Performance analysis for CRUD operations in relational databases from Java programs using Hibernate

Alexandru-Marius Bonteanu
Department of Computer Science
"Politehnica" University of Bucharest
Bucharest, Romania
alexbonteanu22@gmail.com

Cătălin Tudose
Learning Management Department
Luxoft Romania
Bucharest, Romania
catalin.tudose@gmail.com

Ana Magdalena Anghel
Department of Automatic Control
"Politehnica" University of Bucharest
Bucharest, Romania
ana.anghel@upb.ro

2023 24th International Conference on Control Systems and Computer Science (CSCS) | 979-8-3503-1339-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSCS59211.2023.00109

***Abstract*— Data is information or a collection of information about something. Almost every application nowadays needs to persist data so that they can keep track of their state in case of any trouble that might appear. Databases are the main piece of an application for persisting data and they can be of two types: SQL or NoSQL. SQL (Structured Query Language) is a query language used to create, read, update, or delete (CRUD operations) the information in most relational databases. Data is structured in a set of tables in the relational database model, with rows as entries and columns as specific attributes of the entry. Over the years, new relational database management systems appeared. A comparison between some of the most used relational database management systems tied to a Java application using Hibernate is worth analyzing in the given context. The evaluation is done on timing for each CRUD operation by using Java Microbenchmark Harness, because these are the most used operations, for thousands and hundreds of thousands of entries. Depending on the majority type of operation, switching the database management system might help.**

Keywords—framework, relational database, object-relational mapping, Hibernate, Java Microbenchmark Harness, MySQL, Oracle, SQLServer, PostgreSQL, performance

I. Introduction

Accessing relational databases from Java applications is the core of the experiment. Any application may need to persist its data and this thing can be done with an Object Relational Mapping (ORM) which maps a Java class on a database table [1].

There are a lot of relational database management systems to be used when you start a new application, and this makes the choice harder. Each of them has its own advantages and disadvantages, at the end coming to a trade-off, as almost everything in the computer science world does.

The problem that this study wants to solve is profiling combinations of the Hibernate framework with the most popular relational database management systems (RDBMS). The main aspect we'll cover is the speed and complexity of the code for doing batches of CRUD operations. The particularity of the research is that it does not focus only on database performance but on the interaction between the ORM software and the database. Consequently, it will be enough that the database contains a limited number of tables, the investigated overhead coming from the mapping operations between the objects in the memory and the tables in the database.

There are several articles about what RDBMS to choose from, with pros and cons on each of them, but with no practical evaluation. Mixing RDBMSs with frameworks is something that might be of interest to any developer[2].

The proposed solution is to run a medium-complexity application, using the Hibernate framework with different RDBMSs, and to test the time it takes to do different amounts (thousands to hundreds of thousands) of each CRUD operation. Java Microbenchmark Harness will be used to run a Java Virtual Machine (JVM) warmup iteration before the main run.

This way, it can give us a pretty clear look at how these technologies work better for optimized access to a relational database in a Java application. The main purpose is to determine how an application can retrieve data faster and how easily is this aspect achieved.

II. Background

Relational databases are a key concept when talking about an application because they help us persist information. Persisting information is important because, if the application stops running (it is interrupted or finishes its execution), data should not be lost. Losing data can create great discomfort for the end users.

In relational databases, the data is organized in tables. Each entry in the table should have a unique identifier called Primary Key (PK). Columns are specific data attributes defined when a table is created. Related tables are connected with Foreign Keys (FK). The structure is described deeply in the „Relational database theory" book[3]. For example, we have a Car table with a unique number as the PK, and Color, Power, Number of Seats, and Manufacturer as its attributes. But the Manufacturer is also a table that has a unique number as a PK, plus Name, Telephone, and Address as attributes. To keep data structured, the Manufacturer column in the Car table will hold the PK of the corresponding entry in the Manufacturer table. This means that the manufacturer identifier will be an FK in the Car table.

There are four categories of commands:

- data definition language (DDL) – used to define and modify the database and tables structure

979-8-3503-1339-0/23/$31.00 ©2023 IEEE
DOI 10.1109/CSCS59211.2023.00109

- data query language (DQL) – used to interrogate the database tables for specific information (the SELECT command)
- data manipulation language (DML) – used to insert, delete and update the database tables
- data control language (DCL) – used to control user's access to the database

The operations on which the implementation is focused are the DQL and DML as they are the main operations of an application. The SELECT command is used for the read operations, while INSERT, UPDATE, and DELETE are used for creating, updating, and deleting information from the database.

Object Relational Mapping has some advantages over the JDBC that make the developer's job easier: hides the SQL interaction, offers development using objects instead of database tables, same model implementation for all RDBMSs, less code written for the same job done and works on top of JBDC.

The database connection in Java applications is done through a JDBC driver. This driver is specified by the database management system and it needs to be present in the application's configuration. A specific dialect also needs to be set up by the developers when configuring the database connection so that the queries are created in the right language.

There are many Relational Database Management Systems (RDBMSs) in the SQL world, but four of them are very popular choices:

- MySQL
- PostgreSQL
- Oracle SQL
- Microsoft SQL Server

The syntax for these is slightly different, but we can do all the database operations we want using any of them. Each of them has its own tool to monitor and modify data from a graphical user interface.

In the end, a comparison will be done with the evaluation of a medium-complexity Java application. Joins will be a key element in our application because this is a key operation when working with a relational database. A join is basically an operation between two tables that have an attribute in common. Let's consider the previous example with the Car and the Manufacturer. If the developer wants to get from the database details about one car, but also its manufacturer phone number, then he simply considers the phone number from the Manufacturer table entry whose ID is corresponding to the Manufacturer ID from the Car table.

Optimization techniques might arise from the testing that will be done. If an application runs slowly, then the database management system could be changed to obtain a better performance.

There are lots of articles on the Internet, explaining what ORM framework should the developer choose and why. The same thing is true about RDBMSs, a simple search will tell us the differences between the most used ones.

Haseeb Yousaf compares in his thesis "Performance evaluation of Java Object-Relational Mapping tools" [4] some ORM frameworks like OpenJPA, EclipseLink, and Hibernate, on a Ubuntu operating system. He performed five queries on a table: read by ID (PK), read by three different type attributes, and one combined read based on two attributes. The results show that Hibernate is the fastest of those three frameworks from 10000 records up to 160000 records, while OpenJPA is the slowest. EclipseLink is getting closer to Hibernate as the number of records increases.

Another work by Ismail Hossain [5] compares the most popular five relational database management systems: MySQL, Oracle, PostgreSQL, SQLite, and Microsoft SQL Server. The first test he does is measuring the installation time for each one of them. Oracle and SQL Server seem to be the laziest ones, while SQLite and MySQL's installations take less than five minutes. The second test he does is measuring query times for every CRUD operation on different numbers of entries. From his work, it seems like Oracle is the laziest when coming to reading information from the database, but pretty good at updating and deleting. MySQL is far from Oracle's update and delete performance with very big times on executing these operations. PostgreSQL seems to be the best choice for a developer, according to Ismail Hossain's work. The other two, SQLite and Microsoft SQL Server can be a second option when starting a new application.

Both these works mentioned above make a comparison at the same level of technology. One is done to test different ORM frameworks, and one to measure which relational database management system is faster.

This study will try to make some combinations of all the technologies included in the comparisons mentioned in the above studies and achieve a more complex evaluation based on the RDBMS, and the Hibernate framework, working together to optimize the access to a relational database from a Java application.

Similar work, but using a different programming language (.NET) was done at the University of Oradea [6], where some teachers compared the execution times and the memory usage of an application, based on three distinct ORM frameworks combined with SQLServer on the database side.

The suite of technologies for evaluating how a developer can optimize access to a relational database is the following:

- Java programming language
- Oracle RDBMS
- PostgreSQL RDBMS
- Microsoft SQL Server RDBMS
- MySQL RDBMS
- Hibernate
- Java Microbenchmark Harness
- Maven – Java package manager

Java is a globally known programming language [7], which offers the possibility to develop an end-to-end application, both the server side and the graphical user interface used by the application's clients. Being a popular language, the number of libraries and frameworks developed for it and the number of developers are considerable. This means a simple search on Google may quickly guide a beginner developer.

Oracle is a popular RDBMS developed by Oracle Corporation. Many important applications like Netflix,

LinkedIn, and eBay, with a very big amount of data, have this technology in their stack. Procedural language for SQL (PL/SQL) is Oracle's wrapper over classical SQL. As its name says, this language has support for procedural operations such as looping, decision-making and declaring variables. Another interesting feature is that it is platform-independent.

PostgreSQL is an extension of the SQL language. It has support for different data types like Numeric, String, JSON, XML, and many others including customizing your own. As mentioned, the developers can add non-relational data, which is a key feature. This relational database management system is mostly preferred by Python developers because they interact very well, due to the psycopg database adapter, but it can also be used in Java programming.

Microsoft's solution for relational database management is Microsoft SQL Server. It has its own dialect called Transact-SQL (T-SQL) which allows transaction leverage and proper error handling. Procedural programming is also part of the wrapper. For deleting data, the user can make a query using joined tables in the FROM clause. The UPDATE command also supports joined tables and the FROM clause.

The open-source software provided by Oracle, regarding the management system for a relational database, is MySQL. It is considered the most popular by many developers because it is easy to use, is free, and has a large community. The installation process is fast, it can be used on almost every computer running an operating system without a high need for resources to do its job.

Hibernate is an ORM framework that is free and open source. It is the most extensively used framework for putting the Jakarta Persistence API specification into practice. The application and the database communicate through a session in which CRUD operations are carried out. It provides transactions, which are generated from sessions, that must be started and committed after all activities are done for changes to be saved in the database.

Java Microbenchmark Harness (JMH), is a tool for developing Java microbenchmarks. The tool was designed by the same team that invented the Java Virtual Machine, so they understand what the JVM is doing and how to effectively implement benchmarking. Basically, JMH handles JVM warm-up and code-optimization pathways, making benchmarking as straightforward as possible.

Maven is an integration tool that is responsible to combine the basic Java Development Kit with other frameworks. The dependencies are declared in an XML file. Maven has a remote repository where the developers can find a lot of JARs (Java Archives) containing already implemented functionalities. These dependencies are downloaded by Maven to a local repository which makes them usable in the application under development.

All these technologies working together lead to the creation of a quality solution. The right selection of technologies before starting a new application is crucial. The developer must be aware of what he can and cannot do with the chosen ”weapons”.

III. Architecture

The main goal of this implementation is to help developers and architects to select to right choice when it comes to the relational database management system in Java applications.

Therefore, the described solution was implemented by creating the configuration file for the Hibernate framework and by changing the database connection's details. Test cases were created for each combination and CRUD operation.

The pros and cons of the different approaches that the implementation is intending to develop, will be examined after a series of tests and proper evaluation. This process will be described in the evaluation section, where all the measurements will be documented with graphics and numerical results.

The database connection and dialect are pretty easy to change from one relational database management system to another. The configuration file help with this change because that is the place where the database settings are kept. The presented idea wants to switch only three attributes, to achieve proper database comparison. These are the dialect, the database driver, and the URL. This analysis targets Java specialists who want to improve their vision of how things work when communicating with a database and how they can be improved to obtain the best performance.

As described earlier, the analysis considers four main RDBMSs: MySQL, Oracle, SQLServer, and PostgreSQL.

The ORM framework is responsible to map Java objects on database models. It controls the flow until data reaches the database.

The relational database management system takes care of the data once this is placed in a database. Each RDBMS has different optimization techniques to simplify data retrieval and data writing. Although it can have a significant impact on efficiency, it is the easiest to replace due to standard configuration files.

RDBMSs offer specific drivers to allow developers to connect to the database. The communication is done through this driver, but it also needs a language that the RDBMS understands, as the code written will be in Java, not SQL related. The solution to this problem is to specify the dialect. This has a suggestive name, it refers to the SQL commands in which the ORM framework should translate the operations before sending them to the RDBMS.

These two configurations, along with the path to the database (URL), are described in a configuration file. Every framework has a different way of specifying these properties, but all of them need to declare this.

End-to-end component communication is obtained by combining all these parts. It starts from the Java application, which gets the framework JARs from Maven and communicates with the database by declaring a set of properties.

To be able to run all the test cases, the communication needs to be functional on the entire flow. If something wrong happens, then the results would not be relevant. It also works bi-directionally, because it can read from or write to the database, as the purpose is to test all the CRUD operations.

The evaluation will be done by implementing a medium-complexity betting application in which the main entities will be: tickets, bets, and matches. These entities will be described in the implementation section.

The timing for each CRUD operation is the most important aspect. The testing will measure the time for each combination

described, making the duration comparison possible. With JMH's help, there will be two iterations: one for warming up the JVM, and one after the warmup phase is completed.

After collecting all the results, these ones will be centralized in tables, to facilitate the analysis. Warmup time results will also be analyzed. Based on that, the solution will provide relevant information to the audience by creating some more attractive graphs with the numbers obtained.

IV. Solution Implementation

The main idea behind the implementation was to create a medium-complexity application. To make things more interesting, it was decided to have a story in the background. As the focus of the research is on the overhead of the ORM and the interaction between the ORM and the database, we'll use a database with a limited number of tables, but generating many records.

The application is focusing on a soccer betting service model. This means that testing is done by creating a different number of tickets, each with its bets and corresponding matches, and simulating their behavior in relationship with the database and the ORM framework.

There are three main entities of the project, the ones described above, on which the CRUD operations are tested:

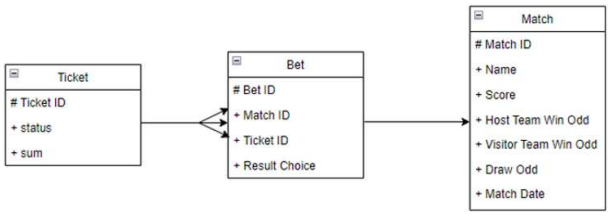

Fig. 1. The Data Model used for the experiments

The figure above presents the data model with the relationship between entities. One ticket can have multiple bets on it (one-to-many relationship), and one bet is based on one match (one-to-one relationship).

These attributes are needed to get a more realistic simulation of a real case scenario. Their number varies from one entity to another, but it is a minimum for describing a betting service data model.

It was essential to configure a SessionFactory for the Hibernate implementation. Hibernate obtains Session instances from this factory object to build transactions. It contains a configuration file named "hibernate.cfg.xml" in which it declares the connection information and setup properties. This file is used to construct the SessionFactory. The flow continues by generating transactions from the session in which the operations are carried out.

For each CRUD operation, a transaction is initiated, marking the time when it was started. Then the operation is executed and afterward the finishing time is set. The difference between these two moments represents the execution time for each operation, measured in milliseconds.

What is special about this implementation is that for the read operation, it is needed to create a query to get the desired entities. This is achieved by using the `CriteriaBuilder` and `CriteriaQuery` classes. On the last one, there is a method `getResultList()` which retrieves all the entities matching the query.

The other three operations, create, update and delete have dedicated methods on the `Session`: `persist`, `update`, and `remove`. These built-in methods help a lot the developers, by reducing the amount of code needed to write to obtain the specific functionalities mentioned before.

Another common part of the project is the data mocking process for the test cases. It implies a builder method for a match used in a builder method for a bet, which is finally used for a builder method of a ticket. On top of these, a method for creating a variable number of tickets was created for easier testing of different bulks of data.

Different technologies which can describe a complete flow between a relational database and a Java application have been selected to test their efficiency. They are relevant in many production applications as the comparison should be of interest, with actual and popular components.

All of the ORM frameworks are based on the JPA standard. This enables easy mapping from a Java class to a database table, by providing some annotations which do this task in the background.

```
@Data
@Entity
@Builder
@Table(name = "tickets")
@AllArgsConstructor
@NoArgsConstructor
public class Ticket {
    @Id
    @GeneratedValue(strategy =
                    GenerationType.IDENTITY)
    private Long id;

    private TicketStatus ticketStatus;

    private Double sum;

    @OneToMany(cascade=CascadeType.ALL)
    private List<Bet> bets = new LinkedList<>();
}
```

In the code above, describing a betting ticket, the Jakarta Persistence API [8] is present through the following annotations:

- `@Entity` -> used to mark the class as data that can be persisted in the database
- `@Table` -> used to mark the name of the corresponding database table
- `@Id` -> used to mark the primary key for the tickets table
- `@GeneratedValue` -> used to describe how the primary key is populated
- `@OneToMany` -> used to describe a one-to-many relationship between the ticket and the bet entities

There are four more annotations in the code above, which are part of the Lombok library. Their purpose is to create constructors, getters, setters, and a builder for this class, and to avoid boilerplate code. The other two data entities, for bets and matches, are similar to this one.

The Hibernate design incorporates a file named "hibernate.cfg.xml" that defines the configuration:

```
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">
            com.mysql.cj.jdbc.Driver
        </property>
        <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/bettingDB
        </property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MyS
QL8Dialect</property>
        <property
name="hibernate.connection.username">root</propert
y>
        <property
name="hibernate.connection.password">root</propert
y>
        <property
name="hibernate.connection.pool_size">50</property
>
        <property name="show_sql">false</property>
        <property
name="hibernate.hbm2ddl.auto">create</property>
    </session-factory>
</hibernate-configuration>
```

This file contains the URL to the database, user details to make the connection, and the RDBMS driver. The dialect is also specified here, to have a common language between the application and the database. The example above is for the connection with a MySQL database.

V. Solution Evaluation

To evaluate the solution, the environment needed to be set up. The configuration of the laptop on which the duration of the CRUD operations was measured is the following:

- CPU: Intel i7 – 6700HQ @ 2.6 GHz
- RAM: 8 GB
- Operating System: Windows 10 Pro 64-bit

The prerequisites were to install all the technologies mentioned in chapter 3 and to find a way to properly measurement of the execution time.

Java Microharness Benchmark provides an annotation `@State` used to mark a class where variables that you do not want to be part of the benchmark measurement, will be initialized, such as the ticket list. The proposed solution uses the System.nanoTime() method to get the exact time before and after each CRUD operation, but these are also methods from the state class. Another thing is the `SessionFactory` and `Session` initialization. The difference between the starting point and the ending point of an operation divided by 10^6 gives the duration in milliseconds. Other operations that took place in the tests were not considered when the measurements were calculated as they were part of the state class.

Hibernate's logs were removed to reduce the execution time because outputting them in the console is a very time-consuming operation. All CRUD operations were executed sequentially, in one single benchmark which was configured to run with one warmup iteration followed by a classic one. The order of the operations was the following: create the entries, update, read, and finally delete them.

The tests that were run on the implementation of a betting service were meant to test the duration of each CRUD operation for a various number of tickets, containing one bet on one match. These numbers were: 1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000 and 500000 tickets.

The update operation changed the ticket's status, the bet's result choice, and the name of the match to keep the operations running on all three database tables, enhancing the application's complexity.

These tests run on each combination of Hibernate framework and relational database system, which led to 4 different scenarios (1 ORM framework and 4 RDBMSs). The interpretation of the results might give developers a better view of how their application can be optimized to achieve better timings, even if it is a small application (10-20k entries) or a bigger one (up to 500k entries).

Some similar work on a single database table was done by Tudose Cătălin and Odubăşteanu Carmen [9], where the same framework plus Hibernate and Spring Data JPA were compared on a MySQL RDBMS. The plan was to extend this research paper with a more complex application, more relational database management systems, and by increasing the number of entries to hundreds of thousands. An additional element was introducing a benchmark to warm up the JVM before the experiments. The warmup results were also taken into consideration and analyzed, to see if they improved.

The results for each combination tested were translated into the tables below from which corresponding graphs by CRUD operation were created for a better overall view. Further interpretation will be done on each pair to check their pros and cons.

TABLE I. MySQL – Hibernate Warmup Execution Times

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	2580	134	1653	1291
2000	3883	163	2457	1922
5000	7981	174	3928	4483
10000	14121	200	7845	8437
20000	24161	214	17527	17779
50000	49300	272	31505	39011
100000	90727	395	62460	79682
200000	154690	519	132628	162276
500000	422960	754	452453	536749

TABLE II. Oracle – Hibernate Warmup Execution Times

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	2900	176	1366	1728
2000	5156	208	2127	2467
5000	9563	237	3660	6123
10000	15538	369	6736	13117
20000	31089	535	12600	32559
50000	65035	901	29441	112994

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
100000	124436	1416	57427	351471
200000	242417	2295	111213	1208644
500000	599872	5143	275656	6749529

TABLE III. SQLSERVER – HIBERNATE WARMUP EXECUTION TIMES

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	2249	157	1194	2519
2000	3612	172	2031	6098
5000	7056	191	3325	27331
10000	11151	215	6542	75966
20000	19603	235	11847	93770
50000	45659	339	30036	645622
100000	85737	351	53012	978065
200000	162910	459	106024	2931522
500000	413812	668	303875	18919726

TABLE IV. POSTGRESQL – HIBERNATE WARMUP EXECUTION TIMES

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	1272	139	794	1010
2000	2049	171	1298	1703
5000	4244	198	2451	5971
10000	7075	213	4028	17406
20000	12756	226	7514	57645
50000	30374	264	18040	299375
100000	58557	314	36891	1093245
200000	112398	387	72108	4350143
500000	280235	566	179190	27096855

TABLE V. MYSQL – HIBERNATE EXECUTION TIMES

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	1356	4	791	970
2000	2232	13	2042	2488
5000	4195	19	3251	4065
10000	7454	21	6414	7951
20000	20723	25	18556	20381
50000	37630	62	31941	37681
100000	79584	120	65103	75940
200000	158046	212	131501	160246
500000	462010	515	447930	416474

TABLE VI. ORACLE – HIBERNATE EXECUTION TIMES

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	2087	16	654	1150
2000	3613	42	1245	2374
5000	6525	76	2922	5934
10000	12877	173	5846	12888
20000	25958	286	11714	32497
50000	61604	563	28781	116607
100000	122788	1080	57296	368170
200000	237372	2001	109526	1330209
500000	598497	5056	286199	7009015

TABLE VII. SQLSERVER – HIBERNATE EXECUTION TIMES

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	1144	4	736	2539
2000	2313	8	1347	6457
5000	4725	12	3118	35963
10000	8986	16	5528	64561
20000	17244	25	11309	98772
50000	40057	40	27389	308409
100000	76050	64	52283	904024
200000	158971	136	104486	3263787
500000	401552	260	274256	16404632

TABLE VIII. POSTGRESQL – HIBERNATE EXECUTION TIMES

Number of operations	Execution times (ms)			
	Create	*Read*	*Update*	*Delete*
1000	802	5	447	725
2000	1430	10	761	1625
5000	2962	14	1796	6317
10000	5612	16	3538	18831
20000	11625	18	7212	60457
50000	27041	36	17647	317984
100000	57995	67	40528	1228630
200000	118198	129	81967	4296717
500000	291326	320	202372	27254448

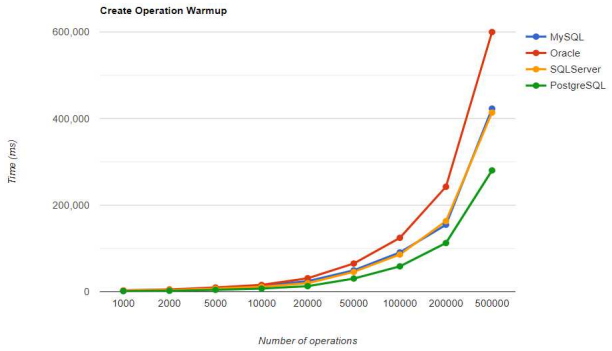

Fig. 2. Create warmup execution times

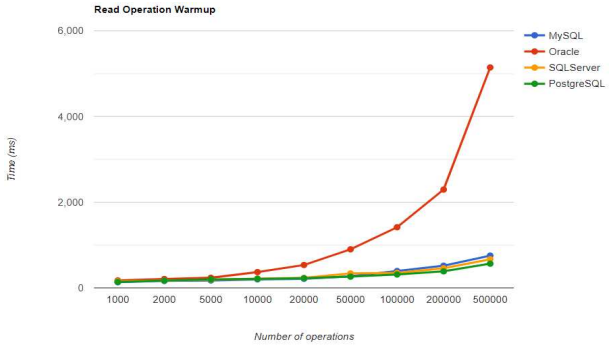

Fig. 3. Read warmup execution times

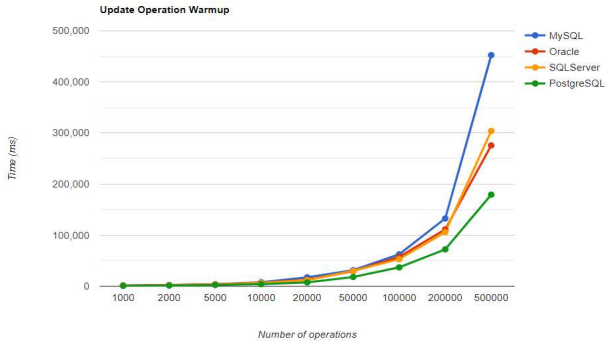

Fig. 4. Update warmup execution times

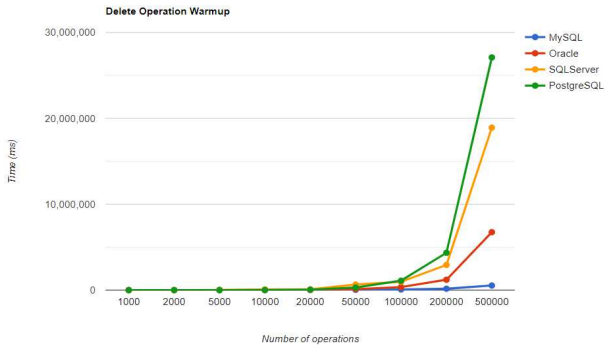

Fig. 5. Delete warmup execution times

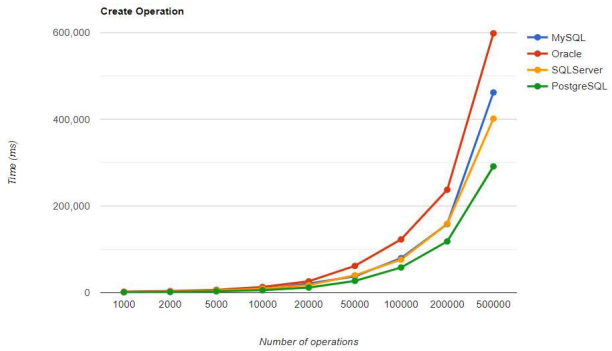

Fig. 6. Create execution times

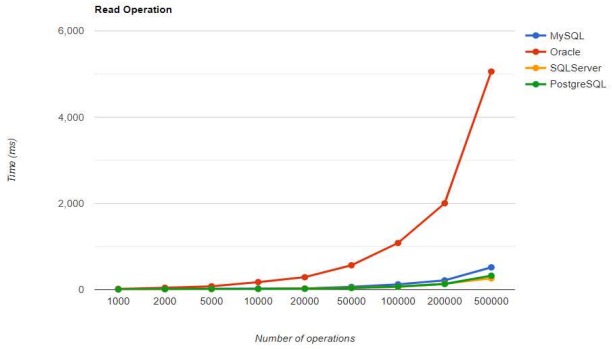

Fig. 7. Read execution times

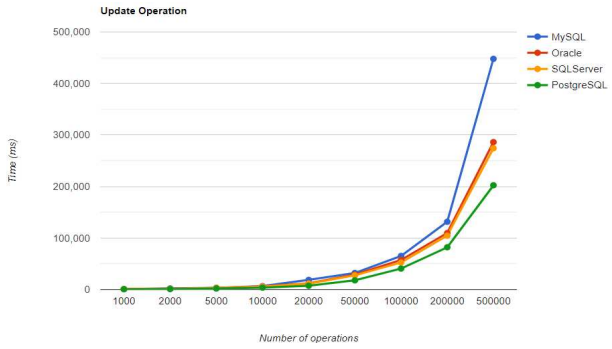

Fig. 8. Update execution times

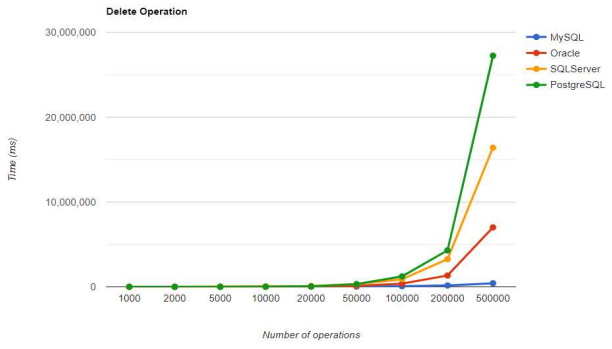

Fig. 9. Delete execution times

VI. Conclusions and Future Work

The experiments summarized in this article intended to provide a foundation for the multi-criterial decisions made by engineers working with Java and databases about the

frameworks and servers to select, according to the specificity of the projects. They also represent a solid ground for detecting critical time-consuming operations, the subject of optimizations, as we would like to address in future work.

As the purpose of this research is to investigate the overhead of the object-relational mapping operations, we generated a large number of records, persisted in a limited number of tables.

Regarding the warmup execution times, PostgreSQL has the best timing on the CREATE and UPDATE operations for each number of entries tested, but it is terrible when deleting more than 100k entries. Reading is also great with PostgreSQL, but for over 50k entries. For fewer entries, MySQL is the best choice. The most valuable point for MySQL is the deletion, which is more than 12.5 times faster than the second RDBMS (Oracle), for 500k entries run.

Oracle looks a bit lazy, especially the READ operation, almost 7 times slower at the biggest run than MySQL, which is placed in the third position. Its strengths are updating and deleting for big numbers of entries, making it the second-best choice for these operations. Oracle is the worst solution for less than 50k entries, at every operation.

Ranking second for the CREATE and READ operations, SQL Server is recommended for a new application, but only if deletions are limited, as it performs almost as badly as Oracle for this operation.

The most visible impact the warmup had on the second run is at the READ operation. It runs almost instantly for several entries smaller than 50k and reaches a maximum of half a second for 500k entries, except the Oracle RDBMS, but a big improvement can be noticed here also.

Even if the execution times after the warmup iteration are expected to be lower, this is true only for the CREATE operation measured on the Oracle and SQLServer RDBMSs. This operation has better results on MySQL and PostgreSQL too, but not for more than 100k entries.

On the MySQL RDBMS, the UPDATE and DELETE operations have almost the same performance, with or without warmup. The same thing is available for deleting on an SQLServer RDBMS or a PostgreSQL one.

Updating times are reduced for less than 50k entries with MySQL, 100k entries with PostgreSQL, and 500k entries with Oracle. The most visible change overall was for Oracle, which reduced the durations for almost every number of entries.

The final results after the second iteration made some slight changes in the comparison of the performance for each operation, by each RDBMS. However, PostgreSQL still has the best timing for creating and updating entries, and also the worst overall timing when it comes to deletions.

SQLServer provides the best reading, over-passing PostgreSQL and MySQL, which falls to the third position after it was first in the warmup phase. Oracle has some serious time reduction for less than 200k entries, but at the largest run, it has almost the same time as before, a bit over 5 seconds.

The best deletion is taken again by MySQL, with an even bigger difference from the warmup comparison: almost 17 times faster than the Oracle performance. But something new for this RDBMS is the poor performance when it comes to updating entries because it is the worst solution of all four.

A great overall performance is offered by MySQL. It has incredibly good execution times, despite having a slower performance than PostgreSQL or SQLServer for creating, reading, and updating entries, but it saves a lot of time on the delete operation (over 65 times faster than PostgreSQL).

SQLServer and PostgreSQL have the best execution times on the update operation. Reading the entries from a database is fast in every combination. SQLServer has the best reading, while Oracle has the worst one. However, the difference between these two is under 5 seconds, for 500k entries. Oracle has the slowest insertion of all four RDBMSs.

The performance analysis of different RDBMSs in combination with the Hibernate framework intends to help developers that start a new application, but also the ones not satisfied with their current application's performance.

SQLServer is the best RDBMS when coming to reading data, but with quite a poor performance on deletion. Oracle seems to be worth trying when it is needed to work with even more entries than this article's tests have tried because it has a better performance as the number of entries is increasing.

Looking forward to extending the results, or checking how combinations behave in different circumstances, the research would not end here. It is planned as future work to run the tests using other popular frameworks like JPA or Spring Data JPA and detect the critical time-consuming operations that may be optimized at the level of the framework code.

Another interesting factor is the timing of the execution of methods from frameworks. This way, the most time-consuming calls will be detected and investigated. For this kind of work, JMeter, an Apache solution for performance testing, can help.

Last, but not least, there are a lot of drivers for the same RDBMS. It will be interesting to see the changes when switching between these drivers. Additionally, introducing indexes and composite primary keys in the data model and looking into the execution differences could extend the research.

REFERENCES

[1] Keith, M., & Schnicariol, M. (2009). Object-relational mapping. In Pro JPA 2 (pp. 69-106). Apress.

[2] Sumathi, S., & Esakkirajan, S. (2007). Fundamentals of relational database management systems. Springer.

[3] Atzeni, P., & De Antonellis, V. (1993). *Relational database theory.* Benjamin-Cummings Publishing Co., Inc.

[4] Yousaf, H. (2012). Performance evaluation of java object-relational mapping tools.

[5] Ismail Hossain, M. M. (2019). Oracle, MySQL, PostgreSQL, SQLite, SQL Server: Performance based competitive analysis.

[6] Zmaranda, D., Pop-Fele, L., Gyorödi, C., Gyorödi, R., & Pecherle, G. (2020). Performance Comparison of CRUD methods using NET Object Relational Mappers: A Case Study. International Journal of Advanced Computer Science and Applications, 55-65

[7] Arnold, K. a. (2005). *The Java programming language.* Addison Wesley Professional.

[8] Tudose, C. (2023). Java Persistence with Spring Data and Hibernate. Manning.

[9] Tudose, C., & Odubăşteanu, C. (2021). Object-relational Mapping Using JPA, Hibernate, and Spring Data JPA. *Proceedings CSCS-23.* Bucuresti.