

# A framework to Optimize ORM Data Retrieval using Indexed Search Engines



Author

Tassaddaq Sultan

FALL 2015-MS-15(CSE) 00000119224

MS-15 (CSE)

Supervisor

Dr. Farooque Azam

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING  
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING  
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

July 2019

A framework to Optimize ORM Data Retrieval using Indexed Search  
Engines

Author

Tassaddaq Sultan

FALL 2015-MS-15(CSE) 00000119224

A thesis submitted in partial fulfillment of the requirements for the degree of  
MS Software Engineering

Thesis Supervisor:

Dr. Farooque Azam

Thesis Supervisor's Signature: \_\_\_\_\_

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING  
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING  
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,  
ISLAMABAD

July 2019

## DECLARATION

I certify that this research work titled “*A framework to Optimize ORM Data Retrieval using Indexed Search Engines*” is my own work under the supervision of Dr. Farooque Azam. This work has not been presented elsewhere for assessment. The material that has been used from other sources has been properly acknowledged / referred.

---

Signature of Student

Tassaddaq Sultan

FALL 2015-MS-15(CSE) 00000119224

## **LANGUAGE CORRECTNESS CERTIFICATE**

This thesis is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University for MS thesis work.

---

Signature of Student

Tassaddaq Sultan

FALL 2015-MS-15(CSE) 00000119224

---

Signature of Supervisor

## **COPYRIGHT STATEMENT**

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of NUST College of E&ME. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST College of E&ME, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the College of E&ME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST College of E&ME, Rawalpindi.

## ACKNOWLEDGEMENTS

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed, I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to express my gratitude to my supervisor **Dr. Farooque Azam** and my co-supervisor **Dr. Wasi Haider Butt** for their constant motivation and help throughout this thesis. Also, for Software Development and Architecture (SDA) and Model-driven Software Engineering (MDSE) courses which they have taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would like to pay special thanks to **Muhammad Waseem Anwar** for his incredible cooperation and providing help at every phase of this thesis. He has guided me and encouraged me to carry on and has contributed to this thesis with a major impact. Thank you for guiding me, often with big doses of patience.

I would also like to thank my Guidance Committee Members **Dr. Usman Akram** and **Dr. Rashid Ahmed** for being on my thesis guidance and evaluation committee. Some special words of gratitude go to my friend **M. Nouman Zafer** who has always been a major support and cooperation when things would get a bit discouraging.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

*Dedicated to my exceptional parents whose tremendous support and cooperation led me to this wonderful accomplishment*

## ABSTRACT

Database management Systems (DBMS) are one the most critical component of a software application. Searching data from DBMS is an enormous part in software performance. Text search engines are also used for searching, but these engines lack sophisticated DBMS features. Relational database management systems (RDBMS) are not quite compatible with modern object-oriented languages. To overcome the complexity of data and object-oriented programming, modern development practices adopted Object Relation Mapping frameworks (ORM). ORM bears a layer of abstraction between object models and database. This layer automatically bridges objects in OOP languages to database records, which results in significantly reducing custom mapping code complexity. ORM has its advantages but on the other side it comes with be some challenges too. In process of mapping objects and data, ORM keeps the relations between objects intact and that results in retrieval of multiple objects from multiple tables. When the data is big and have a hieratical structure, data retrieval or search becomes more complex. Database performance for the retrieval of data are optimized by adding indexing to each table. Indexing makes search significantly fast but also makes other processes slow because tables are required to be re-index every time a record is changed. Hence an optimized solution is required to resolve this problem in ORM search process. To overcome this problem, this research proposes a java-based framework that can interact between ORM and search engine. It consumes search engine web APIs to provide a layer that can convert and search objects to/from XML. It makes search process faster and support ORM with its object-oriented methodology. Moreover, this framework not only reduces performance load on databases but also makes search queries simpler when implemented in development process. The results have been validated by two case studies, which were carried out by implementing each approach. 1000 similar search queries were processed on each framework and results shows 30 to 40 % improvement in query time.

**Keywords:** DBMS Search, Indexing, Text Search Engines, Solr indexes, Object oriented programming (OOP), Object Relation Mapping (ORM), Search optimization, Information Retrieval, database indexing.



## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>i</b>
<b>LANGUAGE CORRECTNESS CERTIFICATE</b> .....	<b>ii</b>
<b>COPYRIGHT STATEMENT</b> .....	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>iv</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b> .....	<b>vii</b>
<b>LIST OF FIGURES</b> .....	<b>ix</b>
<b>CHAPTER 1: INTRODUCTION</b> .....	<b>11</b>
<b>1.1. Background Study</b> .....	<b>11</b>
<b>1.1.1. Relational Databases</b> .....	<b>11</b>
<b>1.1.2. Indexing</b> .....	<b>12</b>
<b>1.1.3. Text Search Engine</b> .....	<b>13</b>
<b>1.1.1. Object Relational Mapping</b> .....	<b>13</b>
<b>1.2. Problem Statement</b> .....	<b>14</b>
<b>1.3. Proposed Methodology</b> .....	<b>15</b>
<b>1.4. Research Contribution</b> .....	<b>16</b>
<b>1.5. Thesis Organization</b> .....	<b>16</b>
<b>CHAPTER 2: LITERATURE REVIEW</b> .....	<b>19</b>
<b>2.1. Literature Review</b> .....	<b>19</b>
<b>2.2. Research Gap</b> .....	<b>22</b>
<b>CHAPTER 3: PROPOSED FRAMEWORK</b> .....	<b>24</b>
<b>3.1. Architecture for Designing a Search System</b> .....	<b>25</b>
<b>3.2. Architecture for Tools and Techniques</b> .....	<b>26</b>
<b>3.3. Data and Search</b> .....	<b>26</b>
<b>3.3.1. Hibernate Related Concepts</b> .....	<b>27</b>
<b>3.3.2. Search Indexer Related Concepts</b> .....	<b>28</b>
<b>3.3.3. Errors/Logging Related Concepts</b> .....	<b>29</b>
<b>3.3.4. Data Types and Enumerations</b> .....	<b>30</b>
<b>CHAPTER 4: IMPLEMENTATION</b> .....	<b>33</b>

4.1. Optimization Architecture .....	34
4.1.1. Control Flow .....	34
4.1.2. Optimized Control Flow .....	35
4.2. Data Conversion Architecture .....	36
4.3. Re-indexing Service .....	38
<b>CHAPTER 5: VALIDATION.....</b>	<b>41</b>
5.1. Case Study .....	41
5.1.1. Requirement Specification .....	41
5.1.2. Integration of ORMSEM Framework with Software System .....	42
5.1.3. Search and data retrieval .....	44
5.1.4. Verification .....	45
<b>CHAPTER 6: DISCUSSION AND LIMITATION .....</b>	<b>51</b>
6.1. Discussion.....	51
6.2. Limitations.....	52
<b>CHAPTER 7: CONCLUSION AND FUTURE WORK.....</b>	<b>54</b>
<b>APPENDIX A</b>	<b>55</b>
<b>REFERENCES</b>	<b>63</b>

## LIST OF FIGURES

<b>Figure 1: SOLR Architecture</b> .....	13
<b>Figure 2: Research Flow</b> .....	16
<b>Figure 3: Thesis Outline</b> .....	17
<b>Figure 4: Search Engine Working</b> .....	24
<b>Figure 5: Architecture for Web base Search System</b> .....	25
<b>Figure 6: Architecture for Tool and Techniques</b> .....	26
<b>Figure 8: Hibernate</b> .....	28
<b>Figure 9: Client Related Concepts</b> .....	29
<b>Figure 10: Errors/Logging Related Concepts</b> .....	30
<b>Figure 11: Data Types and Enumerations</b> .....	31
<b>Figure 12: Class Diagram of Our proposed Framework</b> .....	33
<b>Figure 13: Flow of data retrieval query using Database</b> .....	34
<b>Figure 14: Flow of data retrieval query without Database</b> .....	35
<b>Figure 15: Conversion Engine</b> .....	36
<b>Figure 16: Object components</b> .....	37
<b>Figure 17: Synchronizing Database with Search Engine</b> .....	39
<b>Figure 18: Configure enumerations</b> .....	42
<b>Figure 19: Running SOLR</b> .....	43

# Chapter 1

---

## Introduction

## CHAPTER 1: INTRODUCTION

This section provides a detailed introduction about the research and research concepts. This section is organized in multiple sub sections. **Section 1.1** provides the background study, **Section 1.2** presents the problem statement of research, **Section 1.3** discusses the proposed methodology, **Section 1.4** gives the detail about research contribution, and thesis organization is presented in **Section 1.5**.

### 1.1. Background Study

The purpose of this section is to introduce the background study of multiple concepts which have been used in this research. These concepts include;

- Relational Databases
- Indexing
  - Cluster Index
  - Non-Cluster Index
- Text Search Engine
- Object Relational Mapping (ORM)
  - Data Access Object (DAO)

#### 1.1.1. Relational Databases

Now a day's software is playing an essential part in automating the business industry. Almost all businesses and areas where software is the primary element, need to store and retrieve their personal and confidential data. In software, databases are the elementary source for storing and retrieval of data as well as providing security to the data. The stored data is controlled and organized by Database Management Systems (DBMS). Database stores the data logically in form of tables which consist of rows known as record and column and physically on disk space in units of pages. The collection of pages creates heap data structure to store the data. To search a data from the heap a row locator identity is needed. Row locator is a pointer contains the identity number of documents, page number and page slot present on heap and it is enough of information which is required to obtain any record from the database.

Database follows some restrict rules to organize data. Storage and retrieval of data is carried out by mean of formal query languages such as SQL. The efficiency of queries has been paid more attention because of rapidly growing data stored in databases. Moreover, the user satisfaction and experience is mostly affected by the efficiency of queries in system where large scale of data is used. Query engine loads data into memory in form of units of pages. While dealing with large amount of data, the query engine distributes the data among multi-pages which can cause low query efficiency. To search data from a database using queries, indexing is used to optimize the search process [1]. Search engine indexing collects, parses and save data to improve the information retrieval process. In indexing, an index contains the selected columns of relevant data from a table. Purpose of storing an index is to optimize the searching speed and performance of a query as well as it improves the computing power and time required by the search engine.

### **1.1.2. Indexing**

In database such as SQL server,  $B^+$  tree is used for indexing purpose. It is a balanced multi branch tree which optimize the I/O operations of the database. It provides the  $O(\log N)$  time to the query to perform its insert, update, delete and other dynamic operations. The working process of index is to perform search on data present in form of tree starting from the root node. Index consist of index key range and a pointer to next node of the branch. The bottom node of index is leaf node that contains information about location or data itself. Searching required number of rows of data using I/O operation in database is determined by index layer throughout the index's root node to leaf node. Hence, indexing provides same traversing lengths and efficiency for all queries to search data from database.

Depending upon the data indexes are divided into two type; cluster Index and non-cluster index. Every table contain 249 non-cluster index and one cluster index. Physical index is categorized as cluster index while logical index is non-cluster index. Cluster index requires around 1.2 times of table's disk space which is the average size of about 5% of the table [2]. The leaf node of  $B^+$  tree in cluster index is data page and it may require splitting of data page and reorder in index page while performing insert, update, delete and other dynamic operations. But physical position of data does not change while dealing with non-cluster index. Non-cluster index is like vocabulary page in a book and leaf node use the pointer to point the position of record. Hence in non-cluster index tree do not contain data pages and is composed of index key entry.

### 1.1.3. Text Search Engine

A full text search engine provides services of indexing large data exposed through APIs to be consumed by other software. As these engines are designed dedicatedly for search purposes, they provide much superior index management and information retrieval. Fig 1 illustrate the architecture of a sample search engine that communicates through web services with other application systems. [4]

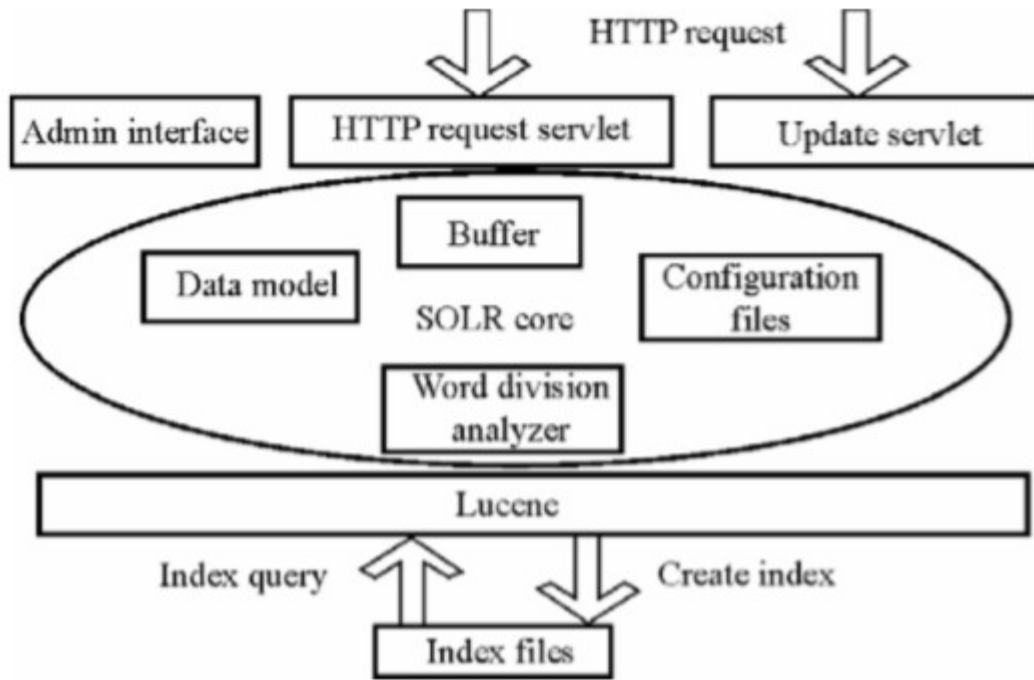


Figure 1: SOLR Architecture

### 1.1.1. Object Relational Mapping

Managing data consistency between source code and database is a difficult task, especially for complex large-scale systems. As more systems become heavily dependent on databases, it is important to abstract the database accesses from developers. Hence, developers nowadays commonly make use of Object-Relation Mapping (ORM) frameworks to provide a conceptual abstraction between objects in Object-Oriented Languages and data records in the underlying database. Using ORM frameworks, changes to object states are automatically propagated to the corresponding database records. A recent survey [9] shows that 67.5% of Java developers use ORM frameworks (i.e., Hibernate [10]) to access the database, instead of using

JDBC or other frameworks. However, despite ORM's popularity and simplicity, maintaining ORM code (i.e., code that makes use of ORM frameworks) may be very different from maintaining regular code due to the nature of ORM code. [6]

ORMs are tightly linked with object models that makes data retrieval specific to models. That means whenever we are querying database for data retrieval, we will always get whole object, including child objects or linked objects which is either required or not. This behavior makes our application a little heavy. Data access models (DAO) are used in ORM models specifically to make data retrieval simpler. DAO is a simplified version of big object to retrieve only those entities or attributes from DB only which are needed.

## **1.2.Problem Statement**

With evolution of software industry and modern paradigm of businesses. Data is becoming the biggest asset of a software. More data makes a software more reliable. With this expansion in data, data management is becoming more complicated especially for daily use applications. More and more data make it hard for DBMS to perform CRUD operations on it. But the most affected operation is search, especially in text.

Most significant factor of a software performance is its search capabilities. Keeping data adding up every day, application's search procedures need to be change. Indexers are often used to overcome search problems. But we cannot index every column in DBMS or in OODB. After every change in any record whole dataset needs to be re-indexed. This may significantly reduce I/O accessing cost in query processing.

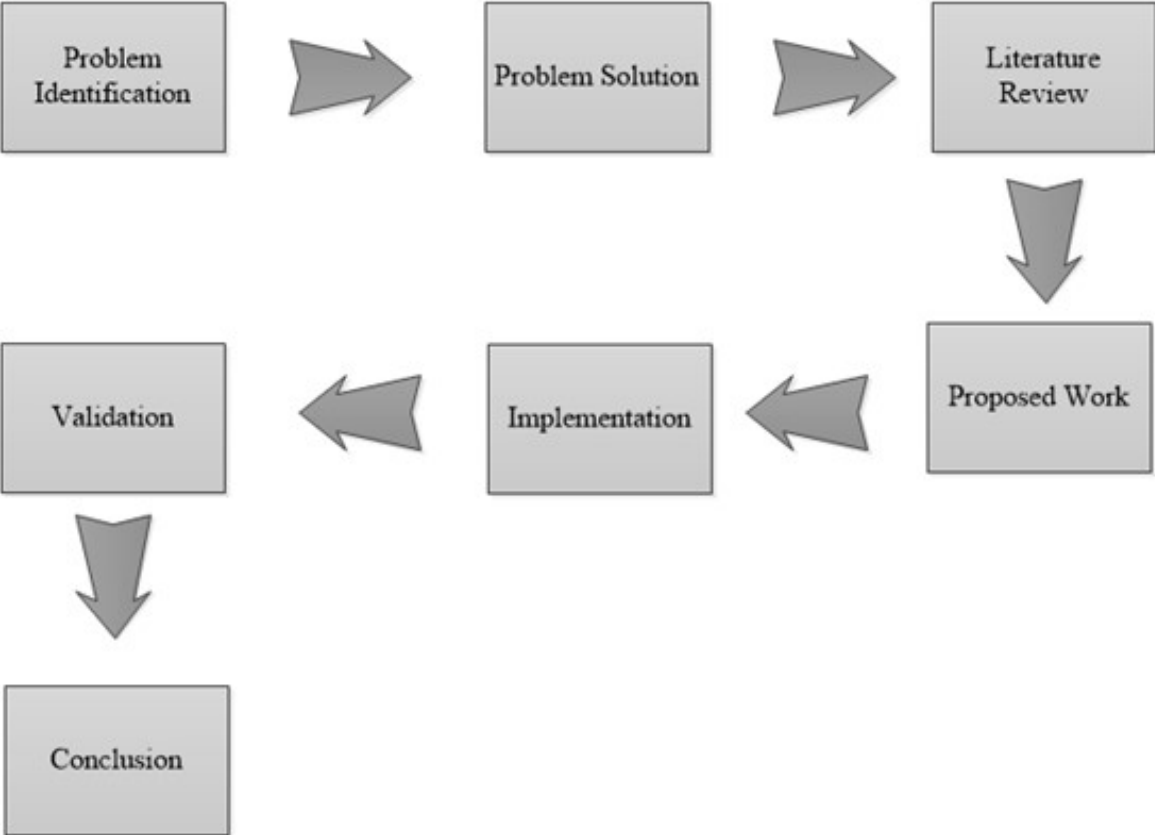
Now a day when technology is so advance, and AI is implemented in major processes. Majority of DBMS are taking care of indexing process by themselves. But DBMS does not have a concept of object-oriented structure. So, whenever we try to retrieve data, we have to query every linked table. ORMs came across a long way to deduce this problem and established a link between OOP and relational databases. Querying through ORMs helps one way or another in going to use DBMS and same is the cost data retrieval cost. This cost becomes a problem when there is a huge data and it has object-oriented structure implemented by ORMs.



### 1.3. Proposed Methodology

Entire research is done in a very systematic way. **Figure 2** represent the flow of research step by step. In first step we identified the problem. Then proposed the ideal solution for the problem identified in first step. We carried out a detailed and comprehensive literature review which helped us to identify the optimal solution for the problem. We reviewed the researches carried out related to our proposed solution, analyzed and compared them.

The proposed solution implements an approach that is going to make our data retrieval process significantly faster than traditional methods. The propose framework will provide an interface between our ORM and the text search engine. Our proposed framework is also going to take care of boxing and unboxing of objects into readable formats for ORM and search engine as well. Interaction of ORM and DBMS is reduced and use a separate search engine which also have self-indexer. ORMs interacts with the search engine through our designed framework and it has also capability to keep search index updated via a service that keeps DBMS blend with search engine. The proposed methodology has been validated for two case studies of different sizes.



**Figure 2: Research Flow**

## **1.4. Research Contribution**

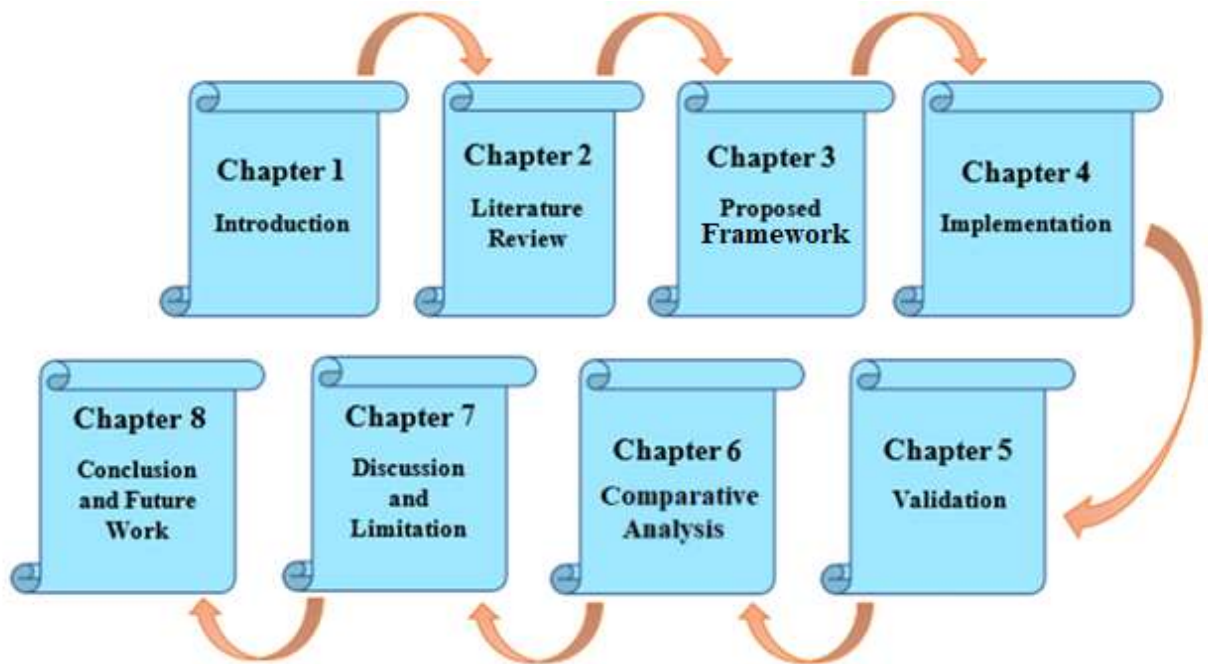
Contribution of current research work are; changing and optimizing conventional methodology of data retrieval and providing a new platform for ORM to interact with text indexers. Detailed set of contributions of the proposed approach are as follows:

- We have developed a framework that provides an interface between JAVA and search engine persisting object-oriented programming structures.
- We have made search engine capable of storing more than just text. Generation of XML, boxing and unboxing of objects is a big part of it.
- Providing a time triggered service that can keep data updated on both data storages i.e. DBMS and search engine.
- We have provided a framework that helps ORMs like hibernate to consume APIs of search engine keeping its structure intact.
- We have provided a generic configurable a framework uses connections and configurable variable to embed with any java project.
- We have provided validation of our proposed work using two benchmark case studies.

## **1.5. Thesis Organization**

**Figure 3** represent the organization of thesis. **CHAPTER 1:** deals with introduction having detailed background study about the concepts used in the research, problem statement, research contribution and thesis organization. **CHAPTER 2:** contains the literature review which provide a description of work done in the field of Distributed Control System. In Literature review we also highlights the research gaps that we encountered. **CHAPTER 3:** covers the details of proposed framework used as a solution for identified problem. **CHAPTER 4:** presents the detailed implementation regarding the proposed model, profile, tool and transformation engine along-with its architecture. **CHAPTER 5:** provides the validation performed for our proposed methodology using two important case studies. The two case studies selected for validation purposes are of different domains and different sizes to make sure that our proposed approach works on every case. Error! Reference source not found. contains a brief analysis of our proposed work with previous researches. **CHAPTER 6:** contains a brief discussion on the work done and also contains

the limitations to our research. **CHAPTER 7:** concludes the research and recommends a future work for the research.



**Figure 3:** Thesis Outline

# Chapter 2

---

## Literature Review

## CHAPTER 2: LITERATURE REVIEW

This chapter presents research work that we carried out focusing on object relational mapping interaction with databases and involvement of indexers. After a deep literature review of scrutiny of our efforts and research we were able to find some research gaps which support our research significantly.

### 2.1.Literature Review

Lujia et al., implemented an information retrieval system or corpus based on solr (a search engine). The from mangolian websites stored in MySQL database. The problem of words with same pronunciation but different shape was resolved. Using the approach proposed in [11], the mangolian words are represented as latin characters and specific meaning is associated with ASCII characters. The data crawled from mangolian websites and textbooks was transcoded which involves transformation of original data to Unicode. The proposed retrieval approach involved transformation of code for mangolian, building indexing document with solr, text clustering and finally the retrieval.

Nemanja et al., proposed an ORM approach in order to exploit the optimization techniques that minimize or abandon the positive effects of normalization by reducing the database performance. This ORM approach is a hybrid approach which combines the static and dynamic characteristics of model. This ORM transformation pattern involves the generation of object identifier, incorporating mapping of single class inheritance, mapping associations, optimizing transitive associations and storing the derived values. Authors also presented an initial framework for detailed analysis and comparison of ORM approaches which use the de-normalization techniques [12]. Another research provides detailed study on approaches that can be used for maintaining the ORM code in Java [14]. ORM provides an abstraction between source code and database. Authors highlighted that more detailed research is required for the use of ORM framework in software maintenance community.

Yang et al., proposed an algorithm for full text retrieval using block linked list index structure [13]. Experiment shows that the proposed approach increases the index initialization time

an also improves the efficiency of data retrieval. The approach has been implemented for large data of Chinese full text retrieval system.

C. Giordano et al., presented the implementation of business application based on Hibernate ORM framework which can communicate with Relational Database Management System as well as noSQL database. The developed application provides performance evaluation of selected two databases including MySQL and MangoDB. The system was later evaluated on parameters like throughput, scalability and response time. Hibernate ORM facilitates in development of DB independent applications which are portable or easily migrated across various RDBMS [15].

Junwen Yang et al., provides a detailed survey of 12 real world ORM applications on the basis of 9 generalized ORM performance anti-patterns. This research identifies fixated performance issues for each application. Furthermore, causes of inefficiencies are discussed in detail [16]. In another research, indexing has been focused for Information Retrieval systems [17]. Indexing process involves four stages: content specification, document tokenization, processing of terms and building index. Multiple indexing algorithms like two-pass indexing, single-pass indexing, and block indexing have been discussed by the authors.

Platonov et al., implemented an ORM Polar system and suggested a disk space optimization method. Results obtained from the suggested methodology were compared with that of RDBMS. The proposed method increases the possibilities of use of ORM for non-relational Nosql databases (Polar). The results demonstrated that ORM can be used for specialized databases and it successfully competes with same constructions for universal databases [18]. Another research [19] discusses the use of ORM for modelling the domain ontologies for semantic web in OWL or SWRL. The research proposed some rules for mapping structure and constraints of ORM with SWRL.

Shailender et al., implemented Temporal Functionality in Objects with Role Models using PostgreSQL, an open source ORM. The authors proposed a methodology where initially schema of database is designed. After designing the database, mapping process is performed which maps the TF-ORM with underlying relational database. Two types of rules are defined by the authors. First is state transition rule and the other is binding rule for roles and classes with temporal information. The proposed model has been validated for a hospital scenario [20].

In [21], Object Relational Mapping has been utilized for automating the persistence layer of web-based systems. Authors have investigated that how ORM approach can be integrated to support the event driven development. The proposed approach has been validated with running example of RubiS web app benchmark example in order to evaluate the performance of approach.

QI Chunxia et al., worked on index-based query data in order to improve SQL Server database query performance. The authors have explained the principles of properly using the index-based approach and also highlighted the factors that can reduce the query efficiency. Some query conditions, calculations and functions to be avoided in order to get effective indexing have been discussed in detail [22]. Similarly, another research paper gives a demo of a tool named ‘TagTick’ which offers a fully functional annotation tagging environment for Apache Solr to data curators [23]. The presented tool allows tagging and un-tagging actions over the index in work sessions without compromising end users’ index searching activities.

Joseph Armas et al. worked with structured language query and ORMs simultaneously. Using sql server on backend as database and Entity framework with C# as an ORM. The authors focused on comparison of query performance between both data retrieval strategies [24]. Querying and calculating time differences significantly shows that ORM performs well but this only problem was that author compare single level of hierarchy. Object with no child objects or table with no sub tables in relation. If we try multiple level of hierarchy, but system performance drops significantly.

Jiawei Han et al worked on structural difference indexes between object-oriented data (hierarchal) and single level data. Due to complex structure of objects via class/subclass, attributes, associations, navigation between objects by composition hierarchy it is highly recommended that data indexed should have the same capabilities [25]. Another research by Yang Lai et al, also support the same concept [26]. Using Hadoop, HDFS and MapReduce with Java persistence API (ORM) and comparing MySQL Clustering shows a significant perform improvement. And analyzing that big data failure with MySQL Clustering.

In [27], a brief study is carried out ORM performance query and its flaws author also recommend some anti patterns that are necessary for ORM performance. Problems like unwanted column data retrieval, aggressive computation of DBMS queries, query batching etc. were also analyzed.

## 2.2. Research Gap

This section deals with the research gap and proposed solution in industrial scale usage of data repositories. With automation of every industry now we are more depending on software than ever before. Due to expansion of this software industry we are generating data exponentially. Data generated in last 5 years is approx. 97 % of whole data ever created. Data storage in merely in DBMS is not working anymore. Database systems are developed with additional features, but each comes with their own cons. Database management systems are responsible to enable user to perform CRUD operations, but data retrieval is the most curtail one. All operation except create operation must performs retrieval or search operation to complete itself. So, all other operations have great dependency on data retrieval. Considering the size of data that we are working with, it also is the costliest one.

Previous carried out research have contributed a lot in this domain, and we have significantly improved and different variety of database systems. Continuing best programming practices researchers also implemented middle ware like ORMs to distribute the load on DBMSs. ORMs have their draw back, as they are keeping out data in a OOP structured form but makes data retrieval more complicated as it also retrieves all the assisted objects too. Multiple researches presented noble approaches to improve the architecture of system and made the design and implementation of the system, a complex and sophisticated process. Adding indexers to DBMS improved data retrieval significantly. But DBMS indexer can't handle huge data especially when we are talking about relational and hierarchal data. Indexers need to be updated every time there is create, update or delete operation performed. As they are built into DBMS it cost them a lot to re-index huge data every time there is an operation performed. Majority of the work carried out is on tightly coupled ORM, indexer and database. Making in efficient query systems. [27]

The solution that we proposed is significantly helpful in separating these frameworks and systems to distribute load on separate resources. Creating a middle ware between them not only improve data retrieval efficient also open a new variety of customizations and object compatibility support. Furthermore, we have provided an open source and generic solution for the data retrieval which can keep data synced with database having lesser complexity.



# Chapter 3

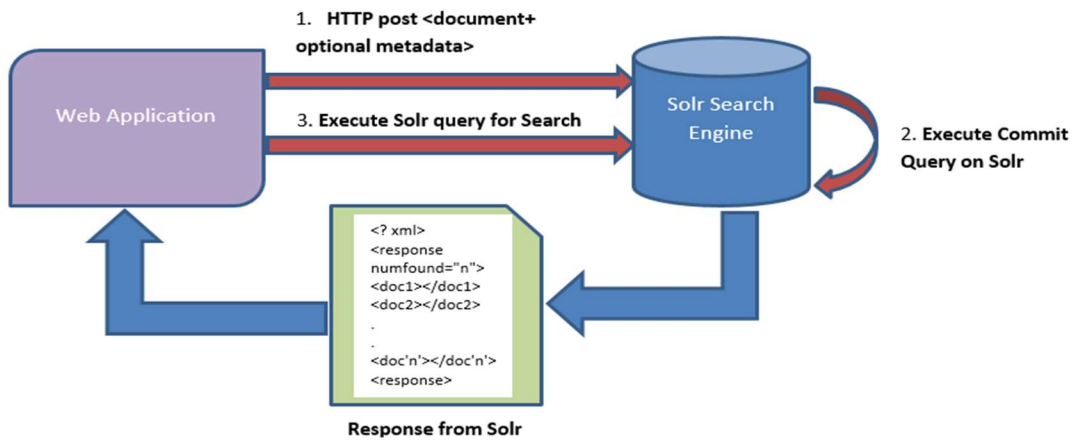
---

## Proposed Framework

### CHAPTER 3: PROPOSED FRAMEWORK

As discussed, earlier Data management and retrieval system is a widely used as software systems which control, monitors and execute the critical processes of the industry and execution of these critical processes depend on size of data. Developing a system that can save time and performance makes either of complexity cost unbalanced. To provide a generic, well-coordinated and interoperable platform ORM and search engine mapper (ORMSEM) is proposed. **Figure 4** shows the overview of framework. The data transactions between a search engine and a web application that supports ORM and it can communicate to search engine by web APIs. The data retrieval system is the integral part of the Software Systems where performance depends on data transactions. To maintain and synchronize data between database and search engine, a data search-based retrieval system needs to connect with it and other parameters such ORM to process the data etc. Thus, multiple concepts need to be integrated to provide a generic data retrieval system for software systems which needs search in big data.

The purpose of this chapter is to give detail of concepts used in the proposed solution. The recommended solution is based on Object oriented architecture and object relational mapping

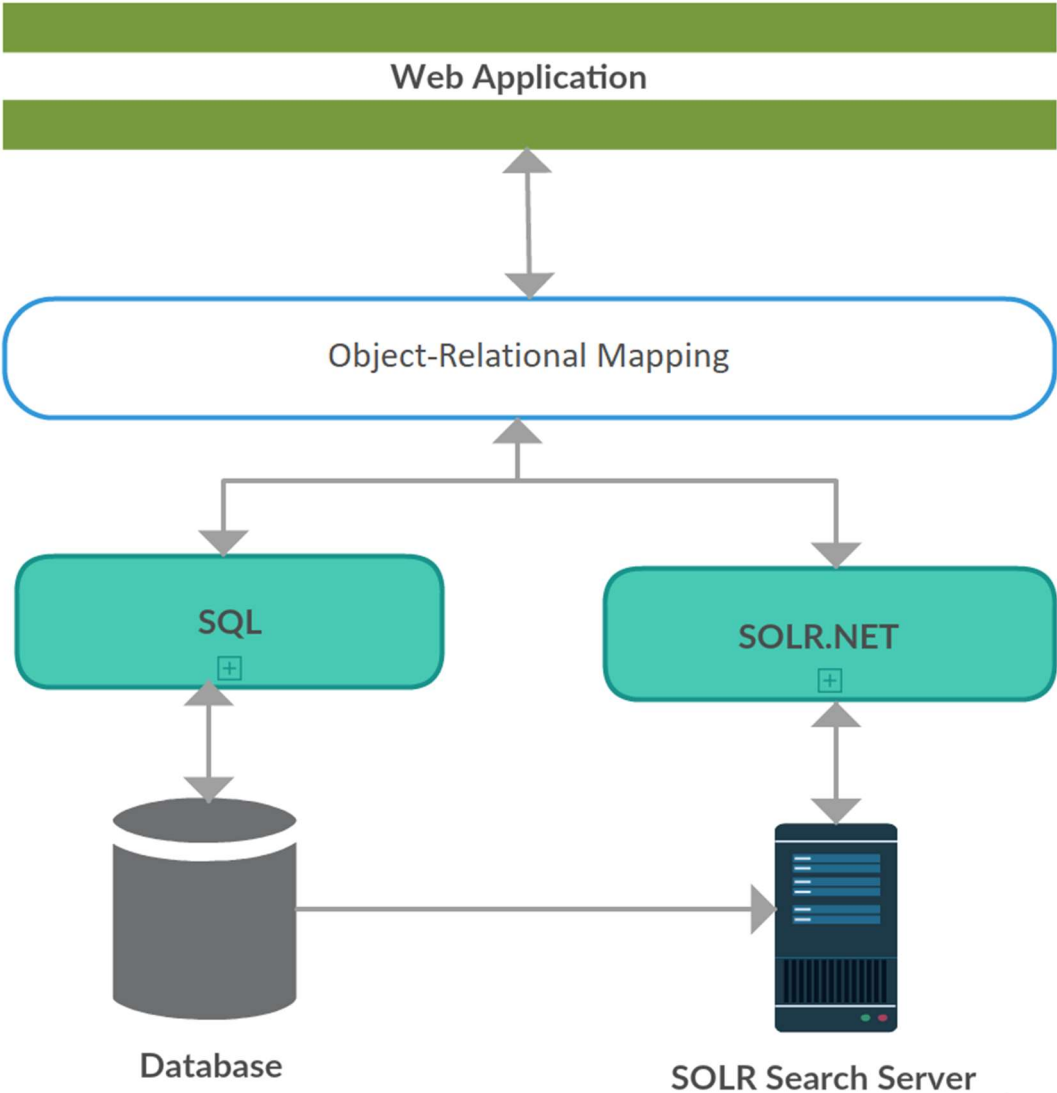


**Figure 4:** Search Engine Working

### 3.1. Architecture for Designing a Search System

A search system is usually designed along with other data bases like sql to enhance the performance of system but with the ORM is in the middle it is still not very effective because of extra data retrieval calls to database.

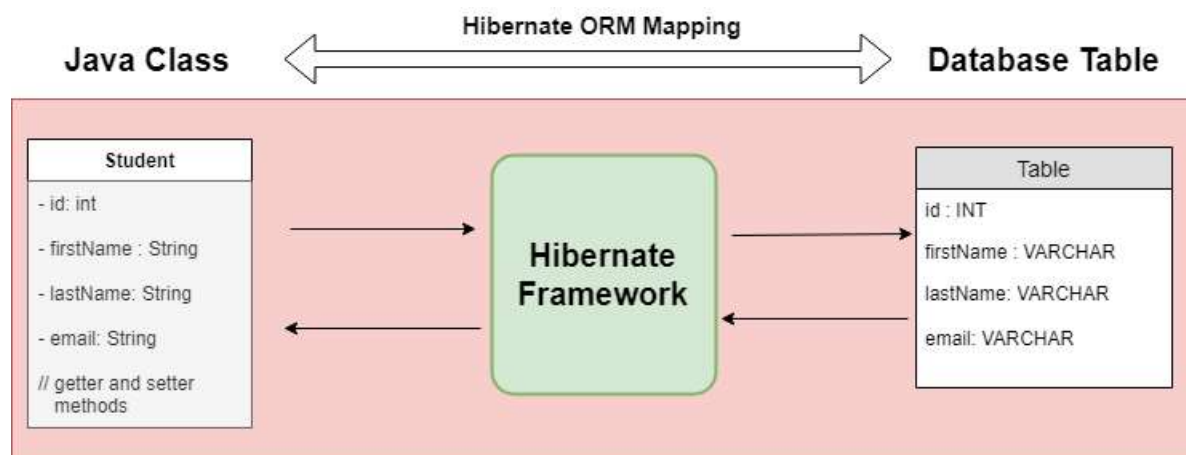
**Figure 5** The data is still being retrieved from database because ORMs like hibernate or entity can only support conversion of objects to data and vice versa with only DBMS. To take it further introducing a new framework between ORM and search engines ORM can be supported.



**Figure 5:** Architecture for Web base Search System

### 3.2. Architecture for Tools and Techniques

Tool support is an important factor to increase the productivity of software development. A tool support architecture to support the framework is shown in **Figure 6**. It consists of four layers development environment, ORM, database and Simulation Tools. Development environment contains the Eclipse Neon which is an open source environment having variety of plugin available in market for software development and modeling purpose. ORM layer in eclipse provides a communication between object-oriented programming and data tables. Eclipse support ORMs like hibernate and entity to help conversion of data from table rows to linked objects. The simulation tool layer contains Visual Studio, simulator, Output Analyzer, Address space updater and visual displayer. These tools are for testing the code, address space and data.



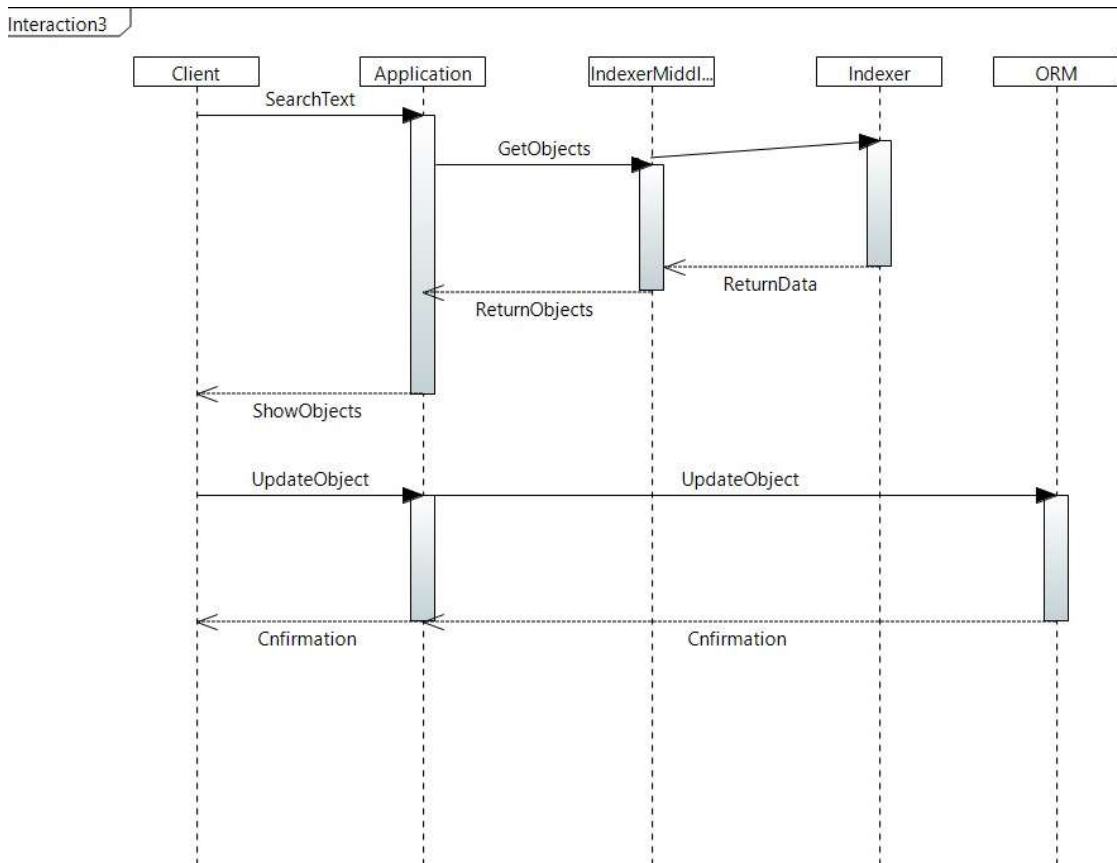
**Figure 6:** Architecture for Tool and Techniques

### 3.3. Data and Search

The proposed solution implements an approach that is going to make our data retrieval process significantly faster than traditional methods. The propose framework will provide an interface between our ORM and the text search engine. Our proposed framework is also going to take care of boxing and unboxing of objects into readable formats for ORM and search engine as well. Interaction of ORM and DBMS is reduced and use a separate search engine which also have

self-indexer. ORMs interacts with the search engine through our designed framework and it has also capability to keep search index updated via a service that keeps DBMS blend with search engine. **Figure 7** explains a brief flow.

We can completely skip a data retrieval call from database. Our framework will enable us to get objects from search engines.



**Figure 7.** Control flow of solution

### 3.3.1. Hibernate Related Concepts

**Figure 7** Hibernate is an open source Java persistence framework project. It performs powerful object-relational mapping and query databases using HQL and SQL. Hibernate is a great tool for ORM mappings in Java. It can cut down a lot of complexity and thus defects as well from your application, which may otherwise find a way to exist. This is especially boon for developers with limited knowledge of SQL.

Initially started as an ORM framework, hibernate has spun off into many projects, such as Hibernate Search, Hibernate Validator, Hibernate OGM (for NoSQL databases), and so on.

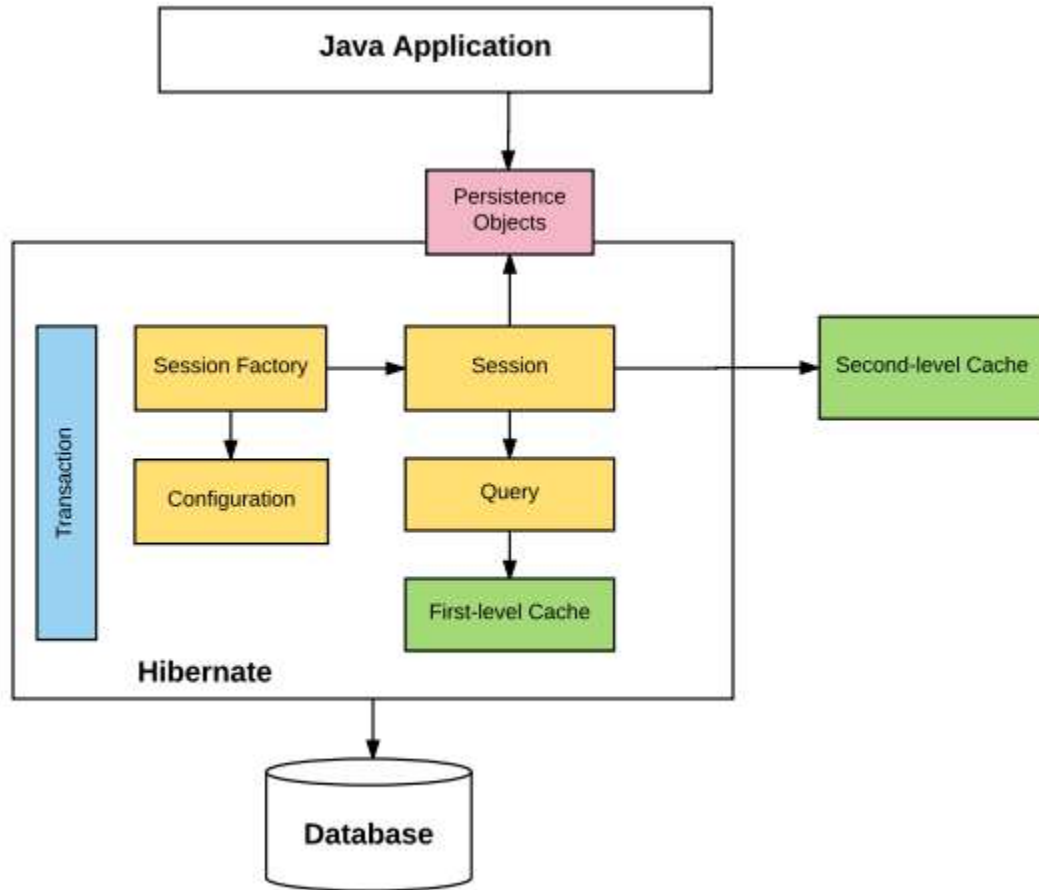
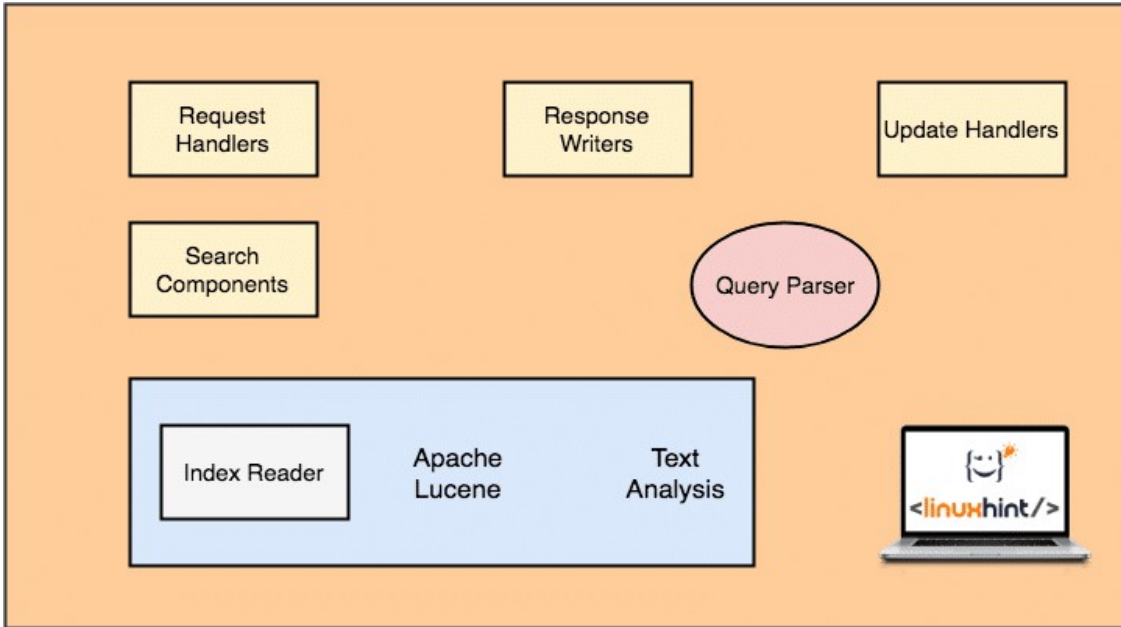


Figure 7: Hibernate

### 3.3.2. Search Indexer Related Concepts

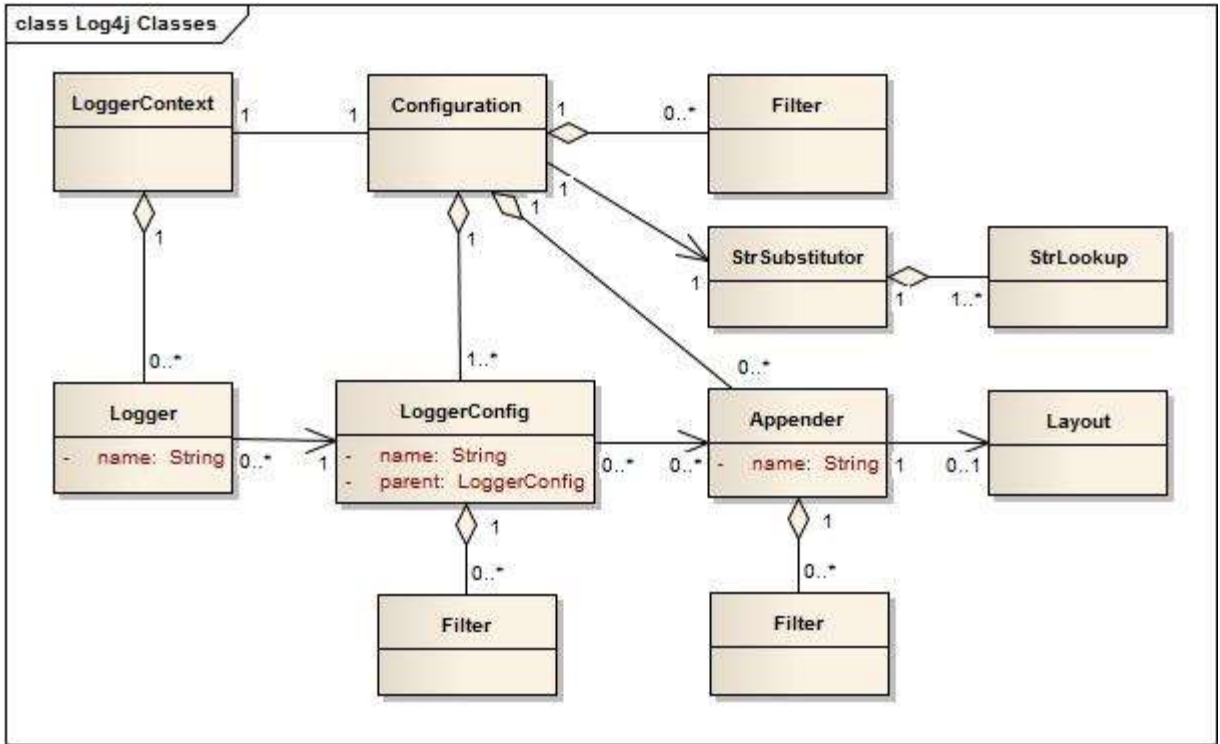
Concept related to indexer shown in **Figure 8**. A full text search engine provides services of indexing large data exposed through APIs to be consumed by other software. As these engines are designed dedicatedly for search purposes, they provide much superior index management and information retrieval. Fig 1 illustrate the architecture of a sample search engine that communicates through web services with other application systems. [4].



**Figure 8:** Client Related Concepts

### 3.3.3. Errors/Logging Related Concepts

**Figure 90** represents the concepts related to data conversion logs and errors. *Logging* stereotype is introduced for the modeling of conversion concept and contains the attributes; *Name* to specify its identity, *Appender* which states the conversion type, and *Parameters* which contains the necessary information about the conversion. *Loggercontext* is introduced to model the necessary parameters for conversion and it's high and low values in integer. *ErrorsAndException* stereotype represents the concepts of errors and exceptions that could possibly occur during connection or at any stage of data transmission. It contains the attributes; *Connection Error* which deals with error occur during connectivity, *Slrlookup* checks if the requested input crosses the limited range of memory, and *Logout* checks if connection could not be established with in specific time.

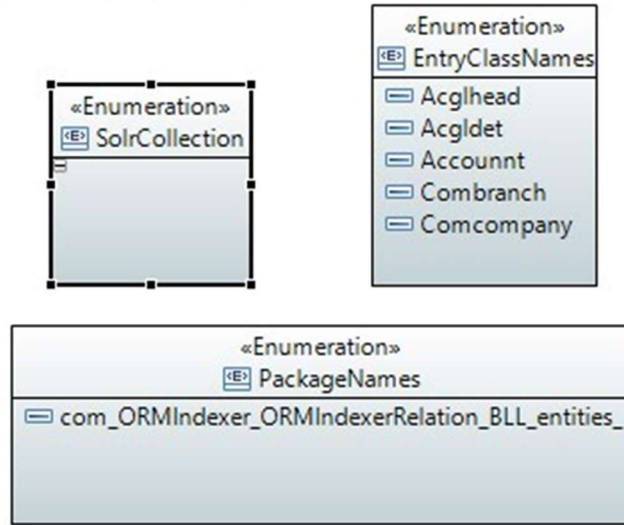


**Figure 9:** Errors/Logging Related Concepts

### 3.3.4. Data Types and Enumerations

**Figure 10** shows the all data types and enumerations used in modeling of above discussed concepts. Enumeration *EntryClassNames* is introduced to represent the classes that need to be indexed i.e. *Acglhead*, *Acglhead*, *ComBranch* and *ComCompany*. These classes are specified to indicate the level of object to be indexed and the specific classes that should be added to search engine. Function codes are also used to read and write data on these memory types. *SolrColelction* enumeration contains the set i.e. in which our data is going to be stored on search engine. *PackageNames* contains the package name in which our models initially lie.





**Figure 10:** Data Types and Enumerations

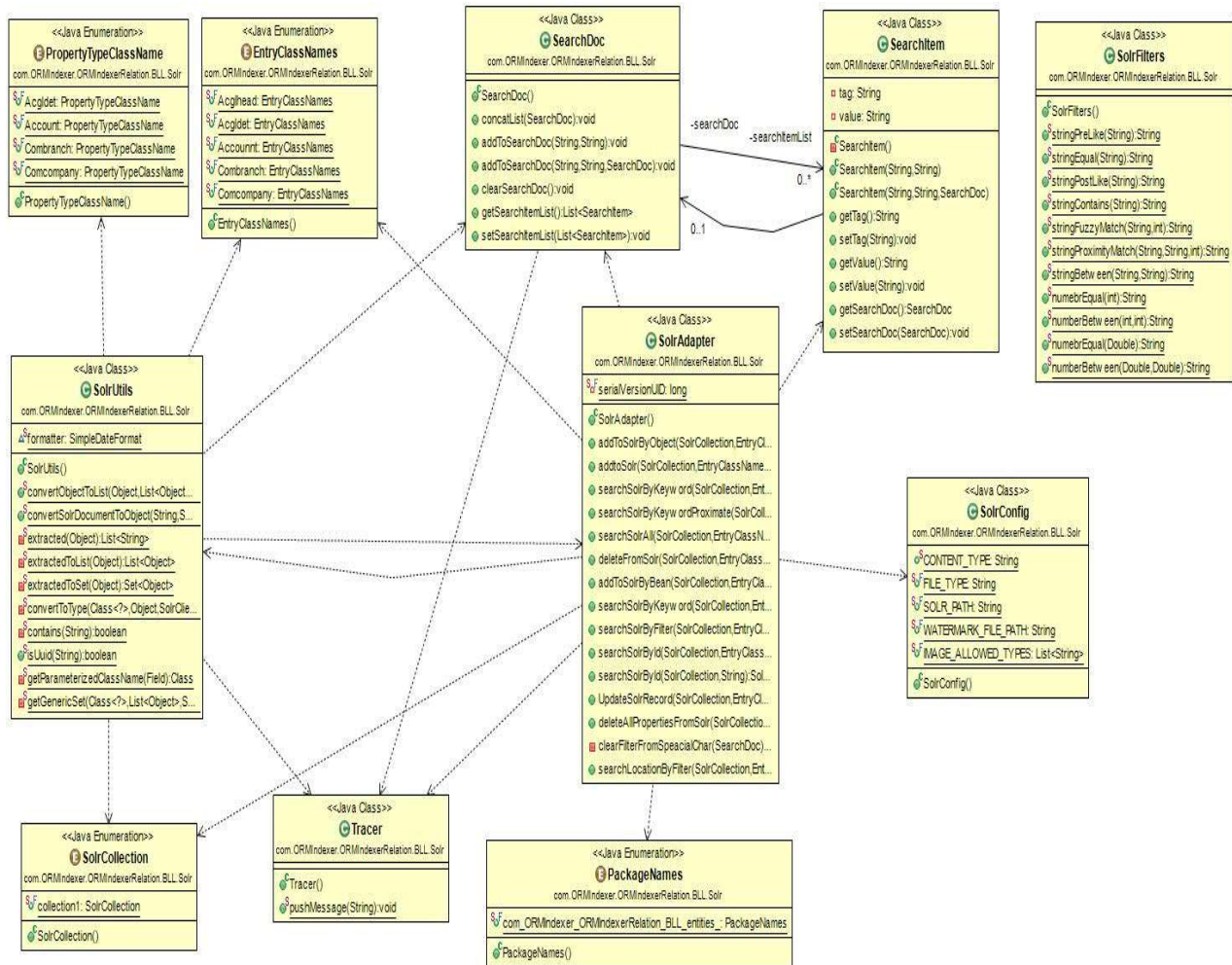
# Chapter 4

---

## Implementation

## CHAPTER 4: IMPLEMENTATION

This chapter deals with the implementation detail for ORM and Search Engine Mapper (ORMSEM) (a tool we have proposed to enable ORM to directly get data in object form from search engine). The tool we have implemented has three main features. Firstly, it provides the facility to synchronize database with search engine while keeping its object model. Secondly, it provides facility to convert xml documents to ORM objects. And finally, it provides a custom level object search (including Elastic search). **Figure 11** shows the main Class structure of our proposed framework. The base of our work strictly depends on Object Oriented Architecture. We have used Eclipse Database development and hibernate in our project. Hibernate is being used as an ORM to communicate between our objects and database.



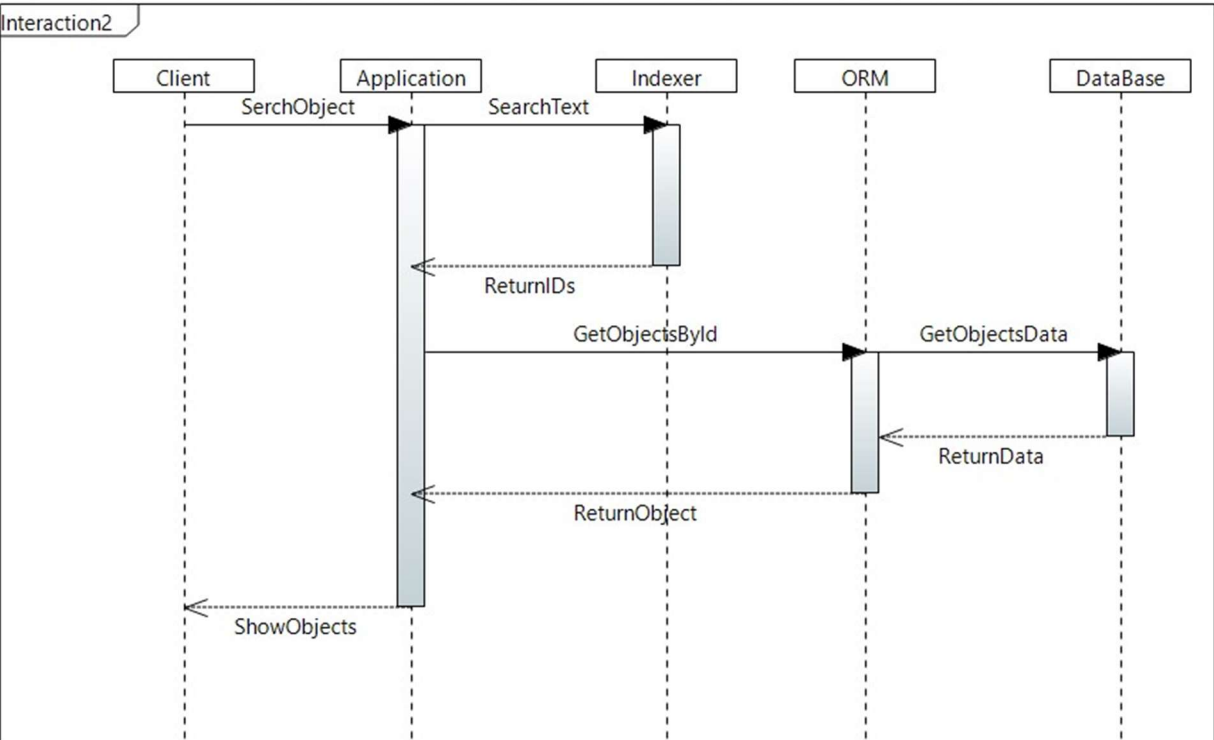
**Figure 11: Class Diagram of Our proposed Framework**

### 4.1. Optimization Architecture

In this section optimization and its conversion procedures are discussed which are used to map the model from ORM and transform it into indexers strings and address space. The process of modifying or converting an OOP model into text or another model is known as boxing of objects and the formal rules which are used to transform the concept of one model to another model or text are usually maintain. The main aim of transformation or conversion is focused to develop the set of formal rules to ensure model continuity and to reduction of information loss and effort as much as possible.

#### 4.1.1. Control Flow

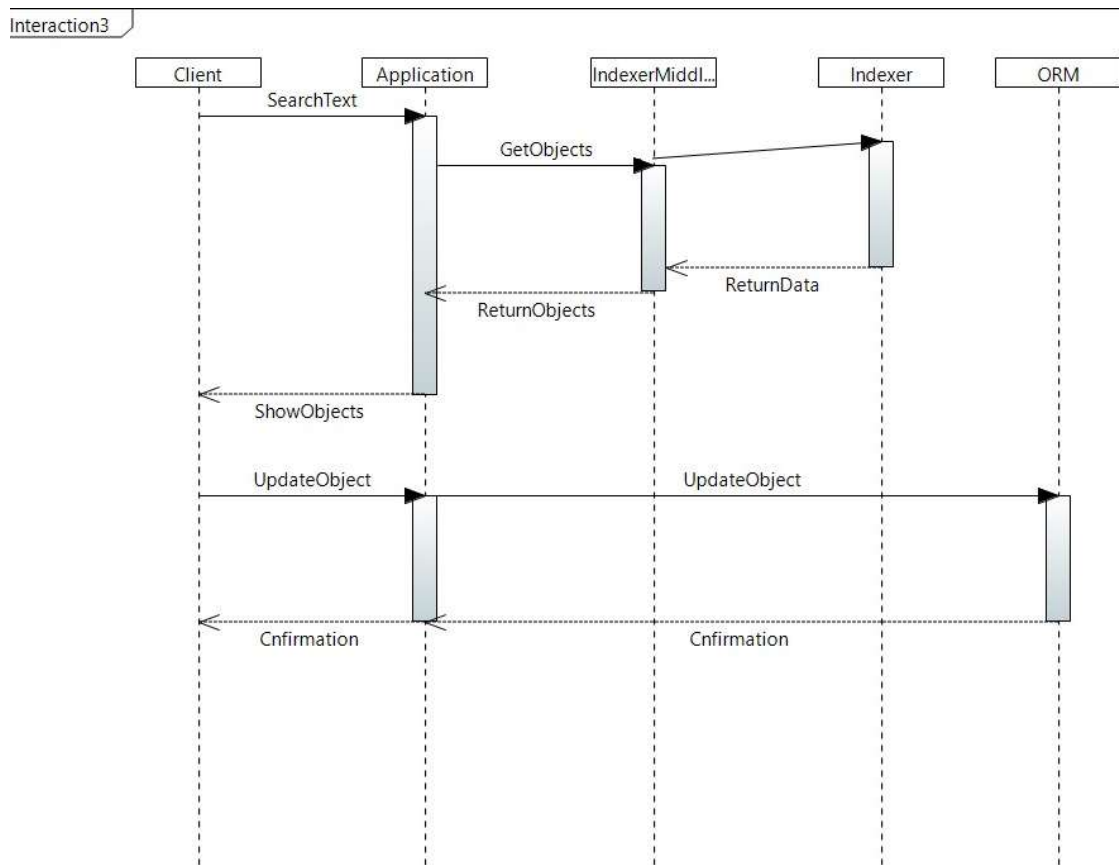
Conversion of objects to text or indexer documents is usually denoted as the focus is to develop the textual or code artifacts from the model. The engine which is used to carry out conversion process is being designed in our framework. Current section explains the different calls an application is to make and how data is being retrieved. In **Figure 13** it is explained that how an architecture of a well indexed system works. How multiple calls must be made to get complete objects from database.



**Figure 12:** Flow of data retrieval query using Database

### 4.1.2. Optimized Control Flow

After transformation engine being emplaced the complexity of architecture is optimized. As shown in **Figure 14**, Error! Reference source not found. call to database is completely omitted which is the most expensive in terms of complexity and performance. A simple search call to indexer or search engine via the transformation engine ORM can retrieve objects. This phase of process is known as unboxing of objects. It converts text back to objects to support OOP and ORMs.



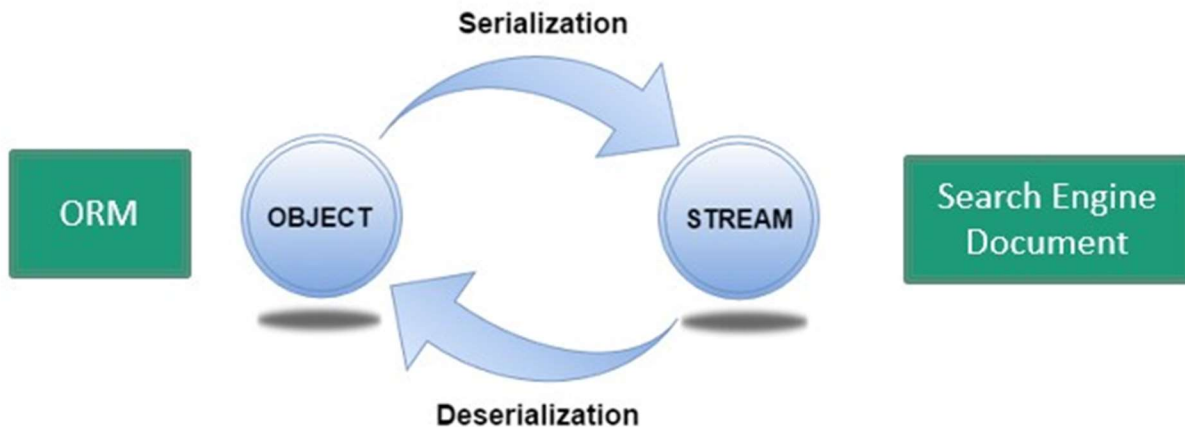
**Figure 13:** Flow of data retrieval query without Database

As data is being retrieved directly from text search engine it is connected to database anymore. Any change to this retrieved data will not be reflected to database. Any update this data will have to be manually save to database. As the second call showing in **Figure 14** any update in object will have to be updated to database through ORM. And later it will be automatically synced to search engine through our re-indexing service, as discussed in **section 4.3**. To retrieve data or objects search methods are being implemented. Search is carried out through *SearchDoc*, it takes

search filter as parameters. Filters can be applied on multiple object level in hierarchy to retrieve most suitable objects found.

## 4.2.Data Conversion Architecture

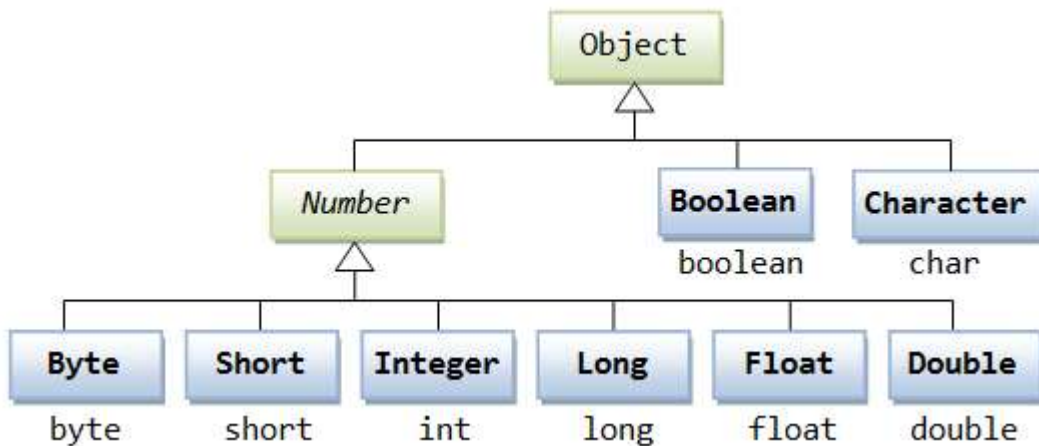
The data conversion is the core of our framework, it solely responsible for converting objects to *SolrDocuments* and from *SolrDocuments* to ORM objects. transformation engine is responsible for converting the model into string and then into solr documents and its architecture is presented in **Figure 14**. *SolrAdapter* contains methods that are responsible for conversions. A concept of serialization and deserialization is used to support *solrdocument* with ORM or java objects. Already built-in serialization in java supports only XML formats to in order to convert it into *solrdocument* we need to develop our own conversion engine, which is using same concept.



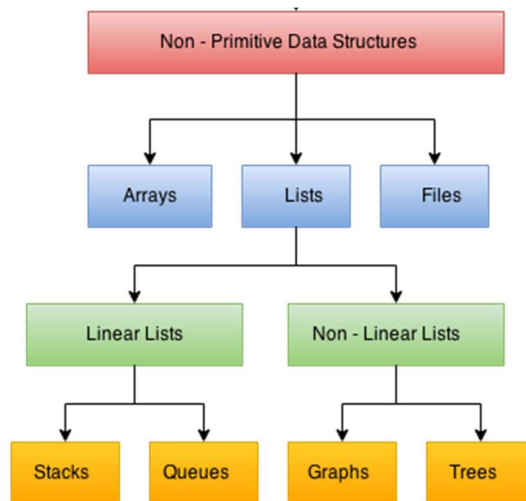
**Figure 14:** Conversion Engine.

Main interface of the transformation engine contains three major sub-components; Main Screen, Launcher, WinMain and Text Refiner. *Main Screen* is the main executor of the transformation engine which provides list of actions it can perform and provides graphical user interface with help of buttons and text field. *Launch* and *WinMain* are the java classes which implement these functionalities. *Text Refiner* is a java class which is used to convert the strings into proper format for further use. It provides access to three main functionalities of application

i.e. Papyrus, Address space updater and Transformation Engine. Papyrus button provide access to open Papyrus which is the source tool for modeling. Address space updater provide access to a tool which is specially created for editing address space file containing devices information. And finally, transformation engine can be accessed by button provided in Main Interface. Interface for transformation engine is provided in **Figure 15**. Transformation Engine takes the UML model and path of destination folder for generating the code from model using the browse button. Check boxes are provided for user to select only the type of output he needs to generate. Generate button is provided to generate the required outputs. *Status* bar show the progress of the transformation process. A *Reset* is provided to clear all fields i.e. input model path, destination folder path, status and all check boxes. *Close* button is provided to closes the interface from the screen.



**Figure 15:** Primitive Object components.



**Figure 16:** Non-Primitive Object components.

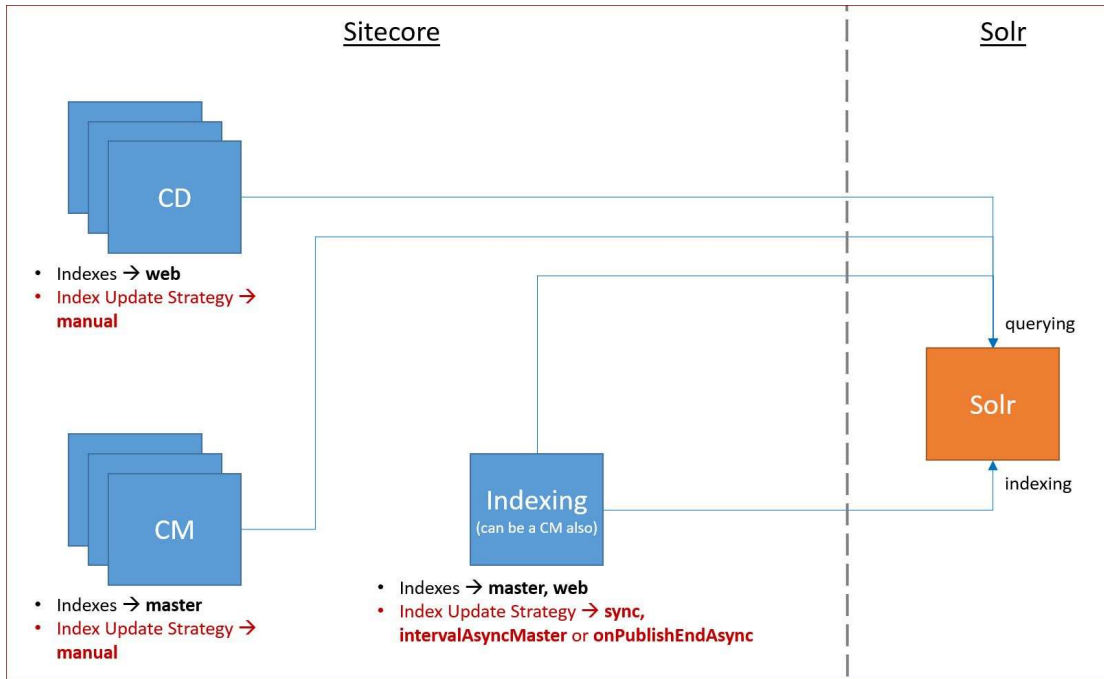
**Transformation:** POJOs are the Plain Old Java Objects, which were easy to handle because they contain only primitive datatypes as shown in **Figure 16**. Conversion of these objects are easy. For non-primitive objects have complex structures as shown in **Figure 16a**. These objects are hard to serialize as they have complex member types. *Java Reflection* is used in our framework to define rules and deal with the details and specifications of objects. Objects are being converted to text keeping object data and data member types with them. Links with other objects are also maintained by linking *solrDocuments* Ids with one another, a unique UUID is being used to maintain each object identification. an ain Interface accepts the UML model as input and passes it to the transformation.

### 4.3.Re-indexing Service

As our database and clients need to be synchronized. We created a service i.e. *ReIndexAll*. what it does is firstly it delete all the solr documents that are already indexed in engine and then ORM provide it the objects in bulk to re index. Bulk data transactions have less complexity and takes less time. A query is being made by ORM to database to return all the objects of some model already mentioned in enumeration *EntryClassName*. Now as ORM is already integrated with our framework It calls *ReIndexAll* and provide that list of objects to *solrAdapter* which is then beign converted to solr documents. Now these solr documents are going to be indexed on search engine. These solr documents are query able and can be searched easily. As we have created the documents on the objects relation bases, so it is query able even on multiple level.

This re-indexing service is needed to be running after some intervals (sync), so our data can be up to date in search Engine. Or may be called by every CRUD event (async). As shown in **Figure 17**.





**Figure 18:** Synchronizing Database with Search Engine.

# Chapter 5

---

## Validation

## CHAPTER 5: VALIDATION

This section deals with the implementation and validity of our proposed framework with help of case studies. The case study is discussed and documented in descriptive form. The Capital Cars Rental case study is discussed and validated in **Section 5.1**.

### 5.1. Case Study

This case study is explained and validated by dividing it in four sections. **Section 5.1.1** contains the Requirement specification of the automated home. **Section 5.1.2** contains the UML class diagram to present the system architecture of the required system. **Section 5.1.3** shows the retrieval of results in form of Objects. And lastly **Section 5.1.4** contains verification of the system.

#### 5.1.1. Requirement Specification

In this era of technology, home automation has been growing rapidly. This section contains the detail of hardware and its specifications for business process automation. Generic specifications are given in this section about hardware deployed in a company and along with software specification of accounting system. So, search optimization for data acquisition must be done intelligently.

**Database:** The data retrieval system is dependent upon database because all the data is to be dump in database server. The database server we are using for this case study is going to be *MySql*. All the data is going to be saved in database server and then later synchronized to search engine after some time intervals.

**ORM:** The data retrieval system for capital car rentals should have a mapper to communicate between database and a web applicator or web application and search engine. Our data retrieval system is going to be integrated with this ORM to communicate with search engine that will return objects instead of strings to ORM. The open source ORM we are using in this case study is *hibernate*. A service is being designed to convert objects from data base through ORM so the data could be dumped in search engine and synchronized.

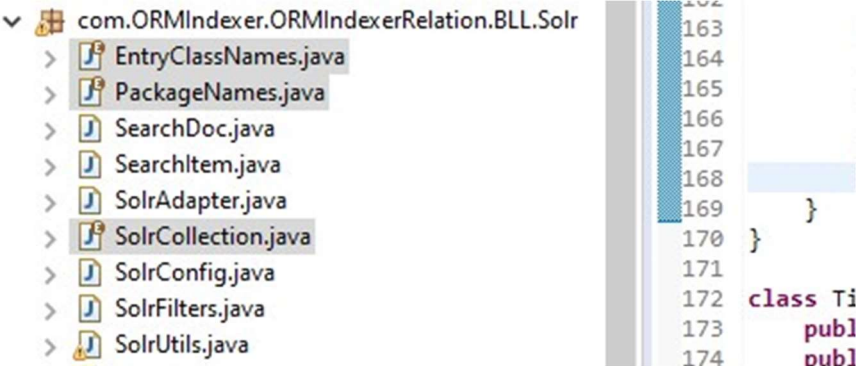
**Text Search Engine:** The accounting system should contain a search engine server for data retrieval and search purposes. The search engine should accept the connection request from our designed framework and retrieve data in objects form. This server should be configured as a text indexer and should re-index on every data sync. We have a synchronizing service, so it should be constantly connected with that service. We are using SOLR, a full text search engine which is an open sourced and elastic search supported.

**Communication Specifications:** The data acquisition system should be implemented using *Web APIs* and *JSON* data to communicate between retrieval system and search engine.

**Search parameters:** Data retrieval system should accept multi-level search depending upon objects relations. It should provide query able document with multiple filters. Also enabling users for elastic search.

**5.1.2. Integration of ORMSEM Framework with Software System**

As our proposed tool is to provide a communication between ORMs and search engine it must be integrated with software applications ORM. Importing our framework’s library into Eclipse project. Initial configuration is required for complete integration. Enumerations are required to be specified as shown in **Figure 197**. But before that SOLR must be installed and running in the background as **Figure 208**. Create a new collection to store our data in. or we can use the default collection i.e. *collection1*. As in **Figure 219**



**Figure 228:** Configure enumerations.

The screenshot shows the Solr web interface in a browser window. The address bar displays 'localhost:8983/solr/#/'. The interface includes a sidebar with navigation options: Dashboard (selected), Logging, Core Admin, Java Properties, Thread Dump, and a message 'No cores available Go and create one'. The main content area is divided into three sections:

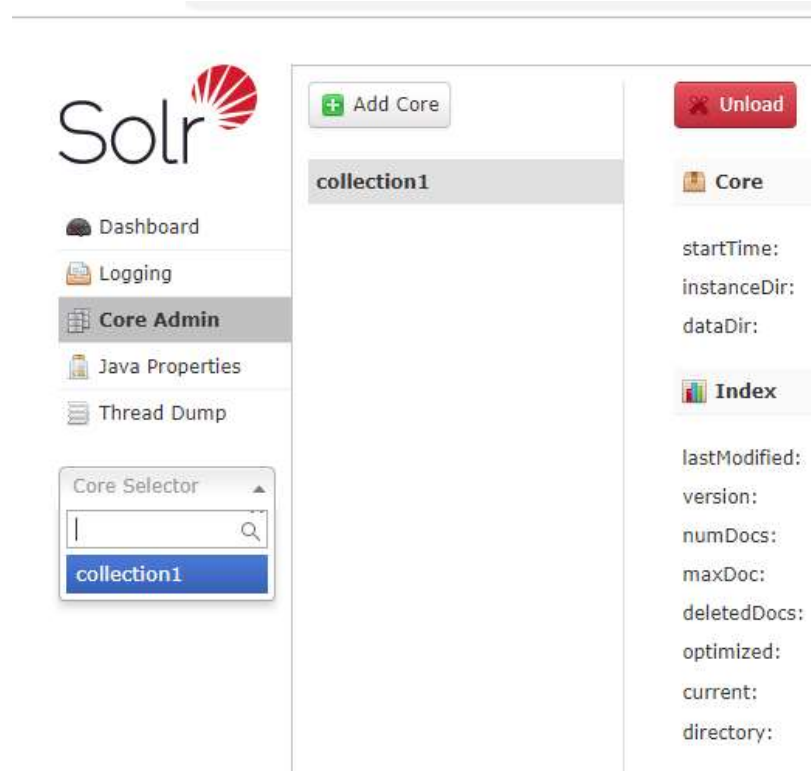
- Instance:** Shows a 'Start' button and the status 'less than a minute ago'.
- Versions:** A table listing the versions of Solr and Lucene components.
 

Component	Version
solr-spec	7.2.1
solr-impl	7.2.1 b2b6438b37073bee1fca403
lucene-spec	7.2.1
lucene-impl	7.2.1 b2b6438b37073bee1fca403
- JVM:** Shows the Java runtime environment details.
 

Property	Value
Runtime	Oracle Corporation Java HotSpot(
Processors	8
Args	-DSTOP.KEY=solrrocks -DSTOP.PORT=7983 -Djava.io.tmpdir=D:\Area Develo -Djetty.home=D:\Area Developer -Djetty.host=0.0.0.0

**Figure 23:** Running SOLR.

Solr uses a specific port i.e. 8983. Although its changeable but default is usually recommended. Solr have a web portal through which it can be managed and configured. It works on core; every core can have multiple collections and each collection can contain separate data.



**Figure 20:** Creating *Collection1*.

Solr is the main package used to contain whole communication model model the system. Containing multiple classes for different purposes. *SolrAdapter* is main with built in methods to communicate with solr search engine.

*SolrAdapter* contains method *reindexAll* which is going to act as a service to synchronize database with our search engine. ORM is going to use this method to transfer data between database server and search engine server.

### 5.1.3. Search and data retrieval

To perform search opinations we again need to communicate throws *SolrAdapter* class methods. *SearchDoc* is created with desired object type and *SolrFilters* is provided as a parameter to *SearchDoc* and create a filter on a document. Solr deals only in solr documents on adding data we have to create solr documents and on retrieval we need to create filters on document to get data in XML form and then converted to objects of their respective types' latter. We can also apply elastic search by using these searchDoc. Following are two operations we are going to perform in current case study.

1. Adding all data to solr engine throw indexing service as shown in **Figure 21**.
2. Apply search to *acglDET* object with some filter to this and get all linked objects as shown in **Figure 22**.

Status shows the progress of transformation process.

```

130     writer.close();
131 }
132
133
134 public static void reindexAllLocations()
135 {
136     SolrAdapter solrA = new SolrAdapter();
137
138     solrA.deleteAllPropertiesFromSolr(SolrCollection.collection1, EntryClassNames.Acglhead);
139
140     ArrayList<Acglhead> acglheadList = getAllAcglhead();
141     if(acglheadList != null)
142
143

```

**Figure 21:** Synchronizing database.

```

}
}
public static void displayMatch(SolrAdapter solra) {
    SearchDoc fieldsToSearch = new SearchDoc();
    fieldsToSearch.addToSearchDoc("noOfRooms", "[1 TO 3]");
    fieldsToSearch.addToSearchDoc("bathRooms", "[5 TO 5]");
    // List<String> obj = solra.searchSolrByKeyword(SolrCollection.collection1,
    // SearchRespository.Reg_Property, fieldsToSearch, "900", "propertyId");
    List<Object> obj = solra.searchSolrByFilter(SolrCollection.collection1, EntryClassNames.Reg_Property_Images,
        fieldsToSearch, PackageNames.com_reco_spring_domain_definitions_);

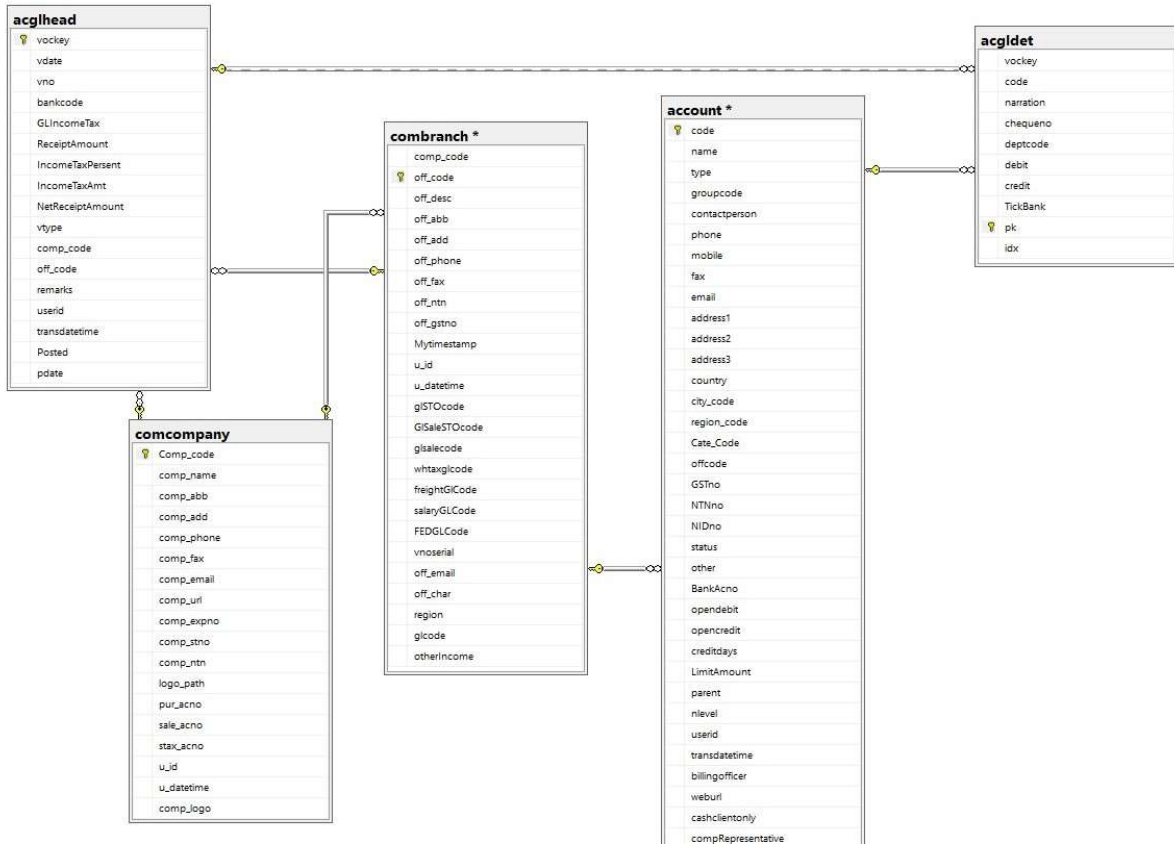
    Tracer.pushMessage("Found " + obj.size() + " rows ");
}
}

```

**Figure 22:** Performing Search.

#### 5.1.4. Verification

Results of execution and search time was compared by querying 1000 searches by traditional approaches and our current framework. Java as a programming language, hibernate as an ORM and SOLR as a text search engine was used. Real time industrial data was used which was provided by a software company looking for a better search solution. **Figure 23** is an ERD of database used for current scenario.



**Figure 23:** ERD of case study data.

**Table 1** is a record specification of data in case study.

Table name	Records
account	1370
acglhead	32835
acglDET	153395
combranch	4
Company	1

**Table 1:** ERD of case study data.



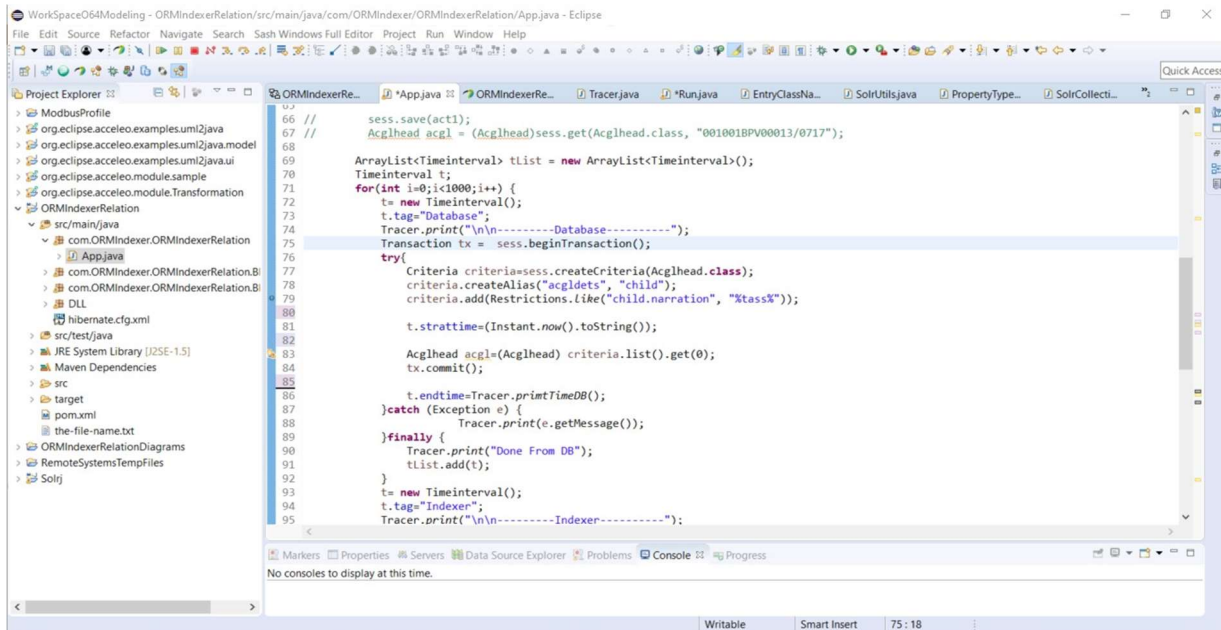


Figure 24: Querying using tradition hibernate search.

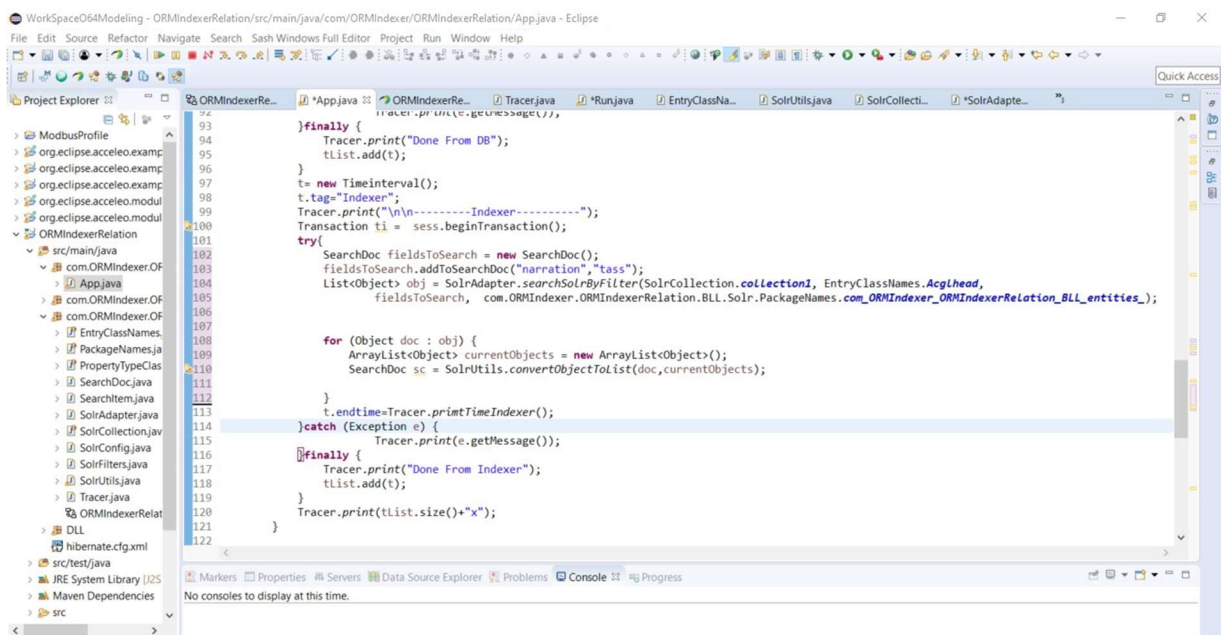


Figure 25: Querying using framework.

**Table 2** After running continues 1000 queries following were the results.

Type	Start min	Start Sec	End Min	End Sec	Start time	End time	Cost (Sec)
Database	42	12.042	42	15.677	2532.042	2535.677	3.635
Database	42	17.657	42	21.136	2537.657	2541.136	3.479
Database	42	22.724	42	25.905	2542.724	2545.905	3.181
Database	42	27.339	42	30.802	2547.339	2550.802	3.463
Database	42	32.526	42	35.868	2552.526	2555.868	3.342
Database	42	37.638	42	41.031	2557.638	2561.031	3.393
Database	42	42.687	42	45.843	2562.687	2565.843	3.156
Database	42	47.504	42	50.697	2567.504	2570.697	3.193
Database	42	52.311	42	55.417	2572.311	2575.417	3.106
Database	42	57.34	43	0.693	2577.34	2580.693	3.353
Database	43	2.757	43	6.519	2582.757	2586.519	3.762
Database	43	8.181	43	11.511	2588.181	2591.511	3.33
Database	43	13.171	43	16.618	2593.171	2596.618	3.447
Database	43	18.428	43	21.524	2598.428	2601.524	3.096
Database	43	22.966	43	25.978	2602.966	2605.978	3.012
Database	43	27.768	43	31.123	2607.768	2611.123	3.355
Database	43	32.889	43	35.941	2612.889	2615.941	3.052

**Table 2:** Querying time tradition hibernate search.

**Table 3** After running continues 1000 queries using framework following were the results

Type	Start min	Start Sec	End Min	End Sec	Start time	End time	Cost (Sec)
Framework	42	15.677	42	17.657	2535.677	2537.657	1.98
Framework	42	21.136	42	22.724	2541.136	2542.724	1.588
Framework	42	25.905	42	27.339	2545.905	2547.339	1.434
Framework	42	30.802	42	32.526	2550.802	2552.526	1.724
Framework	42	35.868	42	37.638	2555.868	2557.638	1.77
Framework	42	41.031	42	42.687	2561.031	2562.687	1.656
Framework	42	45.843	42	47.504	2565.843	2567.504	1.661
Framework	42	50.697	42	52.311	2570.697	2572.311	1.614
Framework	42	55.417	42	57.34	2575.417	2577.34	1.923
Framework	43	0.693	43	2.756	2580.693	2582.756	2.063
Framework	43	6.519	43	8.181	2586.519	2588.181	1.662
Framework	43	11.511	43	13.171	2591.511	2593.171	1.66
Framework	43	35.941	43	37.429	2615.941	2617.429	1.488
Framework	43	40.558	43	41.947	2620.558	2621.947	1.389
Framework	43	44.872	43	46.477	2624.872	2626.477	1.605
Framework	43	49.721	43	51.08	2629.721	2631.08	1.359
Framework	43	54.095	43	55.446	2634.095	2635.446	1.351

**Table 3:** Querying using Framework.

**Table 4** shows a comparison between both times consumed while executing. It is showing significant difference between both calls.

Type	Cost in sec
Average by Database (Hibernate)	1.5525
Average by Framework	1.2125
<b>Average Improvement</b>	<b>21.2</b>

**Table 4:** Querying using Framework

# Chapter 6

---

## Discussion and Limitation

## CHAPTER 6: DISCUSSION AND LIMITATION

**Section 6.1** contains a detailed discussion on proposed research work and **Section 6.2** deals with the limitations of the research.

### 6.1. Discussion

From this research, it has been analyzed that there is very trivial amount of work has been carried out to optimize search that can support object relational mapping. Researchers who tried to optimize search using indexers or search engines kept their focus on either text base search or didn't try to add ORM in the picture at all. As modern technology is advancing ORMs are more in practice then ever. Most of the researches are search specific like elastic search, text indexing or database tuning. A very little amount of work was found in which researchers tried to map these modern technologies together. Our proposed solution is a step towards uniting ORM and search engines to achieve improved results.

Motive behind this work is to provide a generic system that can communicate between ORMs and search engines, independent of object entities and save time cost of data retrieval. In order to implement it we have used an open source search engine which support web APIs so we should not be depending upon platform. It has a high compatibility to communicate and transform data between both application and support different platforms. Moreover, indexer and search engines are already heavily used in large data applications where search is required.

The proposed framework generates code artifacts for the data retrieval framework by lowering the cost of development. The model developed to represent the data retrieval framework shows higher abstraction of the system and is much smaller than expressed in code. It is less sensitive to changes in business requirement as it is easier to understand the behavior of the system, manage changes, and maintaining the application at abstract level. A large number of functionalities can be added to the system in same amount of time resulting in minimizing the cost of time and a smaller number of people are required to build the system using model driven approach. Furthermore, the search engine and ORMs used in our proposed framework is available in market as a freeware library which also reduce the cost of the overall system.

In order to validate our proposed framework, we have selected a real time case study. In which we are going to index data into search engine and import into database. A comparison will be done after evaluating the time cost difference. The purpose of choosing the case study of this size is only to validate our proposed system.

## **6.2.Limitations**

Our proposed research is a step towards the grouping of ORM and text search engines but there exist limitation and constraints. We have validated it in a case study with real time data and it shown significant results, but our entity model was not that complicated. Scenarios exists where it might have to deal with way too much complex models and deep heretical object search. In such scenario boxing and unboxing techniques my gives limited results.

# Chapter 7

---

## Conclusion and Future Work

## **CHAPTER 7: CONCLUSION AND FUTURE WORK**

Our proposed framework provides a generic solution to overcome the compatibilities issues between ORMs and search engine or indexer. It is consuming web APIs of indexer and provides a middle layer for the communication with ORM. It uses a generic concept of boxing and unboxing of objects. It is always synchronized with the help of an independent service which makes data available for search all the time. Framework support generic configurations to define which objects needs to be saved and up to what depth. Framework support a query able document that can contain a defined query in order to search not only with in the object but also sub objects. Currently it is been developed for java and supports Hibernate. As this framework is not constrained to its containing project it can be used for multiple projects to search data from same search engine. It significantly minimalizes the cost of search in large data and especially if multiple level object search is required. The results are validated on test data provided for the test case and it shows significant differences.

Future work includes making framework more generic using transformation and model driven approached which will be easy to implement and specification of entities will not be required. Currently its platform dependent and works with java and hibernate only.



## APPENDIX A

### USER MANUAL

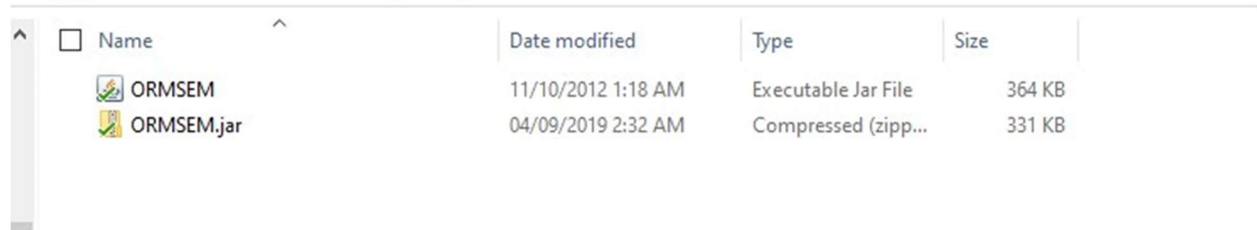
#### 1. Download Instructions

##### 1.1.ORM and Search Engine Mapper (ORMSEM)

Download ORM and search engine mapper (ORMSEM) from ORMSEM website as: “ORMSEM.zip”

Extract **ORMSEM.zip** file. You will find “ORMSEM” Folder.

In the “ORMSEM” folder, you will find two files shown in *Figure 1* below.



Name	Date modified	Type	Size
ORMSEM	11/10/2012 1:18 AM	Executable Jar File	364 KB
ORMSEM.jar	04/09/2019 2:32 AM	Compressed (zip) file	331 KB

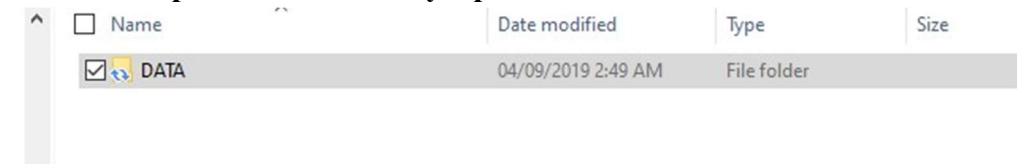
**Figure 1:** Files in “ORMSEM” folder

Add ORMSEM to java project.

##### 1.2.Sample Case Studies

Download sample data of case study from ORMSEM website. (Capital Car Rentals Data of transaction) from ORMSEM website as “Sample-DataCaseStudy.zip”

Extract **Sample- DataCaseStudy.zip** file. You will find “DATA” folder as shown in *Figure 2*.



Name	Date modified	Type	Size
DATA	04/09/2019 2:49 AM	File folder	

**Figure 2:** Sample Case Study folder

Open this folder and you will find a file named “DBScript.sql” for database tables and data.

#### 2. Prerequisites for ORMSEM

**Java Runtime Environment (JRE) version 8 or above** is required to be installed in order to create a java project.

**Sif4J.jar** need to add as a dependency to java project to make ORMSEM useable.

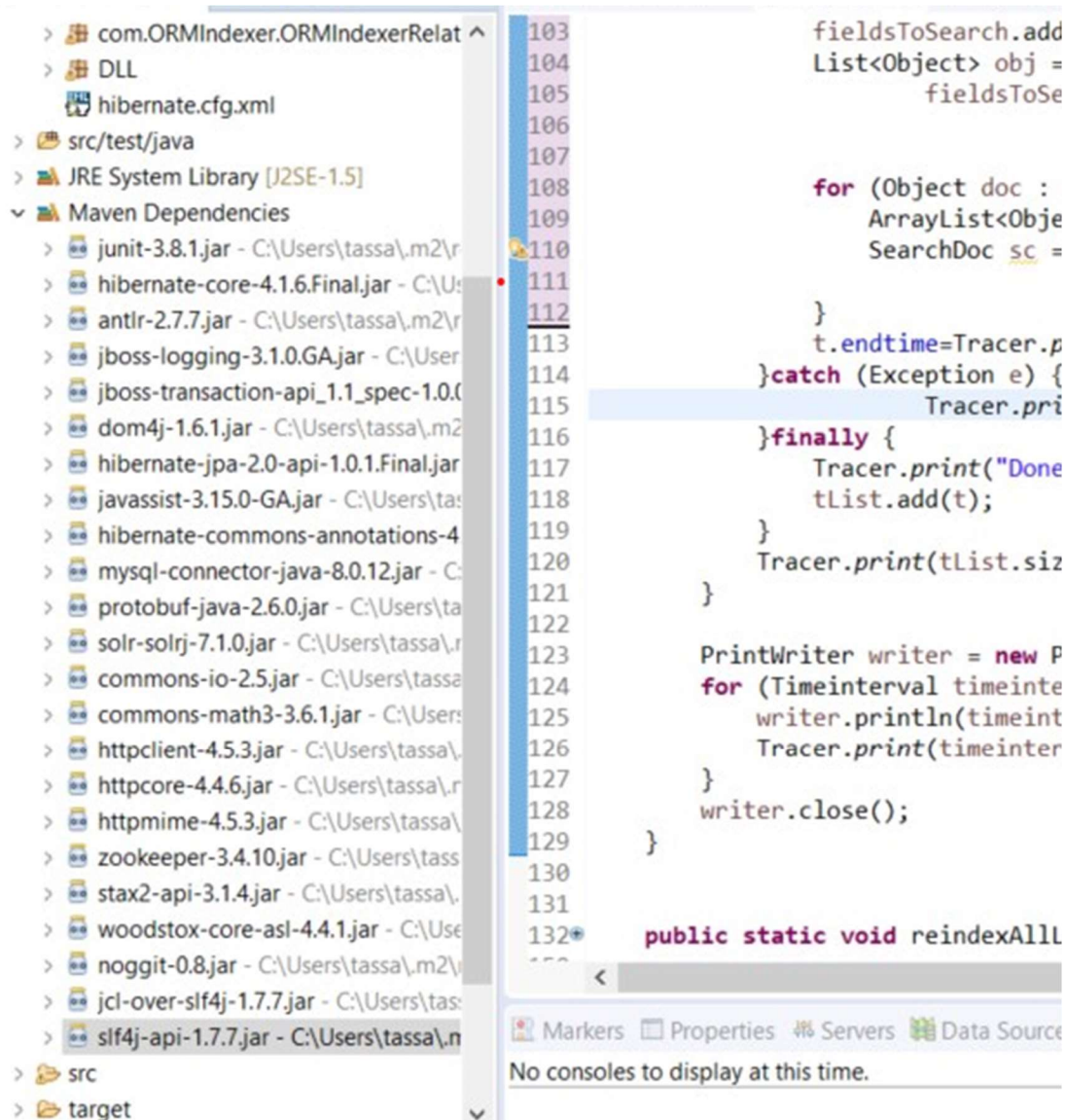
**SOLR search engine** is also required to use ORMSEM.

We have tested ORMSEM on all neon or higher versions of eclipse. However, we are confident that ORMSEM can also be executed on previous versions of eclipse. And using without IDE it will work fine too.

### 3. Execution of ORMSEM

Copy and add ORMSEM to your project with other dependencies.

As shown in *Figure 3*.



**Figure 3:** ORMSEM as dependency

The ORMSEM contains three main functionalities.

- **Configurations:** Configure ORMSEM into your project by adding Enums in certain packages.
- **Index All:** A service that sync database with solr search engine.
- **Search:** Search with in object and child objects comparing attributes.
- 

#### 3.1. Configurations

Add a package named “com.ORMIndexer.ORMIndexerRelation.BLL.Solr”.

Add Enum “EnterClassName” with entity name that need to be indexed into created package.  
As shown in **Figure 4**

```
*EntryClassNames.java
1 package com.ORMIndexer.ORMIndexerRelation.BLL.Solr;
2
3 public enum EntryClassNames {
4
5     Acglhead, AcglDET, Account, Combranch, Comcompany
6 }
7
```

**Figure 4:** EnterClassName as Enum

Add Enum “PackageNames” with entities package name that need to be indexed.  
As shown in **Figure 5**

```
*EntryClassNames.java PackageNames.java
1 package com.ORMIndexer.ORMIndexerRelation.BLL.Solr;
2
3 public enum PackageNames {
4     com_ORMIndexer_ORMIndexerRelation_BLL_entities_
5 }
6
```

**Figure 5:** PackageNames as Enum

Add Enum “Sol Collection” with collection name on solr server.  
As shown in **Figure 6**

```
1 package com.ORMIndexer.ORMIndexerRelation.BLL.Solr;
2
3 public enum SolrCollection {
4     collection1
5 }
6
```

**Figure 6:** SolrCollection as Enum

### 3.2.Indexed All

Consume a method `reindexAllLocations()` in to sync database and solr data.  
Firstly, import data from “DBscript.sql” file to **mySql** database and start **solr** server.  
Import data by executing provided query with data in it. **Figure 7**

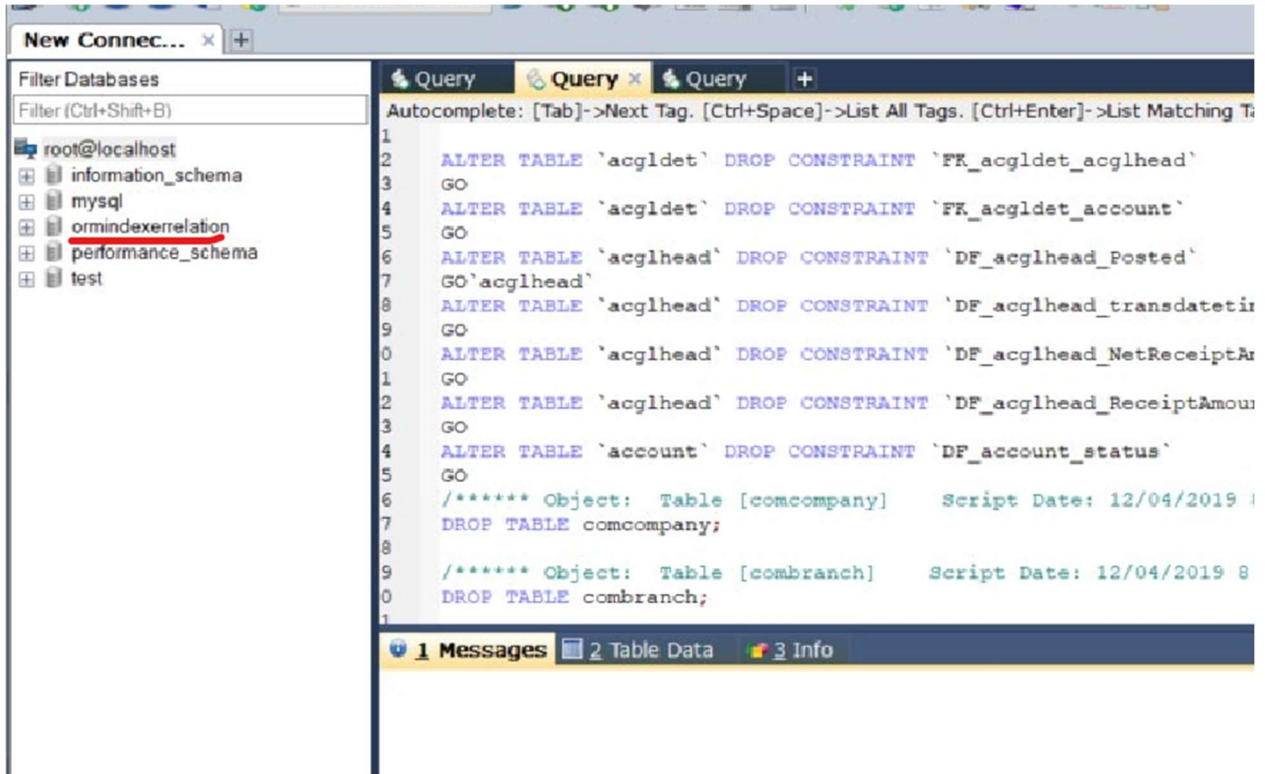


Figure 7: Data import

### 3.3.Search

Search can be performed by consuming methods in SolrAdapter class. Figure 8 shows a brief concept of implementing search method.

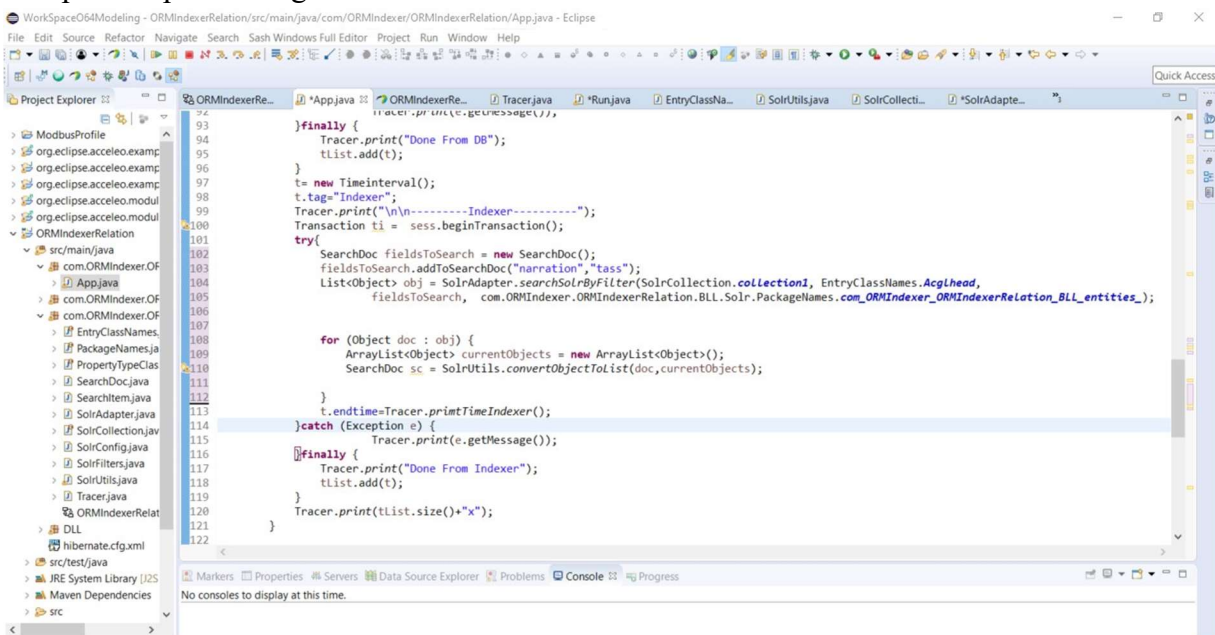
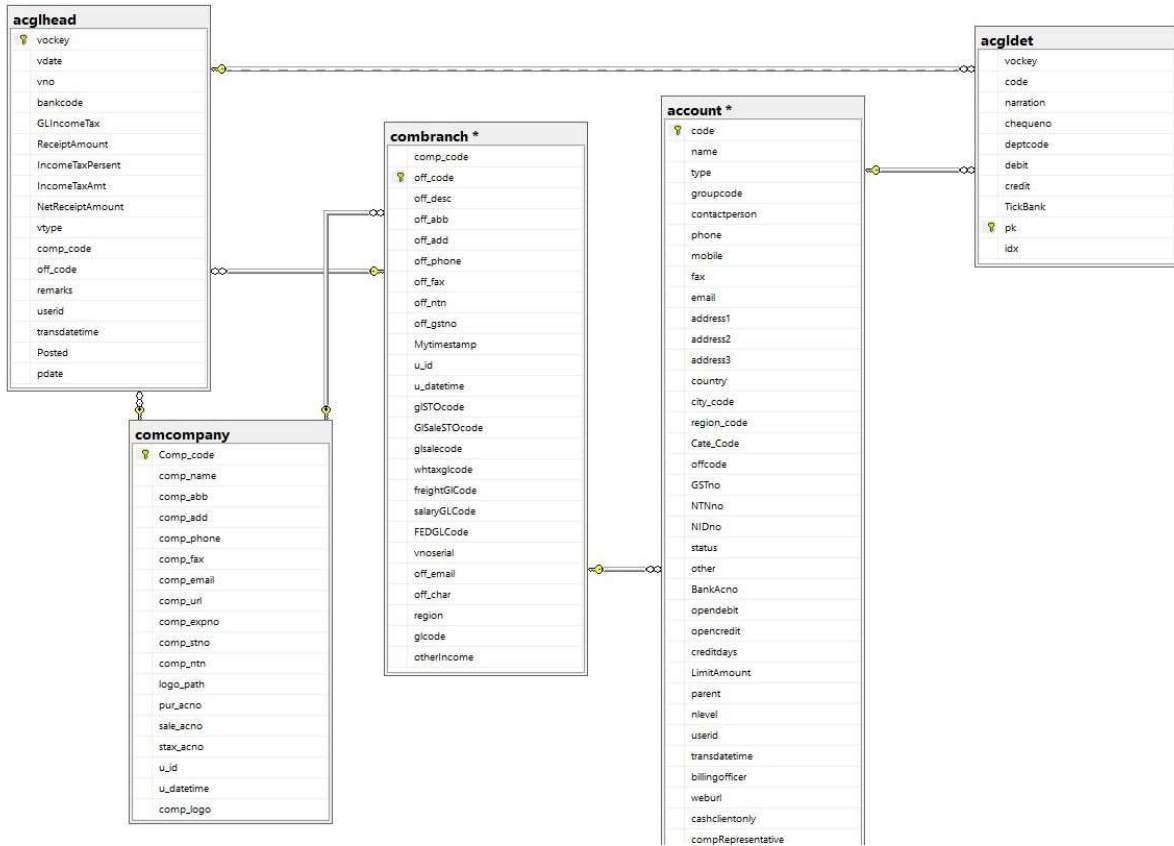


Figure 8: Search method consumed

### 3.4.Verification

Results of execution and search time was compared by querying 1000 searches by traditional approaches and our current framework. Java as a programming language, hibernate as an ORM and SOLR as a text search engine was used. Real time industrial data was used which was provided by a software company looking for a better search solution. **Figure 9** is an ERD of database used for current scenario.



**Figure 9:** ERD of case study data.

**Table 1** is a record specification of data in case study.

Table name	Records
account	1370
acglhead	32835
acglDET	153395
combranch	4
Company	1

**Table 1:** ERD of case study data.



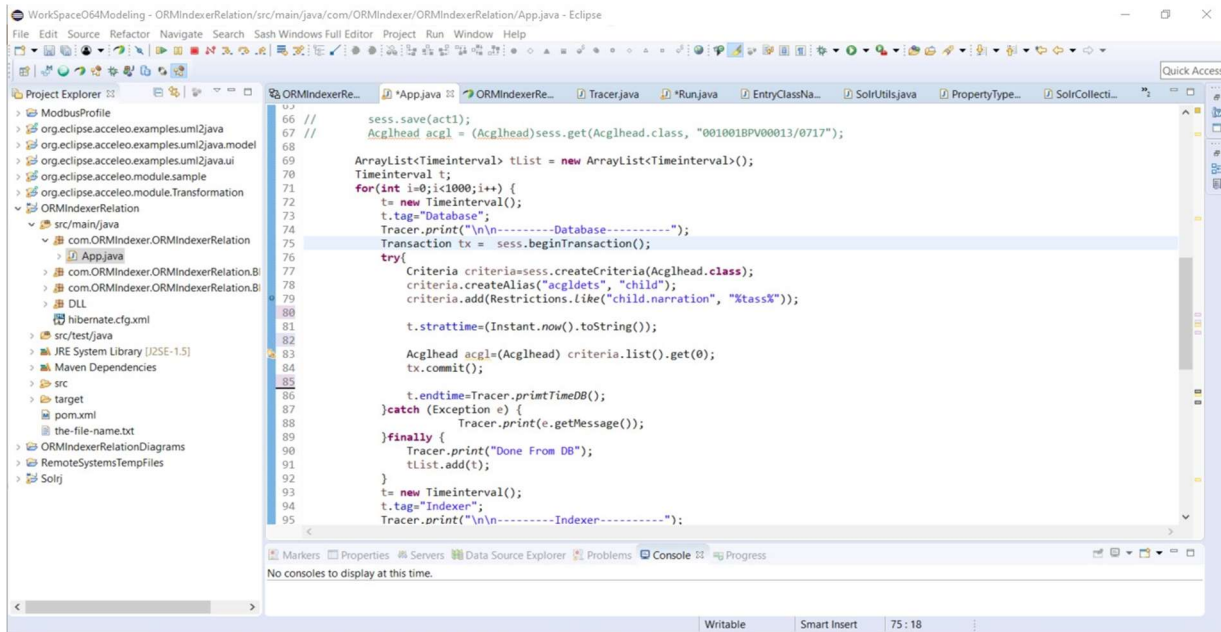


Figure 10: Querying using tradition hibernate search.

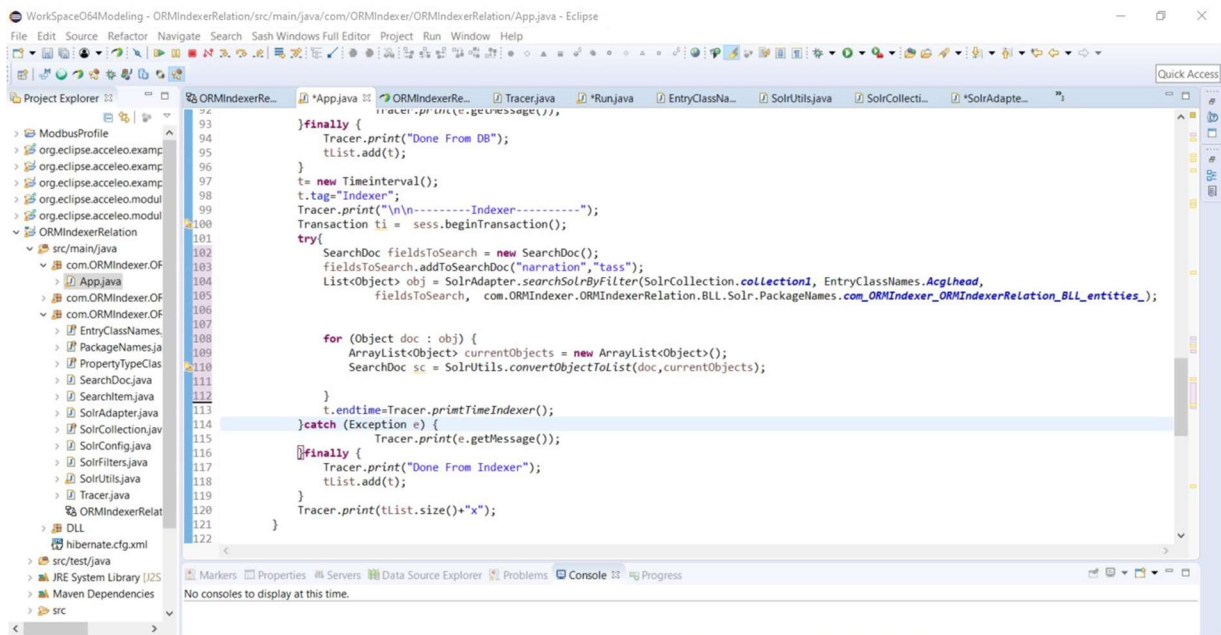


Figure 11: Querying using framework.

**Table 2** After running continues 1000 queries following were the results.

Type	Start min	Start Sec	End Min	End Sec	Start time	End time	Cost (Sec)
Database	42	12.042	42	15.677	2532.042	2535.677	3.635
Database	42	17.657	42	21.136	2537.657	2541.136	3.479
Database	42	22.724	42	25.905	2542.724	2545.905	3.181
Database	42	27.339	42	30.802	2547.339	2550.802	3.463
Database	42	32.526	42	35.868	2552.526	2555.868	3.342
Database	42	37.638	42	41.031	2557.638	2561.031	3.393
Database	42	42.687	42	45.843	2562.687	2565.843	3.156
Database	42	47.504	42	50.697	2567.504	2570.697	3.193
Database	42	52.311	42	55.417	2572.311	2575.417	3.106
Database	42	57.34	43	0.693	2577.34	2580.693	3.353
Database	43	2.757	43	6.519	2582.757	2586.519	3.762
Database	43	8.181	43	11.511	2588.181	2591.511	3.33
Database	43	13.171	43	16.618	2593.171	2596.618	3.447
Database	43	18.428	43	21.524	2598.428	2601.524	3.096
Database	43	22.966	43	25.978	2602.966	2605.978	3.012
Database	43	27.768	43	31.123	2607.768	2611.123	3.355
Database	43	32.889	43	35.941	2612.889	2615.941	3.052

**Table 2:** Querying time tradition hibernate search.

**Table 3** After running continues 1000 queries using framework following were the results

Type	Start min	Start Sec	End Min	End Sec	Start time	End time	Cost (Sec)
Framework	42	15.677	42	17.657	2535.677	2537.657	1.98
Framework	42	21.136	42	22.724	2541.136	2542.724	1.588
Framework	42	25.905	42	27.339	2545.905	2547.339	1.434
Framework	42	30.802	42	32.526	2550.802	2552.526	1.724
Framework	42	35.868	42	37.638	2555.868	2557.638	1.77
Framework	42	41.031	42	42.687	2561.031	2562.687	1.656
Framework	42	45.843	42	47.504	2565.843	2567.504	1.661
Framework	42	50.697	42	52.311	2570.697	2572.311	1.614
Framework	42	55.417	42	57.34	2575.417	2577.34	1.923
Framework	43	0.693	43	2.756	2580.693	2582.756	2.063
Framework	43	6.519	43	8.181	2586.519	2588.181	1.662
Framework	43	11.511	43	13.171	2591.511	2593.171	1.66
Framework	43	35.941	43	37.429	2615.941	2617.429	1.488
Framework	43	40.558	43	41.947	2620.558	2621.947	1.389
Framework	43	44.872	43	46.477	2624.872	2626.477	1.605
Framework	43	49.721	43	51.08	2629.721	2631.08	1.359
Framework	43	54.095	43	55.446	2634.095	2635.446	1.351

**Table 3:** Querying using Framework.

**Table 4** shows a comparison between both time consumed while executing. It showing significant difference between both calls.

Type	Cost in sec
Average by Database (Hibernate)	1.5525
Average by Framework	1.2125
<b>Average Improvement</b>	<b>21.2</b>

**Table 4:** Querying using Framework



## REFERENCES

- [1] Jiang Wenpei ,SQL Server 2005 Practical Tutorial [M],Post & Telecommunication Press, 2006.12
- [2] De Luke, Microsoft SQL Server 7 Performance Optimization [M]. Machinery Industry Press, 2000.8.
- [3] Working Conference on Mining Software Repositories, An Empirical Study on the Practice of Maintaining Object-Relational Mapping Code in Java Systems, 2016 IEEE
- [4] WANG Hong-man<sup>1</sup>, WANG He-wei<sup>1</sup>, Design and implementation of SOLR-based information retrieval system for value-added services, 2008 IEEE
- [5] T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora. Detecting performance anti-patterns for applications developed using object-relational mapping. In Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pages 1001–1012, 2014.
- [6] Tse-Hsun Chen, An Empirical Study on the Practice of Maintaining Object-Relational Mapping Code in Java Systems 2015, Working Conference on Mining Software Repositories
- [7] C. A. Curino, H. J. Moon, and C. Zaniolo. Graceful database schema evolution: The prism workbench. in Proc. VLDB Endow., 1(1):761–772, Aug. 2008
- [8] D. Qiu, B. Li, and Z. Su. An empirical analysis of the co-evolution of schema and code in database applications. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pages 125–135, 2013.
- [9] ZereturnAround. Java tools and technologies landscape for 2015. <http://zereturnaround.com/rebellabs/java-tools-and-technologies-landscape-for-2014/>. Last accessed March 10 2015
- [10] J. Community. Hibernate. <http://www.hibernate.org/>. Last accessed March 10 2016.
- [11] Lujia Ma, Wei Bao,Wugedele Bao, Wuriga Yuan, Tao Huang, XiaoBing Zhao, A Mongolian Information Retrieval System Based on Solr, 2017.
- [12] Nemanja Kojic and Dragan Milicev, A Survey of Object-Relational Transformation Patterns for High-performance UML-based Applications.
- [13] Yang Yang and Hongyun Ning, Block Linked List Index Structure for Large Data Full Text Retrieval, 2107.

- [14] Tse-Hsun Chen, Weiyi Shang, Jinqiu Yang, Ahmed E. Hassan, Michael W. Godfrey, Mohamed Nasser and Parminder Flora, An Empirical Study on the Practice of Maintaining Object-Relational Mapping Code in Java Systems, 2016.
- [15] Chitra Babu and Gunasingh G, DESH: Database Evaluation System With Hibernate ORM Framework, 2016.
- [16] Junwen Yang and Cong Yan, How not to structure your database-backed web applications: a study of performance bugs in the wild, 2018.
- [17] Harpreet Kaur and Vishal Gupta, Indexing Process Insight and Evaluation.
- [18] Platonov Y. and Artamonova E., Managing Big Data Using Specified System Polar, 2017.
- [19] Wenlin Pan, Mapping ORM into SWRL, 2013.
- [20] Shailender Kumar, Rahul Rishi and Rupender Duggal, Implementation of Temporal Functionality in Objects with Roles Model (TF-ORM).
- [21] Peng Li, Object-Relational event (ORE) middleware for push-based content delivery from the application tier.
- [22] QI Chunxia, On Index-based Query in SQL Server Database, 2016.
- [23] Michele Artini, Claudio Atzori, Alessia Bardi, Sandro La Bruzzo and Paolo Manghi, TagTick: A Tool for Annotation Tagging over Solr indexes.
- [24] Joseph Armas, Optimization of Code Lines and Time of Access to Information through ObjectRelational Mapping (ORM) Using Alternative Tools of Connection to Database Management Systems (DBMS) 2017 2nd International Conference on System Reliability and Safety
- [25] Jiawei Han, Join Index Hierarchy: An Indexing Structure for Efficient Navigation in Object-Oriented Databases
- [26] Yang Lai, An Efficient Data Mining Framework on Hadoop using Java Persistence API. 2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)
- [27] Ömer Yüksel, Jerry den Hartog, Sandro Etalle, 2016, “Reading between the Fields: Practical, Effective Intrusion Detection for Distributed Control System”, ACM Proceedings of the 31st Annual ACM Symposium on Applied Computing.