



Sudan University of Science and Technology

College Of Graduate Studies

Master of Information Technology



Performance Improvement for Enterprise Resource Planning Systems using Big Data Technologies

تحسين الأداء لأنظمة تخطيط المؤسسة باستخدام تقنيات البيانات الضخمة

A Thesis Submitted in Partial Fulfillment of the requirements for the degree of M.Sc.in
Information Technology

By

Samia Abdelmonim Osman

Supervisor

Dr. Talaat Mohiuddin Wahbe

September 2018

الآية

قال تعالى:

"نَرْفَعُ دَرَجَاتٍ مَّن نَّشَاءُ ^{قُلْ} وَفَوْقَ كُلِّ ذِي عِلْمٍ

عَلِيمٍ" (76) يوسف

صدق الله العظيم

Acknowledgments

First I would like to thank our God for giving me knowledge, success and strength to finishing this study ,The success and final outcome of this project required a lot of guidance and assistance from many p eople and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Dr. Hisham Abdulla Manssor, for providing me an opportunity to do the project work and giving me all support and guidance which made me complete the project duly. I am extremely thankful to him for providing such a nice support and guidance.

I owe my deep gratitude to my husband, who give me the time and full support, till the completion of my project work. And giving valuable suggestions and comments on my work. Finally, I would like thank my family and especially my mother and sister for their support.

Abstract

During Enterprise Resource Planning system lifetime, the response time increasing while the amount of data is increasing because data is retrieving from the relational database, where the more data to be processed the more time needs to response. A new data storage system has been developed called NoSQL, which stands for “Not Only SQL or Non SQL systems”. NoSQL systems have been designed for large-scale data storage and apply parallel data processing. To identify and develop the optimal solution to reduce the time of huge data processing in the Odoo framework.

The proposed solution is to import data from Odoo database and store it in NoSQL data storage and build Odoo module that is able to read and write data from both PostgreSQL Database (which is not distributed database) and suitable NoSQL database. This research aims to solve the Odoo performance latency problem, this reduce storage costs, because it is cheaper to store archived data based on NoSQL infrastructure, Open-source software and easy to extending in future, Also can use data for mining “data mining”.

المستخلص

خلال فترة تطبيق نظام تخطيط موارد المؤسسة ، يزداد زمن الاستجابة بينما يزداد مقدار البيانات نظرًا لاستعادة البيانات من قاعدة البيانات العلائقية ، حيث تتم معالجة المزيد من البيانات التي تحتاج إلى مزيد من الوقت للاستجابة. وقد تم تطوير نظام جديد لتخزين البيانات يسمى NoSQL ، والذي يرمز إلى "ليس فقط SQL أو أنظمة غير SQL". تم تصميم أنظمة NoSQL لتخزين البيانات على نطاق واسع وتطبيق معالجة البيانات المتوازية. لتطوير الحل الأمثل لتقليل وقت معالجة البيانات الضخمة في إطار عمل Odoo. والحل المقترح هو استيراد البيانات من قاعدة بيانات Odoo وتخزينها في تخزين البيانات NoSQL وبناء وحدة Odoo القادرة على قراءة وكتابة البيانات من كل من قاعدة بيانات PostgreSQL (التي ليست قاعدة بيانات موزعة) وقاعدة بيانات NoSQL مناسبة. يهدف هذا البحث إلى حل مشكلة انخفاض أداء Odoo. يمكن أن يقلل هذا من تكاليف التخزين ، لأنه أرخص لتخزين البيانات المؤرشفة على أساس البنية التحتية NoSQL. ومفتوح المصدر وسهل التوسع في المستقبل ، ويمكن أيضا استخدام البيانات في التنقيب "data mining".

Table of Contents

الأية	i
Acknowledgments	ii
Abstract	iii
المستخلص	iv
Table of Contents	v
List of Figures	viii
List of Tables:	ix
List of abbreviations:	ix
CHAPTER ONE	1
Introduction	1
1.1 Overview:	1
1.2 Background:	1
1.3 Problem Statement:	1
1.4 Research Objectives	2
1.5 Scope of Work:	2
1.6 Proposed Solution:	2
1.7 Research Methodology:	3
1.8 Thesis Organization:	4
1.9 Contribution:	4
1.10 Big Data:	4
1.11 Characteristics of Big Data:	5
CHAPTER TWO	6
Related work and Literature Review	6
2.1 Introduction	6
2.2 Enterprise Resource Planning Systems:	6
The Types of ERP Systems:	6
Open-source Software (OSS):	6
Odoo Enterprise Resource Planning Systems:	7
Odoo Software & Architecture:	7
Odoo Applications and Modules:	7

Odoo Features:	7
Disadvantages of Odoo:	8
2.3 Hadoop:	8
2.3.1 Hadoop consists of three main components [6]:	8
2.3.2 Hadoop Distributed File System	8
2.3.3 Architecture of Hadoop Distributed File System:	8
2.3.4 MapReduce Framework:	9
2.3.5 MapReduce Consists of Three Main Steps:	9
2.4 NoSQL Database Storage:	10
2.4.1 NoSQL characteristics:	10
2.4.2 A Comparison between SQL and NoSQL databases:	10
2.4.3 Common Type of NoSQL Database:	11
2.4.4 Popular NoSQL Databases:	12
2.5 Literature review:	12
2.5.1 Solving and Processing Big Data Problem using Hadoop	12
2.5.2 Performance Evaluation and Data Migration to Hadoop	13
2.5.3 A performance Comparison of SQL and NoSQL Databases.	14
2.6 Summary:	15
CHAPTER THREE	16
Research Methodology	16
3.1 Introduction	16
3.2 Research Methodology:	16
3.3 System Architecture:	16
3.3.1 Functional Requirement:	17
3.3.2 Non-functional Requirement:	17
3.3.3 Hbase fetchers:	18
3.3.4 Zookeeper:	18
3.3.5 Phoenix:	18
3.4 System Implementation:	19
3.4.1 Hadoop installation and Configuration:	19
3.4.2 Hbase Configuration	20
3.4.3 Data collection:	20
3.4.4 Data Analysis:	21

3.4.5 Import Data from PostgreSQL to Hbase Databases in Hadoop Ecosystem:	21
3.5 Summary:	25
CHAPTER FOURE.....	26
System Design and Implementation.....	26
4.1 Introduction:	26
4.2 Development:	26
4.2.1 Odoo ORM:	26
4.2.2 The benefits of ORM technology	26
4.2.3 Disadvantages of ORM:.....	26
4.2.4 JDBC concept:	27
4.2.5 Create ORM Method (CRUD):.....	27
4.2.5.1 Create:	28
4.2.5.2 Update:.....	28
4.2.5.3 Delete:	29
4.3 Start system servers and web interfaces:	29
4.4 Hadoop control panel:	32
4.5 Fetch Hbase Table Data in Apache Phoenix:	33
4.6 Hadoop Cluster Implementation.....	34
4.7 Consistency Test Evaluation:	36
4.8 Database Systems Evaluation:.....	38
4.9 Summary:	39
CHAPTER FIVE.....	40
Conclusion and Recommendations	40
5.1 Conclusions and Lessons Learned:	40
5.2 Recommendations:	40
6.1 References:	41

List of Figures

FIGURE 1.1: RESEARCH METHODOLOGY	3
FIGURE 1.1: MAPREDUCE EXAMPLE	9
FIGURE 3.1: SYSTEM ARCHITECTURE.....	17
FIGURE 3.2: CONFIGURE REPLICATION FACTOR IN HDFS-SITE.XML FILE.....	20
FIGURE 3.3: PYTHON WIZARD USED TO GENERATE DATA.....	20
FIGURE 3.4: PYTHON SCRIPT USED TO GENERATE DATA	21
FIGURE 3.5: CREATE EMPLOYEE TABLE IN HBASE.....	22
FIGURE 3.6: IMPORTING DATA FROM POSTGRESQL TO HBASE USING SQOOP	22
FIGURE 3.7: DATA IMPORTING SUCCESSFULLY	23
FIGURE 3.8: SCAN “EMPLOYEE” TABLE	23
FIGURE 3.9: SCAN “EMPLOYEE” TABLE WITH SPECIFIC CONDITION	24
FIGURE 3.10: TABLE IN POSTGRESQL	24
FIGURE 3.11: TABLE IN HBASE (HADOOP CLUSTER)	24
FIGURE 4.1: OPTIMIZATION IN CREATE FUNCTION	28
FIGURE 4.2: OPTIMIZATION IN UPDATE FUNCTION.....	28
FIGURE 4.3: OPTIMIZATION IN DELETE FUNCTION	29
FIGURE 4.4: START HDFS AND MAPREDUCE (YARN).....	29
FIGURE 4.5: RUN ZOOKEEPER SERVER.....	30
FIGURE 4.6: STARTING HBASE SERVER.....	30
FIGURE 4.7: OPEN HBASE SHELL	30
FIGURE 4.8: RUN JPS IN MASTER MACHINE	30
FIGURE 4.9: RUN JPS IN SLAVE MACHINE	31
FIGURE 4.10: START ODOO SERVER.....	31
FIGURE 4.11: START PHOENIX	31
FIGURE 4.13: HADOOP CLUSTER.....	32
FIGURE 4.12: HADOOP DATA DISTRIBUTION (TWO NODES).....	32
FIGURE 4.14: CREATE VIEW IN PHOENIX TO FETCH HBASE TABLE DATA.....	33
FIGURE 4.15: SHOW PHOENIX TABLES	33
FIGURE 4.16: RETRIEVE HBASE TABLE DATA IN APACHE PHOENIX	34
FIGURE 4.17: THE RESULT OF QUERY IN POSTGRESQL	36
FIGURE 4.18: THE RESULT OF QUERY IN PHOENIX.....	36
FIGURE 4.19: QUERY IN POSTGRESQL RETURN NUMBER OF RECORDS.....	37
FIGURE 4.20: QUERY IN PHOENIX RETURN NUMBER OF RECORDS.....	37
FIGURE 4.21: TIME TO RETURN SPECIFIC ROWS IN TABLE FROM POSTGRESQL	38
FIGURE 4.22: TIME TO RETURN ALL ROWS IN TABLE FROM POSTGRESQL.....	39

List of Tables:

Table Number	Table Name	Page Number
Table (2.1)	Comparison between SQL and NoSQL	10
Table(4.1)	Cluster Nodes Specification	35
Table(4.2)	PostgreSQL Database Specification	35
Table(4.3)	Sample Data Record Counts:	35

List of abbreviations:

Abbreviation	Explanation
ERP	Enterprise Resource Planning
ORM	Object Relational Mapper
NoSQL	Not Only SQL
CRUD	(Create,Read,Update,Delete)
SQL	Structure Query Language
HDFS	Hadoop Distributed File System
API	Application Programming Interface
Sqoop	Sql + Hadoop

Chapter One

Introduction

CHAPTER ONE

Introduction

1.1 Overview:

This chapter introduces the research work, states the problem, defines research objectives, significant and describes the methodology.

1.2 Background:

Enterprise Resource Planning Systems Odoo / OpenERP is a comprehensive suite of business applications including Sales, CRM, Project management, Warehouse management, Manufacturing, Financial management, and Human Resources just to name a few. Odoo / OpenERP offers a choice of over a thousand modules. Odoo / OpenERP is available in the cloud or on-site. In the past it is most suited for small to mid-sized companies, but in today's time Odoo / OpenERP used in big companies. With more than a thousand downloads/installations per day, Odoo / OpenERP is one of the most used open source solution in the world. It has a dynamic community, is flexible, and can be adapted to your needs. It can be put in production rapidly thanks to its modularity and is easy to use [1].

1.3 Problem Statement:

In most organizations, the data size doubles every two years, and most relational databases like 'PostgreSQL' are not distributed database, that works on one server. The problem lies in large companies when they decide to carry a large number of data into the database and after long-term data archiving. Its performance is decreasing due to the growing amount of data.

1.4 Research Objectives

1. The research aims to Implement Odoo / OpenERP in large companies without Odoo latency problem.
2. To identify and develop the optimal solution to reduce the time of huge data processing in the Odoo framework.
3. To allows the system to continue working if the server / node fails by applying the distributed environment and choose a NoSQL data storage system that meets the requirements.
4. To evaluate the performance of the proposed solution in terms of time and compare it with the current approach.

1.5 Scope of Work:

Building the system that able to process and store the extremely large data in distributed environment.

1.6 Proposed Solution:

Data is growing rapidly, we need to archive all data. Data archiving is performed by transferring data from primary storage to secondary storage, where we can transfer large data from Odoo to Hadoop system using SQOOP. It is an import / export tool that allows you to transfer data between the SQL database and the NoSQL database in the Hadoop cluster. Then, create a new Odoo module that enables Odoo Object Relational Mapper (ORM) schema to communicate with the NoSQL database system. Finally, the goal is to reduce processing time for storing and retrieving data. After archiving that data, we get better performance and lower cost for hardware and software. This step can reduce storage costs because it is cheaper to store archived data on the Hadoop structure.

1.7 Research Methodology:

The research methodology that will be employed in the research work is described and presented in detail.

- i. Data collection: use tool to generate data or python scripts to create millions of data records in Odoo system.
- ii. Data analysis: identify the suitable NoSQL database type to store imported data from Odoo, by Using Sqoop to transfer all the data from PostgreSQL to Hbase data databases in Hadoop ecosystem, Sqoop recommend because it is faster to deploy and to make updates in a production environment.
- iii. Development: build Odoo module that is able to reading and writing data from both PostgreSQL Database (which is not distributed database) and suggested NoSQL database.
- iv. Evaluation: evaluate the performance of PostgreSQL and Hbase in term of data retrieving time.

Figure 1.1 display methodology which research follow it.



Figure 1.1: Research Methodology

1.8 Thesis Organization:

The rest of the thesis is organized as follows:

- Chapter Two: Related work and Literature reviews.
- Chapter Three: Research Methodology.
- Chapter Four: System design and implementation.
- Chapter Five: Conclusions and Recommendations

1.9 Contribution:

Develop and build new Odoo module that is able to reading and writing data from both PostgreSQL Database (which is not distributed database) and suggested NoSQL database. To ensure the changes reflect to Odoo and new system in the same time.

1.10 Big Data:

With the spread of the Internet, automated systems have been involved into daily operations in different fields. All of this generating huge amount of data from several sources in multiple forms like text, audio, video, documents, emails, images and posts on Facebook or Tweeter, etc. this data release the expression of Big Data. [2][3] With reason of large amount of data, traditional data processing and storage technologies are insufficient. After long-term of data archiving the business will influence negatively.

The expansion and the availability of huge amount of data either structured, semi structured or unstructured described by the term 'Big Data'. So, to optimize our business processes need more data to collect more accurate result. [2]

Another definition of big data is data sets that are so large and complicated that traditional data-processing application software are unqualified to handle with them. [4]

1.11 Characteristics of Big Data:

The term of Big Data do not only refer to the data with a big size. The definition of Big Data included four concepts: Volume, Velocity, Variety and Veracity. So data possesses large volume, generated with high velocity, comes from a variety of sources and formats and having great uncertainty in quality and fidelity is referred as Big Data. [3]

Other definition of big data comes with six characteristics:

- Volume: Through the years we have a lot of online transactions stored, this responsible.
- Velocity: Data is arriving in fast speeds and must be dealt with in appropriate time.
- Variety: Data comes in various types of format structured, unstructured and semi structured. Managing and processing this different variety of data is so hard.
- Veracity: Obtaining data accuracy is a challenge. Due to data can be inconsistent and conflicting.
- Complexity: Data comes from numerous sources. So need to prepare, cleanse and transform data over systems.
- Value: Extract values from Big Data is a big issue because that enhancing the organization's business.

Chapter Two

Related work and Literature Review

CHAPTER TWO

Related work and Literature Review

2.1 Introduction

This chapter describes Hadoop , ERP systems , NoSQL Database and reviews the current literature.

2.2 Enterprise Resource Planning Systems:

ERP involves all of the processes that are important to run a business, including inventory, order management, accounting, human resources and customer relationship management (CRM). ERP software integrates these functions into one complete system to perform processes across the organization. [5]

The Types of ERP Systems:

On-Premise ERP: Everything is stored, installed and managed locally in the organization.

Cloud ERP: Hosting company manages the ERP system and its data. The organization's user can access the ERP system via internet.

Hybrid: Hybrid ERP system combines both On-Premise and Cloud ERP systems.

Open-source Software (OSS):

Open source systems are systems which their source code are available for users, and based on their requirements can customize the source code.

Odoo Enterprise Resource Planning Systems:

Odoo is one of first Open Source ERP All-In-One solutions meet your company's requirements. Odoo have developed 30 main modules which are regularly upgraded and easy to use.

Odoo Software & Architecture:

Odoo uses Python scripting and PostgreSQL database. The software is accessed via a web browser in a one page app developed in JavaScript.

Odoo Applications and Modules:

Odoo is extensible architecture. A large number of freelancers and organizations develop Odoo Modules and place them in the marketplace for sale or to be downloaded for free. Odoo has about 30 core modules and more than 5000 community modules. Main modules are available in all active versions such as 9.0, 10.0 and 11.0.

Odoo Features:

- Open source system.
- Web.
- Extensible architecture.
- Its easy integration and customization options.
- Download all the official modules for free.
- Flexible and easy to use.
- Odoo is one of the fastest growing software its database runs on PostgreSQL.
- Able to integrating with other platforms.
- The travel through the pages looks really simple.
- The software is always up to date with the improvement in the technology.
- The managers able to monitoring the performance indicator by progressing monitor and then making decision.

Disadvantages of Odoos:

Their compatibility problem when a new version released, user needs to solve the conflict of code and some data migration issues.

2.3 Hadoop:

Hadoop is a collection of open-source software, a programming framework built using Java language. Hadoop is developed to execute distributed data processing over hardware. The developers, even without any distributed or parallel programming skills, enables to write distributed data processing jobs.

2.3.1 Hadoop consists of three main components [6]:

- Hadoop Distributed File System
- MapReduce Framework
- Yet Another Resource Negotiator (YARN)

2.3.2 Hadoop Distributed File System

HDFS is the key component of Hadoop. It manages, stored and replicated huge data set in a cluster reliably. HDFS is designed to execute fast read operations but the write operations are slower, so HDFS is optimal for dealing with data that has the access pattern write-once/read-many. Also HDFS partitions the files into blocks and distributes these blocks among the cluster's nodes.

2.3.3 Architecture of Hadoop Distributed File System:

There are two types of nodes:

- NameNode “the master node”: NameNode stores the meta-data and identifies the DataNodes that are responsible for storing each block. The client asks the NameNode to return the DataNodes that should be communicated to store or retrieve the data.

- DataNode “the slave node”: DataNode stores two files: first one contains the data and the second is checksum file to identify corrupted data blocks.

HDFS was designed to expect and detect the hardware failure, each DataNodes regularly sends a heartbeat signal to the NameNode. If not happened it marks this DataNode as out of service and re-replicates all data to other active DataNodes. [6]

2.3.4 MapReduce Framework:

MapReduce it is achieved the “Data Locality” concept which means moving the computation to data instead of moving data to the computation and then aggregates the results to get the final output. Also it is a programming model for processing huge and distributed data.

2.3.5 MapReduce Consists of Three Main Steps:

- Map: convey the input data in to set of <key, value>
- Shuffle and Sort: In this step, Hadoop shuffles and reduce <key, value> pairs and sorts these pairs.
- Reduce: finally, applies the required functions, such as (count, average, etc.). And writes the final result to HDFS. [6]

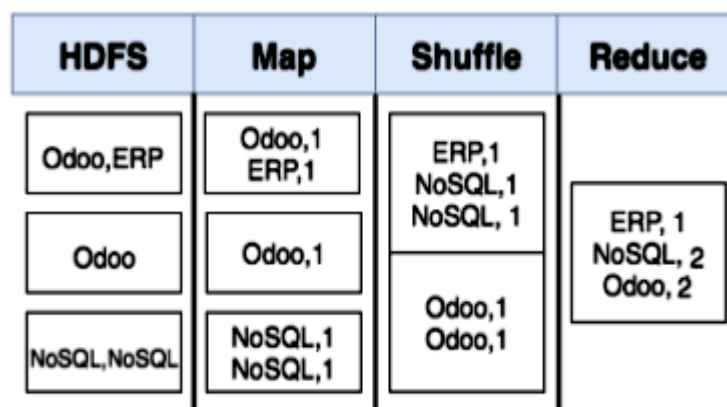


Figure 1.1: MapReduce Example

2.4 NoSQL Database Storage:

It is normal for any web application to serve millions of users simultaneously. These users generate huge amount of data, so data storage system needs to manage his efficiently. Many companies such as Amazon and Google have limitation of relation databases and developed own data storage system to be suitable for the new requirements, these systems become known as NoSQL “Not Only SQL”

2.4.1 NoSQL characteristics:

- Support parallel processing for data that have huge volumes (distributed systems).
- NoSQL data storage systems do not need predefined schema and no tables to store data.
- Each data item can have different attributes.
- NoSQL systems has different query language, so do not necessarily support SQL statement. [6]

2.4.2 A Comparison between SQL and NoSQL databases:

Table (2.1) a Comparison between SQL and NoSQL

NoSQL databases	SQL databases
Are optimized for key-value stores	SQL databases are not
Non-relational databases and distributed	It is relational databases and not distributed
NoSQL databases are document-oriented and have a relationship with applications written in object-oriented programming languages like Java, PHP, and Python through APIs.	SQL is not document-oriented.

Allow developers to execute queries without having to understand SQL or the basic architecture of their database system.	Developers should be learn SQL and understand the underlying architecture of their database system.
Some of the NOSQL databases providers are handles hardware failures.	SQL does not handle hardware failures.
Different query languages: each of these systems has own language to access stored data.	Structured Query Language: SQL systems support standard language to access data.
Horizontal scalability.	Vertical scalability.
BASE properties: NoSQL users prefer read outdated information to obtain preferable performance. Such as E-commerce systems.	ACID properties: SQL systems prefer read the most recent value, which leads to Lower performance such as Banking systems.
Evolving: NoSQL systems are still in the growing phase, and continuously there are new releases.	Stable: SQL databases are stable because it has been in use for years.

2.4.3 Common Type of NoSQL Database:

- Key-value model: which stores data in less complex way that consists of indexed keys and values. Examples: Cassandra, LevelDB, and Riak.
- Column store: which allow high performance and excellent scalability by storing data tables as columns rather than rows. Examples: Hbase, BigTable and HyperTable.
- Document database: using the key-value concept, each document has its own data and unique key, which is used to retrieve it. Examples: MongoDB, and CouchDB.
- Graph database: this method is perfect when you have complex data represented as a graph. Examples: Polyglot, Neo4J. [7]

2.4.4 Popular NoSQL Databases:

- MongoDB: popular open-source and free NoSQL system, It was developed using C++, it is efficient database, have a high performance, persistence, ad hoc queries and indexing. MongoDB is document-oriented database the documents are stored in BSON (Binary JSON) format. Disadvantages is that indexing takes up lot of ram.
- Apache's CouchDB: was developed by Apache software for the web, it is stores the documents in JSON data exchange format.
- Hbase: also Apache project but developed as a part of Hadoop, that main features are open-source and the type is column family.
- Couchbase is a NoSQL document database is the fastest overall for read, write and delete operations for interactive web applications.
- Apache's Cassandra DB: Cassandra is a distributed database for handling huge amounts of structured data. The important disadvantage of Cassandra is that reads are so slowly compared to writes. This applied in Instagram, Comcast and Apple.
- Riak: an open-source and key-value store database. for excellent performance it used automatic data distribution.[7][8]

2.5 Literature review:

In this section, some of previous related research will be addressed and compared to come out with research gab.

2.5.1 Solving and Processing Big Data Problem using Hadoop

In [9] the author's focus on the problems related to big data (data more than one TB) and network, these problems like: Network failure, Disk failure and high storage cost. To prevent these problems apply Hadoop because it provides the solutions, one of this are the data redundancy feature. Means can recover the data from the replicated. The second is map and reduce. Running of map reduce code in apache Hadoop used to process the data. Mappers are programs allow

small amount of data runs in parallel and the reducer will present the final result. Data have to be stored and retrieved in low cost, effective way and process it efficiently.

Enterprise resource planning depends on relational database which has problems including the difficulty to build Data Warehouse with a low cost and time as well as processing of huge data. The scientist in [10] introduced two main solutions:

Solution 1: The data is archived in Hadoop Distributed File System HDFS with map reduce feature which uses Sqoop for data migration. Hive is used as SQL Language to retrieve the data to perform analysis and make decisions.

Solution 2: The ORM method is modified in order for Odoo system storage to be able to contact with Cassandra as NoSQL database.

This solutions achieved high performance and availability for Odoo system even in the case of huge data. It also offered secondary storage with relatively low cost for archiving and making decisions.

2.5.2 Performance Evaluation and Data Migration to Hadoop.

This project [11] try to solve migrating process for data Exist in RDBMS (MySQL database) to DB2 database. Current system take huge resources and time to migrate pita bytes of data. The suggested project migrate data from MySQL to Hadoop distributed file system using tool called Sqoop, then Hadoop HIVE is used to retrieve data. The result was positive because data replication can recover the data if failure. Also, Hadoop used MapReduce in processing so migration process become faster. The NameNode and DataNodes have built in web servers that are makes it easy to check current status of the cluster, low coast and more flexible to add extra nodes

The researchers in [12] apply tests aim to estimating the performance considering the random writing and random reading of rows, and how they make a difference by increasing the number of servers. Use Hbase for this evaluation because it have low cost implementation, the system can scalability by added

nodes and handle a huge volumes of data. In case of failure, it can be restored automatically by using replication feature, the model implemented is write-once-read-many.

The results of this evaluation shows that the random reading on single column family increased by augmenting the number of nodes, while random writes remained the same in the case of a single node, or four nodes. In multiple column family, the performance of both random reads and write are not affected by increasing the number of servers and column families.

2.5.3 A performance Comparison of SQL and NoSQL Databases.

The inquisitive in this research [6] discuss the latency problem in Odoo ERP system caused by two Odoo modules (mail messages, attachments). To solve the problem of Odoo latency using NoSQL system, Hbase-Hadoop is used as a new NoSQL data storage. The data is moved into Hadoop Distributed File System (HDFS) Using Sqoop to Hbase databases. After that, ORM layer is updated. Phoenix Hadoop is used to perform SQL queries over Hbase to retrieve data. The results of experiment and evaluation tests that application can still use PostgreSQL for long time as better choice than proposed Hadoop stack. To get the benefits of using developed solution, large data volumes are required (more than one TB).

The researchers in [13] compare key-value store implementations on NoSQL and SQL databases. Compare time of read, write and delete for some NoSQL databases. Find that not all NoSQL databases perform better than the SQL database. In NoSQL databases there is a wide variation in performance based on the type of operation. For example Couchbase and MongoDB are faster in all operations. RavenDB and CouchDB do not result well in the read, write and delete operations. Casandra is slow on read operations.

In [14] the authors tried to study and compare the different kinds of NoSQL systems under specific criteria. The study focused on Hbase and

Cassandra because they have the same family and framework (Hadoop). Moreover, it took MySQL (relational/SQL database) to see the benefits of using NoSQL. The criteria included features such as persistence, replication, high availability, transactions rack-locality awareness, implementation language, influences /sponsors and license type. Then, the performance is measured for write and read considering the number of nodes and the records. Comparison results show that for over 7000 read or write operations per second in MySQL becoming unresponsive and the latency time is too great. The write performance of Hbase is greatly improved. Furthermore, in a read intensive environment, MySQL is offering better results.

2.6 Summary:

This chapter described concepts such as: Hadoop, ERP systems, NoSQL Database in details and summarized the current literature.

Chapter Three

Research Methodology

CHAPTER THREE

Research Methodology

3.1 Introduction

This chapter introduces the methodology which research follows it, and explains the solution architecture to solve Odoo performance limitations.

3.2 Research Methodology:

The research methodology that will be employed in the research work is in short: First of all, create millions of data records in the Odoo system. Then, identify the suitable NoSQL database type to store imported data from Odoo. Moreover, build an Odoo module that is able to read and write data from both PostgreSQL Database and suggested database. Finally, evaluate the performance of the Odoo system after applying the proposed solution in terms of processing time.

3.3 System Architecture:

In Figure 4.1, we explain an overall view of the new system. Odoo executes all requests using the normal Odoo ORM layer and PostgreSQL. Only requests for Hbase-based models ("employee") are executed by the 'HBaseORM' module. 'HBaseORM' sends all CRUD operations to the Hbase cluster directly through the Hbase Thrift API. Also, it executes search requests using the Phoenix layer via the Query Server. Every information stored in the Hbase database is managed by the Hadoop distributed file system.

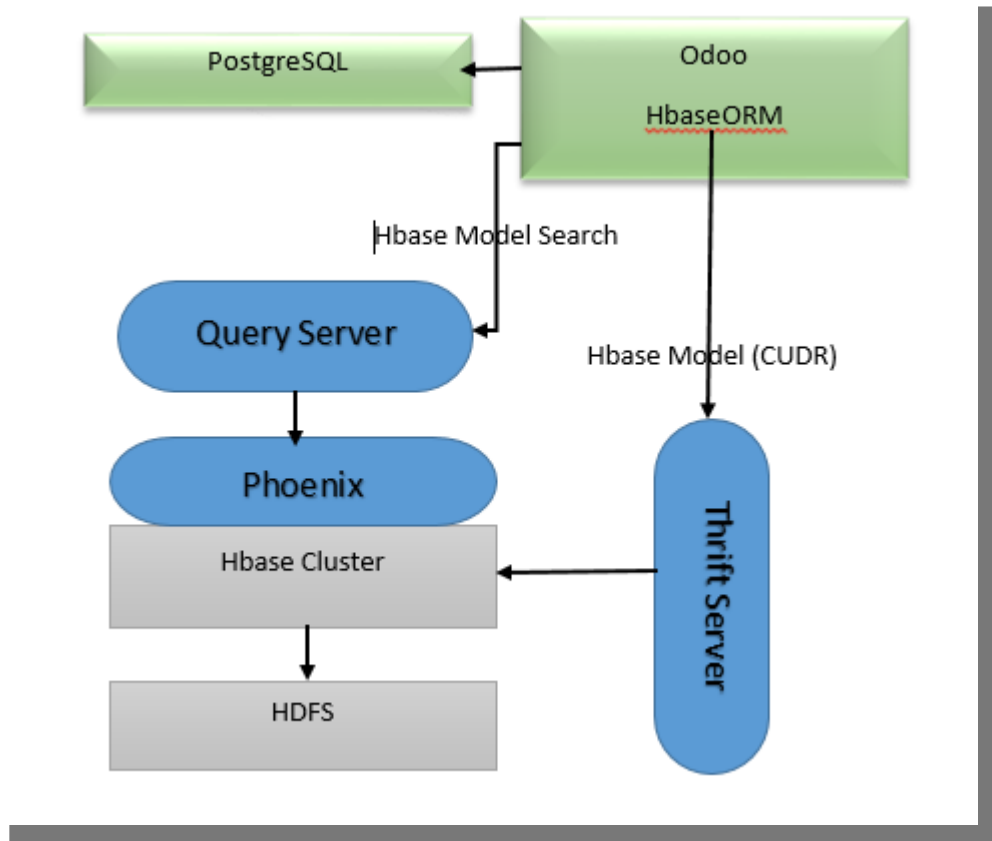


Figure 3.1: System Architecture

3.3.1 Functional Requirement:

Implement all original Odoo functionalities, such as: Add, update, read and search for specific table to support the new data storage.

3.3.2 Non-functional Requirement:

The main non-functional requirement is to minimizing the process time and Archive the data to use for mining in future.

3.3.3 Hbase fetchers:

Hbase is can be partitioned more efficiently because it is one of the column family data models, also these databases are more suitable for huge datasets than document stores. Columns and rows can be added very flexibly at runtime but column families have to be predefined, which leads to less flexibility than key value stores and document stores offer. [15]

Hbase have low cost implementation. Handle a large volumes of data. In case of failure, it can be restored automatically because Hbase use replication for offer full consistency. Finally, is optimal for dealing with data that has the access pattern write-once/read-many. That is the aim of the system. On the other hand Cassandra is slow on read operations but is reasonably good for write. It can be used to handle huge, complex and expressive data structures. [12][15]

3.3.4 Zookeeper:

Zookeeper is a high performance coordination service for distributed applications (like Hbase). It offer common services like naming, configuration management and synchronization, It is configured in hbase-env.sh file and Tell Hbase whether it should be manage its own instance of Zookeeper or not. Export HBASE_MANAGES_ZK=true or false it will start/stop the Zookeeper servers as a part of the regular start/stop scripts. In this project we used external Zookeeper to manage the Hbase.

3.3.5 Phoenix:

Use an application from Hadoop ecosystem, such as Hive and Phoenix that performs SQL queries over Hbase to retrieve data.

Either than Hbase is the NoSQL database and working well , there is one limitation of the Hbase that it is not very user friendly for SQL developer, to overcome this limitation they come up with the SQL layer above Hbase known as Apache Phoenix.

3.4 System Implementation:

We used Java version 1.8.0, Hadoop version 2.7.3, Hbase version 1.2.6, Sqoop version 1.4.7 and phoenix version 4.14.0 which are all compatible with each other.

The processes are running in data which in Hadoop cluster not data in an RDBMS. Because of the storage and parallelism capabilities of HDFS, Hadoop cluster can run optimally on many different readily available hardware systems, and Scales well ,Provides cheaper storage options ,Improves the performance of overall big data environment, can store more data, and store more types of data (both structured and unstructured).

3.4.1 Hadoop installation and Configuration:

Select Hadoop ecosystem, which is open source applications offered by Apache software Foundation. To set up your Hadoop cluster follow the information that is available with the Hadoop solution you are using. After installed Java and Hadoop in two machines/nodes open the file in terminal **hdfs-site.xml** to ensure high availability of data, Hadoop replicates the data, the configuration “dfs.replication” factor is to specify how many replications are required. This factor should be equal to number of nodes in system. Also, configure the path where DataNode and NameNode directory.


```
GNU nano 2.2.6 File: /usr/local/hadoop/etc/hadoop/hdfs-site.xml
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
```

Figure 3.2: Configure Replication Factor in hdfs-site.xml File

3.4.2 Hbase Configuration

Install Hbase in two machines/nodes it used to import data in.

3.4.3 Data collection:

Data was generated by two ways: first of one is python wizard generate data from existing Employee records

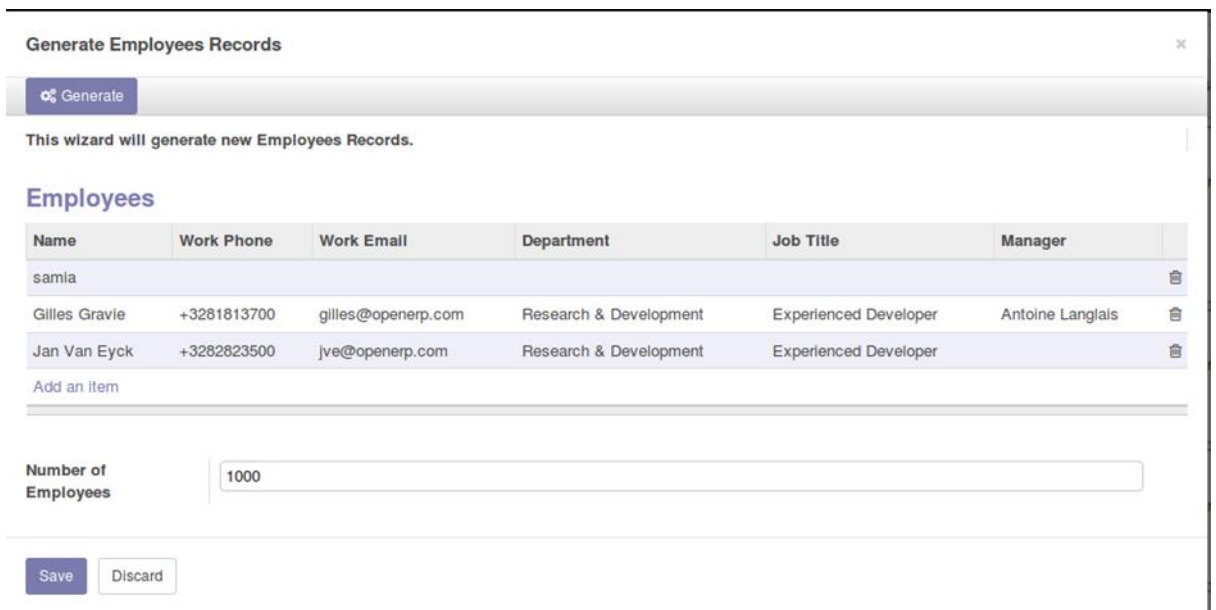


Figure 3.3: Python Wizard Used to Generate Data

The second one by using python method called “generate_series“, then created millions of data records in Odoo system.

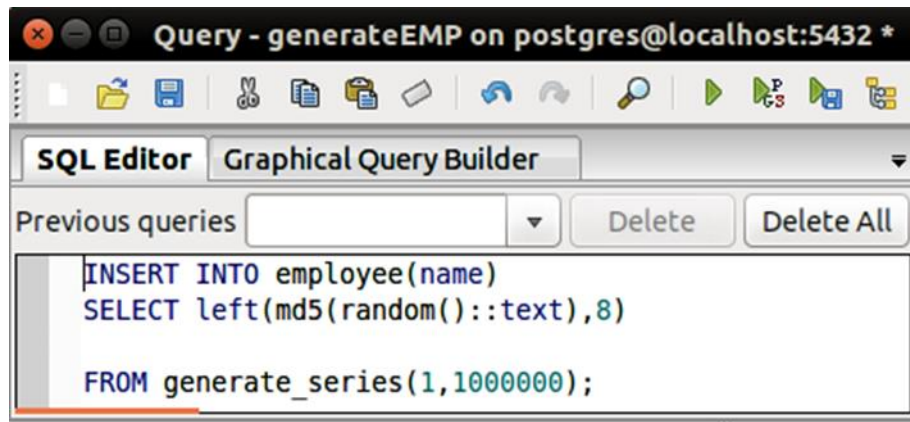


Figure 3.4: Python Script Used to Generate Data

3.4.4 Data Analysis:

Identify the suitable NoSQL database type to store imported data from Odoo, by Using Sqoop to transfer all the data from PostgreSQL to Hbase data databases in Hadoop ecosystem.

3.4.5 Import Data from PostgreSQL to Hbase Databases in Hadoop Ecosystem:

Now that the Hadoop cluster is set up, so, we need to integrate the Hadoop cluster with the existing Odoo / ERP, which uses an RDBMS as its data storage. Sqoop comes now. It is an import/export tool that allows you to move Big Data between an SQL DB and Hadoop cluster using a JDBC driver. Sqoop is faster to deploy and faster to make updates to in a production environment.

Before using Sqoop create “Employee” table in hbase to receive data which coming from PostgreSQL.

```
hbase(main):050:0* create "Employee" , 'EmployeeName'
0 row(s) in 6.9290 seconds

=> Hbase::Table - Employee
hbase(main):051:0> █
```

Figure 3.5: Create Employee Table in Hbase

After the table is created, importing the data to Hbase and specify which database should be accessed and columns want to retrieve.

```
ibrahim@ibrahim:~/usr/local/hbase-1.2.6/bin$ sqoop import --connect jdbc:postgresql://127.0.0.1/newdb --table hr_employee --username odoo --pass
word odoo --columns "id,name_related" --hbase-table EMPLOYEES --column-family EMPLOYEEENAME --hbase-row-key id -m 1 █
```

Figure 3.6: Importing Data from PostgreSQL to Hbase Using Sqoop

When this process finished, Data importing complete successfully.

```

18/07/20 03:13:36 INFO mapreduce.Job: Job job_1532028448015_0004 completed successfully
18/07/20 03:13:38 INFO mapreduce.Job: Counters: 30
  File System Counters
    FILE: Number of bytes read=0
    FILE: Number of bytes written=167034
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=87
    HDFS: Number of bytes written=0
    HDFS: Number of read operations=1
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=0
  Job Counters
    Launched map tasks=1
    Other local map tasks=1
    Total time spent by all maps in occupied slots (ms)=61771
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=61771
    Total vcore-milliseconds taken by all map tasks=61771
    Total megabyte-milliseconds taken by all map tasks=63253504
  Map-Reduce Framework
    Map input records=125032
    Map output records=125032
    Input split bytes=87
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=269
    CPU time spent (ms)=46720
    Physical memory (bytes) snapshot=164794368
    Virtual memory (bytes) snapshot=597295104
    Total committed heap usage (bytes)=80216064
  File Input Format Counters
    Bytes Read=0
  File Output Format Counters
    Bytes Written=0
18/07/20 03:13:38 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 138.3606 seconds (0 bytes/sec)
18/07/20 03:13:38 INFO mapreduce.ImportJobBase: Retrieved 125032 records.

```

Figure 3.7: Data Importing Successfully

To ensure all data transferred, do scan to "Employee" table in Hbase for all rows or scan "Employee" table within specific condition.

```

hbase(main):054:0> scan "Employee"
ROW COLUMN+CELL
1 column=EmployeeName:name_related, timestamp=1532045021670, value=Pieter Parker
10 column=EmployeeName:name_related, timestamp=1532045021670, value=Jimmy Kosikin
11 column=EmployeeName:name_related, timestamp=1532045021670, value=Fanke Jenssens
12 column=EmployeeName:name_related, timestamp=1532045021670, value=Ashley Presley
13 column=EmployeeName:name_related, timestamp=1532045021670, value=Hans Anders
14 column=EmployeeName:name_related, timestamp=1532045021670, value=Jan Van Eyck
15 column=EmployeeName:name_related, timestamp=1532045021670, value=Jean-Pierre Carnaud
16 column=EmployeeName:name_related, timestamp=1532045021670, value=John Doe
17 column=EmployeeName:name_related, timestamp=1532045021670, value=Jo\xC3xA3o Gomer
18 column=EmployeeName:name_related, timestamp=1532045021670, value=Juan Gomez
19 column=EmployeeName:name_related, timestamp=1532045021670, value=Luigi Rondi
2 column=EmployeeName:name_related, timestamp=1532045021670, value=Antoine Langlais4
20 column=EmployeeName:name_related, timestamp=1532045021670, value=\xD0\x9D\xD0\xB8\xD0\xBA\xD0\xBE\xD0\xBB\xD0\xB0\xD0\xB9 \xD0\x9F\xD0\xB5\xD1\x82\xD1\x80\xD0\xB0
21 column=EmployeeName:name_related, timestamp=1532045021670, value=\xE4\xBD\x90\xE8\x97\xA4\xE3\x81\x95\xE3\x81\x8F\xE3\x82\x89 (demo_ja_JP)
22 column=EmployeeName:name_related, timestamp=1532045021670, value=\xE8\xB6\x99\xE7\x94\x9F (demo_zh_CN)
23 column=EmployeeName:name_related, timestamp=1532045021670, value=Roger Scott
25 column=EmployeeName:name_related, timestamp=1532045021670, value=amia abo
3 column=EmployeeName:name_related, timestamp=1532045021670, value=John Smith
4 column=EmployeeName:name_related, timestamp=1532045021670, value=Michael Hawkins
5 column=EmployeeName:name_related, timestamp=1532045021670, value=Liam Nelson
6 column=EmployeeName:name_related, timestamp=1532045021670, value=David Samson
7 column=EmployeeName:name_related, timestamp=1532045021670, value=Gilles Gravie
8 column=EmployeeName:name_related, timestamp=1532045021670, value=Jack Macklin
9 column=EmployeeName:name_related, timestamp=1532045021670, value=Martin Lawrence
24 row(s) in 0.9620 seconds

```

Figure 3.8: Scan "Employee" Table

```

hbase(main):001:0> scan 'Employee', {COLUMNS => ['EmployeeName'], LIMIT => 10, STARTROW => '9'}
COLUMN+CELL
9      column=EmployeeName:name_related, timestamp=1532045574467, value=Martin Lawrence
90     column=EmployeeName:name_related, timestamp=1532045574467, value=Gilles Gravie 29
900    column=EmployeeName:name_related, timestamp=1532045574467, value=\xE4\xBD\x90\xE8\x97\xA4\xE3\x81\x95\xE3\x81\x8F\xE3\x82\x89 (demo_ja JP) 839
9000   column=EmployeeName:name_related, timestamp=1532045582072, value=Jo\xC3\xA3o Gomer 8939
9001   column=EmployeeName:name_related, timestamp=1532045582072, value=John Doe 8940
9002   column=EmployeeName:name_related, timestamp=1532045582072, value=John Smith 8941
9003   column=EmployeeName:name_related, timestamp=1532045582072, value=Juan Gomez 8942
9004   column=EmployeeName:name_related, timestamp=1532045582072, value=Liam Nelson 8943
9005   column=EmployeeName:name_related, timestamp=1532045582072, value=Luigi Rondi 8944
9006   column=EmployeeName:name_related, timestamp=1532045582072, value=Martin Lawrence 8945
10 row(s) in 0.4060 seconds
hbase(main):002:0>

```

Figure 3.9: Scan “Employee” Table with Specific Condition

Can see there are differences between Table in Hbase (Hadoop cluster) and same table in PostgreSQL in the storage format.

	id [PK] serial	address_id integer	create_dat timestamp	ssnid character	coach_id integer	message_l timestamp	color integer	marital character	identificat character	bank_acco integer	Job_id integer	parent_id integer	work_phor character	resource_l integer	country_id integer	depart integer
1	1	3	2018-02-13			0					1		+3281813764			3
2	2		2018-02-13			0					2		+3281813765			4
3	3		2018-02-13			0					4	2	+3281813766			4
4	4		2018-02-13			0					4	2	+3281813767			4
5	5		2018-02-13			0					4	2	+3281813768			4
6	6		2018-02-13			0					4	2	+3281813769			4
7	7		2018-02-13			0					4	2	+3281813770			4
8	8		2018-02-13			0					4	2	+3281813771			4
9	9		2018-02-13			0					3	1	+3281813772			5
10	10		2018-02-13			0					3	1	+3281813773			5
11	11		2018-02-13			0					6	1	+3281813774			2
12	12		2018-02-13			0					5	1	+3281813775			1
13	13		2018-02-13			0					4		+32828235616			4
14	14		2018-02-13			0					4		+32828235617			4
15	15		2018-02-13			0					3		+32828235618			5
16	16		2018-02-13			0					6		+32828235619			2
17	17		2018-02-13			0					4		+32828235620			4
18	18		2018-02-13			0					3		+32828235621			5
19	19		2018-02-13			0					6		+32828235622			2
20	20		2018-02-13			0					3		+32828235623			5
21	21		2018-02-13			0					3		+32828235624			5
22	22		2018-02-13			0					4		+32828235625			4
23	23		2018-02-13			0							+32828235626			

Figure 3.10: Table in PostgreSQL

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	ibrahim	supergroup	0 B	7/20/2018, 2:35:51 AM	0	0 B	Employee
drwxr-xr-x	ibrahim	supergroup	0 B	5/1/2018, 3:52:56 AM	0	0 B	HrPayslip

Figure 3.11: Table in Hbase (Hadoop cluster)

3.5 Summary:

This chapter introduced the methodology which research follow it, and listed the functional and non-functional requirements and explained the solution architecture to solve Odoo performance limitation.

Chapter Four

System Design and Implementation

CHAPTER FOURE

System Design and Implementation

4.1 Introduction:

This chapter introduce the system implementation which enabling Odoo to be contacted Hbase as NoSQL data storage instead of PostgreSQL. In addition, how a new system components interact with each other's. Finally, show the result of the research problem.

4.2 Development:

Build Odoo module that is able to reading and writing data from both PostgreSQL Database and Hbase database.

4.2.1 Odoo ORM:

The ORM, short for Object-Relational Mapping is another technology in Odoo to save and retrieve the data from PostgreSQL without the SQL Queries.

4.2.2 The benefits of ORM technology

- No need to deal with the SQL Queries to save and retrieve the data.
- Simple configuration.
- Easy to learn and use.
- Support the concurrency.
- Provide standardized API to support the business objects.
- Fast development of application.

4.2.3 Disadvantages of ORM:

Slow performance in case of large batch updates.

4.2.4 JDBC concept:

JDBC stands for Java Database Connectivity, is a set of Java API for accessing the relational databases from Java program. The Java API enables the programmers to execute the SQL statements against the JDBC compliant database.

JDBC allows the programmers to quickly develop small Java applications that interact with the databases. [16]

4.2.5 Create ORM Method (CRUD):

In Odoo each model has four main functions for CRUD operations (Create, Read, Update, and Delete).

Hbase is developed using Java language and provides various Client APIs to access Hbase from other programming languages like 'Python'. There are two APIs connector for Hbase: Thrift API and REST API. Thrift API is faster than Representational State Transfer (REST) API, so, we choose Thrift API as the connector between Odoo and Hbase. And have two Python libraries communicate with Hbase: Happybase, Starbase. Only Happybase library support Thrift API. So, we choose Happybase library to send the requests to Hbase through Thrift API. Happybase supports all the needs to communicate with Hbase.

To access the Hbase through the thrift, need to open the thrift service on Hbase by running this command: `Hbase thrift -p 9090 start`

Update the ORM layer emphasized that all data in PostgreSQL and data in Hadoop cluster are consistent.

4.2.5.1 Create:

Create function is called by Odoo when a user need to create a new record .This customization in function store the record in both PostgreSQL and Hbase,

```
@api.model
def create(self, values):
    connection=happybase.Connection(host='localhost',port=9090,autoconnect=False,compat='0.96',transport='buffered')
    connection.open()
    table = connection.table('EMPLOYEES')
    father = super( HrEmployee, self).create(values)
    empid=str(father.id)
    name = values.get('name')
    table.put(empid, {b'EMPLOYEE_NAME:name_related':name})
    return father
```

Figure 4.1: Optimization in Create Function

4.2.5.2 Update:

Update function is called by Odoo when a user tries to update an existed record .This customization in function update the record in both PostgreSQL and Hbase.

```
@api.multi
def write(self, vals):
    connection=happybase.Connection(host='localhost',port=9090,autoconnect=False,compat='0.96',transport='buffered')
    connection.open()
    table = connection.table('EMPLOYEES')
    father = super(HrEmployee, self).write(vals)
    empid = str (self.id)
    name = str (self.name)
    table.put(empid, {b'EMPLOYEE_NAME:name_related':name})
    return father
```

Figure 4.2: Optimization in Update Function

4.2.5.3 Delete:

Delete function is called by Odoo when a user wants to delete an existed record of a model. This customization in function delete the record in both PostgreSQL and Hbase.

```
@api.multi
def unlink(self):
    connection=happybase.Connection(host='localhost',port=9090,autoconnect=False,compat='0.96',transport='buffered')
    connection.open()
    table = connection.table('EMPLOYEES')
    r = str(self.id)
    row = table.delete(r)
    resources = self.mapped('resource_id')
    super(HrEmployee, self).unlink()
    return resources.unlink()
```

Figure 4.3: Optimization in Delete Function

4.3 Start system servers and web interfaces:

```
29912 Jps
hduser@master:/usr/local/hadoop/sbin$ ./start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
18/09/06 19:40:00 WARN util.NativeCodeLoader: Unable to load native-hadoop library
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namen
master: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datan
slave1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datan
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hd
18/09/06 19:40:44 WARN util.NativeCodeLoader: Unable to load native-hadoop library
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource
slave1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-node
master: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-node
hduser@master:/usr/local/hadoop/sbin$ jps
10641 HMaster
1522 ResourceManager
10772 HRegionServer
1046 DataNode
15964 Jps
1340 SecondaryNameNode
```

Figure 4.4: Start hdfs and MapReduce (YARN)

```

hduser@master:/usr/local/zookeeper-3.4.10/bin$ sudo ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper-3.4.10/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
hduser@master:/usr/local/zookeeper-3.4.10/bin$ █

```

Figure 4.5: Run Zookeeper Server

```

hduser@master:/usr/local/hbase-1.2.6/bin$ ./start-hbase.sh
starting master, logging to /usr/local/hbase-1.2.6/bin/./logs/hbase-hduser-master-master.out
/usr/local/hbase-1.2.6/bin/hbase-daemon.sh: line 225: /var/hadoop/pids/hbase-hduser-master.pid: Permission
denied
Java HotSpot(TM) Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
Java HotSpot(TM) Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
slave1: bash: line 0: cd: /usr/local/hbase-1.2.6/bin/./: No such file or directory
slave1: bash: /usr/local/hbase-1.2.6/bin/hbase-daemon.sh: No such file or directory
master: starting regionserver, logging to /usr/local/hbase-1.2.6/bin/./logs/hbase-hduser-regionserver-mas
ter.out
master: /usr/local/hbase-1.2.6/bin/hbase-daemon.sh: line 225: /var/hadoop/pids/hbase-hduser-regionserver.p
id: Permission denied
master: Java HotSpot(TM) Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
master: Java HotSpot(TM) Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
hduser@master:/usr/local/hbase-1.2.6/bin$ █

```

Figure 4.6: Starting Hbase Server

```

ibrahim@ibrahim:/usr/local/hbase-1.2.6/bin$ ./hbase shell
2018-07-18 00:30:29,390 WARN [main] util.NativeCodeLoader: Unable to load nativ
e-hadoop library for your platform... using builtin-java classes where applicabl
e
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase-1.2.6/lib/slf4j-log4j12-1.7.5
.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4
j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> █

```

Figure 4.7: Open Hbase Shell

```

hduser@master:/usr/local/hadoop/sbin$ jps
10641 HMaster
1522 ResourceManager
10772 HRegionServer
1046 DataNode
15964 Jps
1340 SecondaryNameNode
1677 NodeManager
895 NameNode
hduser@master:/usr/local/hadoop/sbin$ █

```

Figure 4.8: Run JPS in Master Machine


```

hduser@master:~$ ssh slave1
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.4.0-31-generic i686)

 * Documentation:  https://help.ubuntu.com/

Last login: Tue Sep  4 15:28:03 2018 from slave1
hduser@slave1:~$ jps
30530 Jps
29970 DataNode
30070 NodeManager
hduser@slave1:~$ █

```

Figure 4.9: Run JPS in Slave Machine

```

odoo@ibrahim:/home/ibrahim$ /opt/odoo/odoo-10.0/odoo-bin --addons-path="/opt/odoo/odoo-10.0/addons,/home/ibrahim/Desktop/addons"
2018-07-20 00:00:34,702 2407 INFO ? odoo: Odoo version 10.0
2018-07-20 00:00:34,702 2407 INFO ? odoo: addons paths: ['/opt/odoo/.local/share/Odoo/addons/10.0', u'/opt/odoo/odoo-10.0/addons', u'/home/ibrahim/Desktop/addons', '/opt/odoo/odoo-10.0/odoo/addons']
2018-07-20 00:00:34,702 2407 INFO ? odoo: database: default@default:default
2018-07-20 00:00:34,891 2407 INFO ? odoo.service.server: HTTP service (werkzeug) running on 0.0.0.0:8069
2018-07-20 00:00:54,311 2407 INFO ? odoo.addons.report.models.report: You need wkhtmltopdf to print a pdf version of the reports.
2018-07-20 00:00:57,037 2407 INFO ? odoo.http: HTTP Configuring static files
2018-07-20 00:00:57,095 2407 INFO newdb odoo.modules.loading: Loading 1 modules...
2018-07-20 00:00:57,119 2407 INFO newdb odoo.modules.loading: 1 modules loaded in 0.02s, 0 queries
2018-07-20 00:00:57,404 2407 INFO newdb odoo.modules.loading: Loading 31 modules...
2018-07-20 00:00:57,751 2407 INFO newdb odoo.modules.loading: 31 modules loaded in 0.35s, 0 queries
2018-07-20 00:00:58,204 2407 INFO newdb odoo.modules.loading: Modules loaded.
2018-07-20 00:00:58,231 2407 INFO newdb odoo.addons.base.ir.ir_http: Generating routing map
2018-07-20 00:00:58,550 2407 INFO ? odoo.addons.bus.models.bus: Bus.loop listen inbus on db postgres
2018-07-20 00:01:10,882 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:10] "POST /web/dataset/call_kw/hr.employee/write HTTP/1.1" 200 -
2018-07-20 00:01:10,974 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:10] "POST /web/dataset/call_kw/hr.employee/search_read HTTP/1.1" 200 -
2018-07-20 00:01:11,025 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:11] "POST /web/dataset/call_kw/hr.employee.category/read HTTP/1.1" 200 -
2018-07-20 00:01:11,026 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:11] "GET /web/image?model=hr.employee&id=25&field=image_medium&unique=20180720000110 HTTP/1.1" 200 -
2018-07-20 00:01:11,040 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:11] "POST /mail/read_followers HTTP/1.1" 200 -
2018-07-20 00:01:11,082 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:11] "POST /web/dataset/search_read HTTP/1.1" 200 -
2018-07-20 00:01:11,134 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:11] "POST /web/dataset/call_kw/hr.employee.category/read HTTP/1.1" 200 -
2018-07-20 00:01:11,662 2407 INFO newdb werkzeug: 127.0.0.1 - - [20/Jul/2018 00:01:11] "POST /web/menu/load_needaction HTTP/1.1" 200 -

```

Figure 4.10: Start Odoo Server

```

ibrahim@ibrahim:/usr/local/phoenix$ sqlline.py localhost
Setting property: [incremental, false]
Setting property: [isolation, TRANSACTION_READ_COMMITTED]
Issuing: !connect jdbc:phoenix:localhost none none org.apache.phoenix.jdbc.PhoenixDriver
Connecting to jdbc:phoenix:localhost
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/phoenix/phoenix-4.14.0-HBase-1.2-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
Java HotSpot(TM) Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so which might have disabled stack guard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
18/08/09 02:03:56 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Connected to: Phoenix (version 4.14)
Driver: PhoenixEmbeddedDriver (version 4.14)
Autocommit status: true
Transaction isolation: TRANSACTION_READ_COMMITTED
Building list of tables and columns for tab-completion (set fastconnect to true to skip)...
138/138 (100%) Done
Done
sqlline version 1.2.0
0: jdbc:phoenix:localhost>

```

Figure 4.11: Start Phoenix


4.4 Hadoop control panel:

Explain information about DataNodes like Nodes Name and the capacity of each one.

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
master:50010 (192.168.43.149:50010)	2	In Service	18.21 GB	28 KB	16.12 GB	2.08 GB	0	28 KB (0%)	0	2.7.3
slave1:50010 (192.168.43.238:50010)	2	In Service	7.26 GB	28 KB	5.44 GB	1.82 GB	0	28 KB (0%)	0	2.7.3

Figure 4.12: Hadoop Data Distribution (two nodes)



Nodes of the cluster

- Cluster
- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW_SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler
- Tools

Cluster Metrics													
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	
0	0	0	0	0	0 B	8 GB	0 B	0	8	0	1	0	

Scheduler Metrics			
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:1>

Show 20 entries											Search:
Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail		
/default-rack		RUNNING	master:38263	master:8042	Thu Sep 06 22:02:58 +0300 2018		0	0 B	8 GB		

Showing 1 to 1 of 1 entries

Figure 4.13: Hadoop Cluster

4.5 Fetch Hbase Table Data in Apache Phoenix:

After create table in Hbase and import the data from PostgreSQL , then create view in phoenix with the same name and columns of Hbase table ,Phoenix automatically fetch and load the data from Hbase table .

```
0: jdbc:phoenix:localhost>  
0: jdbc:phoenix:localhost>  
0: jdbc:phoenix:localhost> CREATE view EMPLOYEES (id VARCHAR PRIMARY KEY , "EMPLOYEEENAME"."name_related" VARCHAR);
```

Figure 4.14: Create View in Phoenix to Fetch Hbase Table Data

To show all phoenix tables and views just print “!Table” in terminal.

```
sqlline version 1.2.0  
0: jdbc:phoenix:localhost> !tables
```

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_NAME	SELF_REFERENCING_COL_NAME	REF_GENERATION	INDEX_S
	SYSTEM	CATALOG	SYSTEM TABLE					
	SYSTEM	FUNCTION	SYSTEM TABLE					
	SYSTEM	LOG	SYSTEM TABLE					
	SYSTEM	SEQUENCE	SYSTEM TABLE					
	SYSTEM	STATS	SYSTEM TABLE					
		US_POPULATION	TABLE					
		EMPLOYEES	VIEW					

```
0: jdbc:phoenix:localhost>
```

Figure 4.15: Show Phoenix Tables

Now can manipulate the data by using normal SQL language and its features.

```
0: jdbc:phoenix:localhost> SELECT * FROM EMPLOYEES;
```

ID	name_related
1	Pieter Parker
10	Jimmy Kosikin
11	Famke Jenssens
12	Ashley Presley
13	Hans Anders
14	Jan Van Eyck
15	Jean-Pierre Carnaud
16	John Doe
17	João Gomer
18	Juan Gomez
19	Luigi Rondi
2	Antoine Langlais4
20	Николай Петра
21	佐藤さくら (demo_ja_JP)
22	趙生 (demo_zh_CN)
23	Roger Scott
25	amia abo
3	John Smith

Figure 4.16: Retrieve Hbase Table Data in Apache Phoenix

4.6 Hadoop Cluster Implementation

The Hadoop cluster that proposed to make the evaluation consists of two nodes: One Master node and one Slave node (virtual machine).

Table (4.1) Cluster Nodes Specification

NAME	OS	CPU	RAM
Master	Ubuntu 14.04 LTS 32-bit	Intel core i5	18 GB
Slave1	Ubuntu 14.04 LTS 32-bit	Intel core i5	7 GB

Table (4.2) PostgreSQL Database Specification

NAME	OS	CPU	RAM
Master	Ubuntu 14.04 LTS 32-bit	Intel core i5	18 GB

Table (4.3) Sample Data Record Counts:

Table Name	Record No.
Employee	1000.000

4.7 Consistency Test Evaluation:

Consistent testing aims to check if the changes to Odoo return correct results on the new system:

1. Change employee name from 'Pieter Parker John' to 'Pieter Parker' and save, then run the query in PostgreSQL and phoenix, should be return the same data.

```
SELECT id, name_related
```

```
FROM hr_employee
```

```
WHERE name_related = 'Pieter Parker';
```

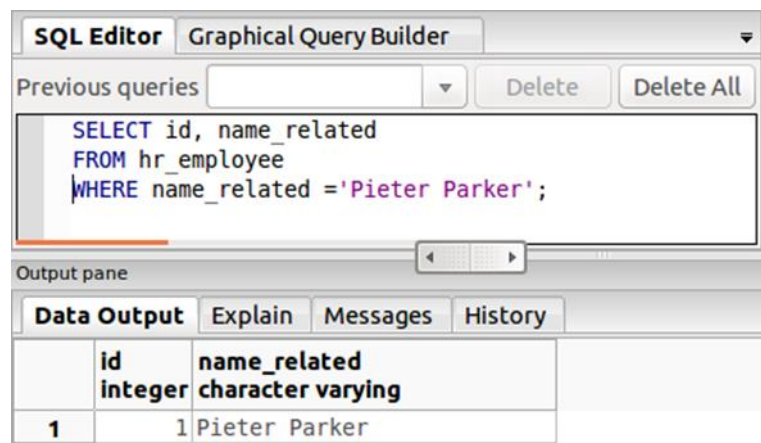


Figure 4.17: The Result of Query in PostgreSQL

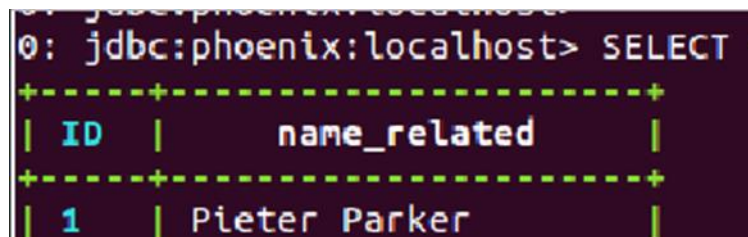


Figure 4.18: The Result of Query in Phoenix

2. delete specific employee and run the query in PostgreSQL and phoenix ,number of records should be same:

```
SELECT COUNT (id)
```

```
FROM hr_employee
```

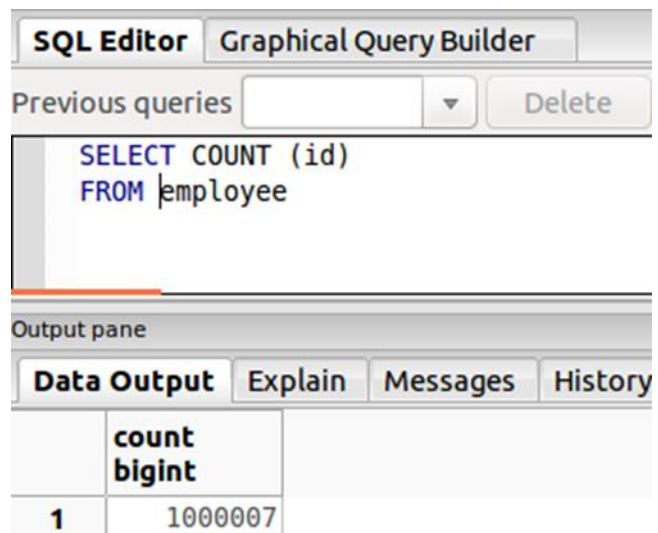


Figure 4.19: Query in PostgreSQL Return Number of Records

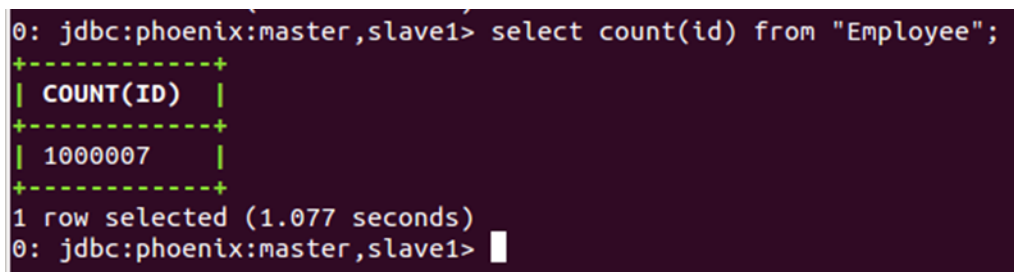


Figure 4.20: Query in Phoenix Return Number of Records

4.8 Database Systems Evaluation:

1. In this section, we evaluate the performance of PostgreSQL and Hadoop ecosystem to return names of the employees that has the word 'Pieter':

```
SELECT name_related  
  
FROM hr_employee  
  
WHERE name ILIKE '%Pieter%';
```

- Time to return Result from PostgreSQL:

Return about 125.000 row in 37 seconds



Unix	Ln 1, Col 18, Ch 18		125033 row	37345 ms
------	---------------------	--	------------	----------

Figure 4.21: Time to Return Specific Rows in Table from PostgreSQL

- Time to return specific result from phoenix:

125,000 row in 30 s

- Return all the data from employee table

```
SELECT *
FROM employee;
```

- Result from PostgreSQL:

Return about 1000.000 row in 56 seconds

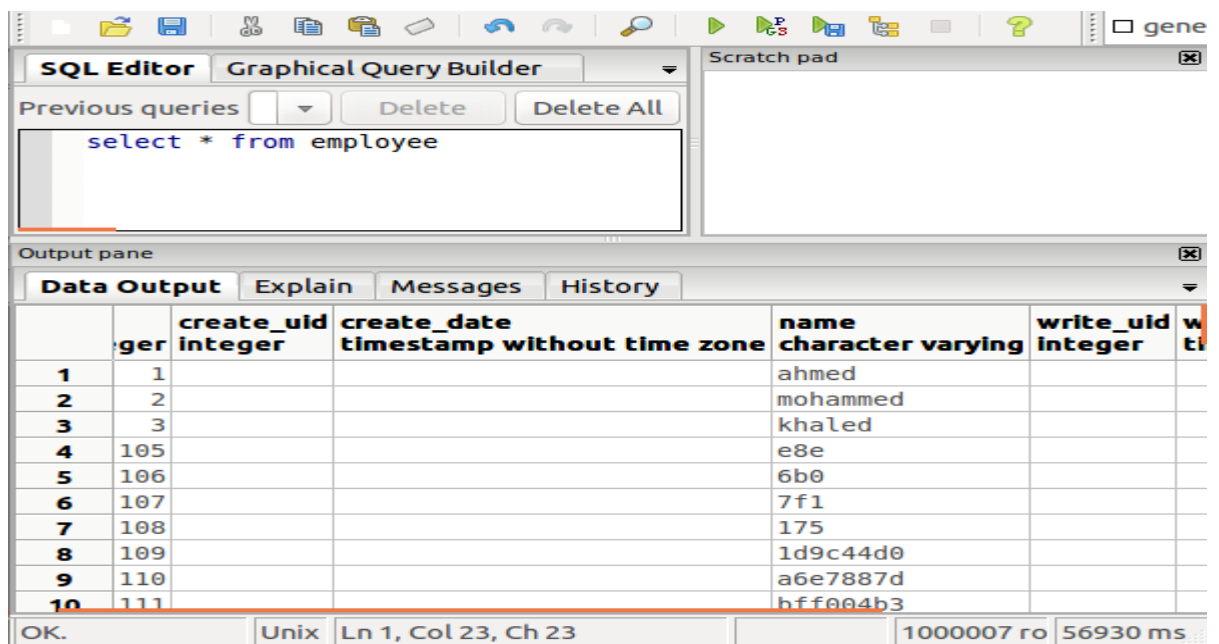


Figure 4.22: Time to Return All Rows in Table from PostgreSQL

- Time to Return All Result from phoenix

1000,000 row in 150 s

4.9 Summary:

This chapter discussed the system implementation in details and introduced the result of the research problem which is the delay and latency in retrieving information due to huge data in Odoo system.

Chapter Five

Conclusion and Recommendations

CHAPTER FIVE

Conclusion and Recommendations

5.1 Conclusions and Lessons Learned:

NoSQL systems are promoted as more performed systems than SQL data storage systems. Therefore, we chose to solve the problem of Odoo latency using NoSQL system.

Hbase-Hadoop was selected as a new NoSQL data storage. To evaluate the new system, compared its performance with the performance of the original Odoo system.

- The system able to scalability (add nodes) without stop.
- The NameNode and DataNodes have built in web servers that makes it easy to check current status of the cluster.

5.2 Recommendations:

- The results of this experiment explain that Hadoop ecosystem can be used for generating reports better than PostgreSQL.
- To get the benefits of using developed solution, large data volumes are required.
- Can apply data mining and PI in archived data in system.

Chapter Six

References

6.1 References:

- [1] (www.sodexis.com), W. (n.d). What is Odoo / OpenERP? | ERP CRM Software Longwood Orlando Florida. [online] Sodexis.com. Available at: <http://www.sodexis.com/services/what-is-odoo-longwood-orlando-florida.html> [Accessed 25 Feb. 2018].
- [2] Verma, J. and Agrawal, S. (2016). Big Data Analytics: Challenges and Applications for Text, Audio, Video, and Social Media Data. International Journal on Soft Computing, Artificial Intelligence and Applications, 5(1), pp.41-51.
- [3] Singh, S., Singh, P. and Garg, R. (2015). Big Data: Technologies, Trends and applications.
- [4] En.wikipedia.org. (2018). Big data. [online] Available at: https://en.wikipedia.org/wiki/Big_data [Accessed 8 Jul. 2018].
- [5] NetSuite.com. (n.d.). Do you know what ERP is? Learn how ERP can help your business with this informative article. [online] Available at: <http://www.netsuite.com/portal/resource/articles/erp/what-is-erp.shtml> [Accessed 17 Feb. 2018].
- [6] Enaya, M. (2016). An Experimental Performance Comparison of NoSQL and RDBMS Data Storage Systems in the ERP System Odoo.
- [7] Anon, (2017). [online] Available at: <https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/> [Accessed 17 Apr. 2018].
- [8] Nayak, A., Poriya, A. and Poojary, D. (2013). Type of NOSQL Databases and its Comparison with Relational Databases.
- [9] Chaturvedi, S., Lowden, F. and Bhirud, N. (2015). Solving Big Data Problem using Hadoop File System (HDFS).
- [10] Khalifa, E., Ahmed, S., Ismeil, O. and Balla, A. (2017). Storing

- [11] Tiyyagura, N., Rallabandi, M. and Nalluri, R. (2016). Data Migration from RDBMS to Hadoop.
- [12] Carstoiu, D., Cernian, A. and Olteanu, A. (2010). Hadoop Hbase-0.20.2 Performance Evaluation. IEEE.
- [13] Li, Y. and Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. IEEE.
- [14] Tudorica, B. and Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. IEEE.
- [15] Hecht, R. and Jablonski, S. (2011). NoSQL Evaluation a Use Case Oriented Survey.
- [16] Roseindia.net. (n.d.). JDBC vs ORM. [online] Available at: <https://www.roseindia.net/jpa/jdbc-vs-orm.shtml> [Accessed 16 Jul. 2018].
- [17] Uye, C. (2015). Integrate Hadoop with an existing RDBMS.
- Analytics, B. (2015). Integrate Hadoop with an existing RDBMS. [online] Ibm.com. Available at: <https://www.ibm.com/developerworks/library/ba-hadoop-rdbms/index.html#ibm-pagetitle-h1> [Accessed 11 Jan. 2018].