried out in this project is used by SAAB in order to get all the connections that match the input. Although it is one query, almost all the tables are visited during the query execution. The Stored Procedure query is around 50 lines of SQL, which is rather large when compared to the previous basic model queries. However, the query only retrieves data and does not change any information in the database. The main focus of these tests is to calculate and compare how fast the query executes with Stored Procedure, NHibernate and Entity Framework.

NHibernate test

The mentioned query is run using NHibernate, all the tables, columns and relations in the data model had to be mapped into 20 classes in C# and equally 20 XML files for setting up the mapping environment. Afterwards, the query was implemented in a test class that reached around 70 lines of code in C#.

Unlike NHibernate testing, the Stored Procedure testing is always the same code without updating the code throughout the whole test, although the Stored Procedure query execution time differs slightly when repeated while testing in Visual Studio. That being said, NHibernate query is tested 3 different times, where every time a minor change in the code is made in order to potentially improve the querying performance of NHibernate. Those changes are later discussed in the Analysis section 5 of this report.

1. Normal ISession

The first round of the tests was run using the ISession which is the regular way to create the first level cache. The first level cache is when NHibernate creates a session factory that has a cache in it. Whenever a session is created, which is illustrated in code snippet 4.9, the queries are cached so reloading does not require visiting the database again. Therefore, the second time of retrieval for the same query will be faster than the first time [9].
The results of the execution time for the query, when running with four different number of executions, are shown in table 4.10. The number of executions with 1 means that the query was normally without any loops. For the other numbers, 10 and larger, a *For* loop was used to repeat the query multiple times. An important part to point out for NHibernate query is that the loop only included the lines of code where the query is executed and the configuration setup which takes most of the time.

<table>
<thead>
<tr>
<th>No. of loops per execution</th>
<th>NHibernate</th>
<th>Stored Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 sec</td>
<td>102 ms</td>
</tr>
<tr>
<td>10</td>
<td>2 sec</td>
<td>268 ms</td>
</tr>
<tr>
<td>100</td>
<td>6 sec</td>
<td>2 sec</td>
</tr>
<tr>
<td>10 000</td>
<td>8.04 min</td>
<td>3.43 min</td>
</tr>
</tbody>
</table>

Table 4.10: *I*Session Querying Performance for NH and SP using ELDIS Data Model

2. **IStatelessSession**

In this round, the code is changed a little, because the ISession that was used before is replaced for IStatelessSession. Unlike the regular ISession, IStatelessSession does not keep a lot of data in the first level cache. Therefore, more memory is unoccupied when IStatelessSession is implemented instead of normal ISession. This is mostly used when dealing with batch processing to improve the performance [9]. The code snippet 4.11 illustrates the change occurred in the second line of code.
The same type of the test is carried out with the same number of executions, and the results are shown in Table 4.12.

<table>
<thead>
<tr>
<th>No. of loops per execution</th>
<th>NHibernate</th>
<th>Stored Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 sec</td>
<td>91 ms</td>
</tr>
<tr>
<td>10</td>
<td>2 sec</td>
<td>257 ms</td>
</tr>
<tr>
<td>100</td>
<td>6 sec</td>
<td>2 sec</td>
</tr>
<tr>
<td>10 000</td>
<td>8 min</td>
<td>3.43 min</td>
</tr>
</tbody>
</table>

Table 4.12: IStatelessSession Querying Performance for NH and SP using ELDIS Data Model

3. Batch Size

The last round of tests was run but before the For loop in the NHibernate query, the batch size of the session was set to 50 that specifies how often the database is visited. The following code snippet 4.13 shows how the batch size is set.

```csharp
using (IStatelessSession mySession = sessionFactory.OpenStatelessSession())
using (ITransaction transaction = mySession.BeginTransaction())
{
    mySession.SetBatchSize(50);
    //Aliases; like variables
}
```

Code 4.13: Setting Batch Size

The following bar graphs, figures 4.14 and 4.15, illustrate the execution time for the tests after setting the batch size. Note that the execution time in the graph on the left-hand side is in seconds, and the graph on the right-hand side is in minutes. The graphs are divided for better visualization and illustration.
Entity Framework Test

The ELDIS query is mapped in Entity Framework (EF) using the built-in feature called “Model First” or “Database First”. This operation in EF is applicable when the database that the application should access, already exists and possibly has data too. This operation as such is valuable if there has been previous applications or iterations of such before and the same database has been used for this. As a result, “Model First” makes application integration with an existing database as simple as a few clicks on the initialization interface dialog. EF then takes care of mapping the database and relations to appropriate entities, maintaining all relations.

The tests shown in table 4.14 conducted for both the basic model testing and the ELDIS tests were with the recommended optimization tactics described in “Linq and Entity Framework. Some do’s and don't's”[5]. More specifically, only putting data in memory that is actually needed by using the Lazy Loading and Eager Loading techniques for caching relevant entities. Writing a longer single query that attempts to keep the query depth low coupled with using JOINs to attain maximum performance.

Normal Testing

<table>
<thead>
<tr>
<th>No. of loops per execution</th>
<th>Entity Framework</th>
<th>Stored Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>730 ms</td>
<td>102 ms</td>
</tr>
<tr>
<td>10</td>
<td>2 sec</td>
<td>268 ms</td>
</tr>
<tr>
<td>100</td>
<td>4 sec</td>
<td>2 sec</td>
</tr>
<tr>
<td>10 000</td>
<td>5 min</td>
<td>3.43 min</td>
</tr>
</tbody>
</table>

Table 4.14: Querying Performance for EF and SP using ELDIS Data Model.
In Code 4.15, a snippet of the test method for testing 10 000 calls of the same query is shown. The queries for 10 and 100 calls is the exact same with the only difference being the value assigned to “n”. The test method for a single call is structured the same way with the exception of removing the for-loop since there is only one call needed. The code snippet has has its variable names and values changed due to the NDA agreement.
5 Analysis

Basic Model Analysis
The results of the basic data model test show how each querying technology performs the CRUD operations. The execution time may differ insignificantly when the tests are repeated. However, that difference would not affect the analysis, since the execution time ratio among the querying technologies is still roughly the same.

NHibernate
When observing the figure 4.7, the operations Create and Retrieve for NHibernate are on the same level as the Stored Procedure execution time. On the other hand, the operations Delete and Update consume more time than both, Stored Procedure and Entity Framework. Of course, these results are only representing a simple database with one relationship between the two tables.

Nevertheless, the Delete and Update operations are not in favor for NHibernate for obvious reasons. When the operation Delete is executed, due to the way NHibernate is built, the entity that needs to be deleted has to be retrieved first from the database. Hence, two operations are performed in order to complete the deletion of an entity, these operations are Retrieve and the requested Delete. The same goes for the operation Update, where Retrieve is executed first to update the required entity then overwritten to the older version of the same entity in the database. Therefore, the conclusions that can be obtained from such results, based on the basic data model, state that NHibernate is not as functional as Stored Procedures and Entity Framework in deleting and updating entities in the database.

Entity Framework
As seen in the execution timetable, Entity Framework has very similar execution times to both NHibernate and SQL Stored Procedure queries. All except for data record retrieval. This is because of how Entity Framework compiles the model entities between every re-compile on the first query to the database.[10]

On the first query of data from the database, Entity Framework compiles the model in the entities that it uses. This way, the first query of data is slow as a result. All following queries after this are retrieved from the compiled model and thus the operations are quicker than a normal SQL query running normal CRUD operations.

ELDIS Model Analysis
As mentioned previously in section 4.2, the query tested is for retrieving an entity from the database, and no new updates occur to the existing information in the database.

NHibernate
Considering the tables and figures generally in section 4.2 regarding NHibernate, the execution time for the queries include also the configuration setup which takes up some time. To explain this in a practical way; the whole test method took 1 second for NHibernate, according to table 4.10, to get an object once for the first test round. The 1 second period is
divided into the configuration setup and the actual query execution, where the actual query took 278 ms and the rest was spent on the configuration setup.

**Entity Framework**

In regards to the ELDIS model testing, Entity Framework’s performance was not so slow compared to the already existing Stored Procedures even though the difference in execution time is quite significant. The results were expected since it was already known that an ORM implementation would be slower in execution.

On the first database call, Entity Framework caches the called tables and entities. This might affect the single execution more than any of the other tests. This was noticed because on some of the single run tests the execution time was a mere 415-440ms. As stated earlier regarding Entity Frameworks model compilation, introducing a small database entity load initializer may improve execution time due to preloading all entities of the database.
6 Discussion
The results of this investigation have pointed to the same thing that other studies and research have proven and speculated. ORM implementations do increase maintainability and flexibility based on our experience throughout the project. ORMs also increase the code developer’s efficiency in the sense that it enables the developer to focus more on the code and less on the data interaction object layer and the database structure. However, this comes at the cost of sacrificing a bit of performance which is also the case in other studies. Performance-wise, the similar results of Entity Framework and NHibernate are also reflected in other research studies as for example in Gergely Orosz’s article.[14] This case is also supported in another instance in Frans Bouma’s blog post.[15] One of the reasons of performance drawback is the caching as it is one of the most advanced element of an ORM which is initially used to improve performance. [16] Therefore, a thorough design and decision-making are required whenever cache is involved in the query transaction in order to increase the performance and not the opposite.

For NHibernate and Entity Framework, the approach used for mapping objects models to relational models in this paper are the same, called One Table per Concrete Class. This approach requires that every table in the relational model has an equivalent concrete class in the object model. The advantages of such approach are fast access to the data and objects manipulation since every table has one relation to its equivalent class. On the other hand, the disadvantage is if the structure of a table needs to be modified, the equivalent class needs also to be modified accordingly keeping in mind of all the possible subclasses.

There are other approaches that can be useful in other structures and scale of databases. One of these approaches is a much simpler way of mapping where one class is needed for all the tables mapped, which favors a simpler database. Another approach is Generic Schema which is more flexible for databases with a larger scale.

NHibernate
Since NHibernate is an ORM tool, there are different approaches to implement a query. As an example, there are Get and Load which are two different methods to retrieve data from the database using an ID. In the basic data model tests, Get always visits the database or the cache to get the requested object, and the return value would be the object or null if the requested object does not exist. In contrast, the Load does not visit the database and always loads a proxy of the requested object or throws an exception due to the contract that exists [13]. Another method to query with a more sophisticated syntax, which is used in the implemented test, called QueryOver API that belongs to NHibernate natively and provides lambda expressions to support the ICriteria API [13]. These various methods exist to assist the developers to achieve better non-functional requirements depending on how their projects are set up and aid them to suite numerous cases.

Entity Framework
The main problems regarding usage of Entity Framework are when and where to load entities and related entities. Where and when are paramount for an efficient code and execution. Entity Framework natively has a mechanism called “Lazy Loading”. Lazy Loading operates in the way that when an entity or a collection is automatically loaded from the database on the first main entity access is requested. This can be achieved by, for example code snippet 6.1, adding an ICollection of a type and add it in the main entity class as a variable with a getter and setter. This informs Entity Framework that the entities that are of the type in the ICollection are related entities and should be loaded as well together with the main entity. A
way to remove the Lazy Loading feature is to simply remove the word “virtual” in the “public virtual ICollection”.

This can in many instances be beneficial in order to not accidentally or unwillingly query the entire database or all related entities for a specific main entity. For example, this would be detrimental for functions where many records are related to the main entity because all related entities would be loaded at the same time and serialized.[11]

```csharp
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public string Tags { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}
```

Code 6.1: Related entity ICollection example

In regards to our main research questions presented in section 1.5, we can now discuss them in table 6.2 and how the results compare and apply to the questions.

<table>
<thead>
<tr>
<th>RQ1.</th>
<th>What are the advantages and disadvantages of the Stored Procedures compared to ORM with the current database design model in terms of queries performance?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>The advantage of ORMS compared to Stored Procedures is flexibility and maintainability.</td>
</tr>
<tr>
<td></td>
<td>• For flexibility, the necessity of writing raw SQL queries is eliminated which enhances the development process flow regarding database interaction.</td>
</tr>
<tr>
<td></td>
<td>• For maintainability, ORM is a new technology compared to Stored Procedures, therefore, has the ability to be better maintained and altered to fulfill the maintainability definition by IEEE.[1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RQ2.</th>
<th>Concluding from the previous research question and considering flexibility and maintainability factors, where and when to use stored procedures or ORMs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>From the answers obtained in research question 1 and experience obtained flexibility and maintainability, a Stored Procedure implementation will always be more efficient when querying the database. This of course under the assumption that the SQL query string is optimized at the same level that on ORM query is. An ORM implementation, however, is more resource intensive and slower than an SQL query. On the other hand, if a company is running a server with sufficient and powerful hardware, Entity Framework can for instance cache big parts of the database into memory and have it more readily available for access compared to a Stored Procedure implementation which would need to retrieve all data when requested.</td>
</tr>
</tbody>
</table>
The main question of whether to use an ORM or Stored Procedure implementation is cost and time. If the development team has time and resources, a Stored Procedure implementation will yield the best performance. However, an ORM implementation can be easier to implement especially using Entity Framework. However, the cost arises hardware needs for making Entity Framework it's most efficient since it will need a big memory cache to load database entities into.

Table 6.2: Research Questions Answers
7 Conclusion
The main research results tell us that an ORM implementation is slower than an implementation using Stored Procedures. The main reason being that ORMs query data in a different way than a Stored Procedure. This through caching and other methods that use more resources and thus time.

Our findings are represented in the form of tables and graphs where the efficiency is compared to the implementations, using execution time as the main test criteria. This paper and research are important for all developers who want to know in what aspects a certain data access method is advisable and in others, unadvisable. Querying data from databases is always used when there is a need for data persistence and as the data sets become bigger and more complex, it is important to know the benefits and drawbacks of each data access model’s nature. This to ensure greater performance but also security.

Although this study is case specific for SAAB and their application and system, the contents and results of this study will also adhere to other areas where a database access model is considered and whether or not it should be implemented. This to ensure the best solution for each individual system and its features.

Our results could have been improved by broadening the test criteria, but this was not the focus of this study. Our results could, however, have been improved through more tests and thus getting a better median in the test values. However, minuscule the change in test results would be. Further testing and tweaking of the code and database models could have been investigated deeper. The database model tweaking would, however, have had to be limited since SAAB’s database structure is substantial in size. A way to bypass this issue would have been to create another separate database structure optimized for entity framework and NHibernate. This, however, would take a lot of time and be hard to fill with test data in a sufficient yet varied way that would correspond to a real database.

7.1 Future Research
The most prominent question raised during this project is how ORMs perform on a wider scale. This research was limited to the scope of SAAB’s domain and operations. The scale of this project was rather small in comparison to more complex and probably larger databases with millions or perhaps hundreds of millions of entries and rows. Also, the scope of this project was only focused on a database built upon Stored Procedures architecture and design. In other words, the database was built almost two decades ago and the architectural structure is suited best for an old relational database to be queried by Stored procedures. Nowadays, the ORM exists to serve best a different architecture of the database with different ways to set up relations among the tables of a database. An ORM architecture and design-based database were not covered in this project, therefore, information regarding an ORM-based database is still to be explored.
References