# AUTOMATIC MIGRATION OF DATA TO NOSQL DATABASES USING SERVICE ORIENTED ARCHITECTURE

**Rohan Koshy**

Roll. 213CS3184

*under the supervision of*

**Prof. Pabitra Mohan Khilar**


**Department of Computer Science and Engineering**

**National Institute of Technology Rourkela**

**Rourkela – 769008, India**

# Automatic Migration of data to NoSQL databases using Service Oriented Architecture

*Dissertation submitted in*

*MAY 2015*

*to the department of*

***Computer Science and Engineering***

*of*

***National Institute of Technology Rourkela***

*in partial fulfillment of the requirements*

*for the degree of*

***Master of Technology***

*by*

***Rohan Koshy***

*(Roll. 213CS3184)*

*under the supervision of*

***Prof. Pabitra Mohan Khilar***


**Department of Computer Science and Engineering**

**National Institute of Technology Rourkela**

**Rourkela – 769 008, India**

Computer Science and Engineering

**National Institute of Technology Rourkela**

Rourkela-769 008, India. www.nitrkl.ac.in

May 30, 2015

# Certificate

This is to certify that the work in the thesis entitled *AUTOMATIC MIGRATION OF DATA TO NOSQL DATABASES USING SERVICE ORIENTED ARCHITECTURE* by *Rohan Koshy*, having roll number 213CS3184, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering Department*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

**Dr. Pabitra Mohan Khilar**
Assistant Professor
Department of CSE
NIT, Rourkela

# Acknowledgment

First of all, I would like to express my deep sense of respect and gratitude towards my supervisor Prof. Pabitra Mohan Khilar, who has been the guiding force behind this work. I want to thank him for introducing me to the field of NoSQL databases and giving me the opportunity to work under him. His undivided faith in this topic and ability to bring out the best of analytical and practical skills in people has been invaluable in tough periods. Without his invaluable advice and assistance it would not have been possible for me to complete this thesis. I am greatly indebted to him for his constant encouragement and invaluable advice in every aspect of my academic life. I consider it my good fortune to have got an opportunity to work with such a wonderful person.

I thank our H.O.D. Prof. Santanu Kumar Rath and Prof. Durga Prasad Mohapatra for their constant support in my thesis work. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would also like to thank all faculty members, PhD scholars, my seniors and juniors and all colleagues to provide me their regular suggestions and encouragements during the whole work.

At last but not the least I am in debt to my family to support me regularly during my hard times.

I wish to thank all faculty members and secretarial staff of the CSE Department for their sympathetic cooperation.

***Rohan Koshy***

# Abstract

For the past few years there has been an exponential rise in the use of databases which are not true relational databases. There is no correct definition of such databases but can only be described with a set of common characteristics such absence of a fixed schema, inherent scalability features, high performance, data etc. These databases have come to be known as NoSQL databases. Various companies are seeing the advantages of NoSQL and want to migrate to these databases. But they find it difficult to migrate their data as a lot of study and analysis is required. Each type of database have their own terminology and query language. We propose a novel automated migration model which utilizes the power of service oriented architecture to help these companies easily migrate to NoSQL databases of their choice. We utilize web services which encapsulates few of the most popular NoSQL databases such as MongoDB, Neo4j, Cassandra etc. so that inner details of these databases are hidden yet providing efficient migration of data with little or no knowledge of the inner working of these databases. As proof of concept relational data was migrated successfully from Apache Derby database to MongoDB, Cassandra, Neo4j and DynamoDB, each vendor representing a different type of NoSQL database.

***Keywords*:** NoSQL, Service Oriented Architecture, Cassandra, MongoDB, Neo4j,Amazon DynamoDB

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

For several decades, we have been using relational databases for various software, websites and web applications. But these databases have several problems like the impedance disparity (i.e. The mismatch between what application developers require as data and how data is actually stored on disk), the inability to support clusters naturally etc., the inability to horizontally scale easily as data grows, etc gave enough motivation to many large players in the industry to look for alternatives to relational databases. But alternatives to relational databases had many disadvantages than advantages. So Google and Amazon succeeded in creating completely new databased(Google BigTable,Amazon Dynamo etc) that can easily run on clusters easily scale according to the very high velocity of data. Later, many other companies gave their own solutions and these formed a group of databases called NoSQL or Not Only SQL which refers to databases that dont use SQL as their query language.

## 1.1 NoSQL databases

NoSQL databases do not give a proper definition, but have certain characteristics such as

- Do not have the normal relational model/absence of schema
- Easily support cluster and horizontal scaling
- Usually open source
- Ready for the high volumes of traffic of Web 2.0
- Usually aggregate oriented

## 1.2 Service Oriented Architecture

It is an architectural design pattern that views the application in the form of services. Most often not all services need to be built from scratch. We can reuse these services from if these services were put in a common location. Later we compose these individual services into a single composite service which can be reused for some other application. How the services work is abstracted. SOA applications support distributed computing inherently and can adapt to the rapidly changing technologies.

## 1.3 Motivation

The data migration market is on the rise as more and more companies want to migrate to NoSQL databases. The survey by Philip and Carl Potter [3] shows that data migration is very important and cost and time overruns usually happens. The proposed model tries to reduce both time and cost overruns by eliminating the need to learn these new technologies for migration.

The budget overruns for the data migration market reached $906 million in 2012 while it was $562 million in 2007. This shows a substantial money can be saved if migration with the right tools.

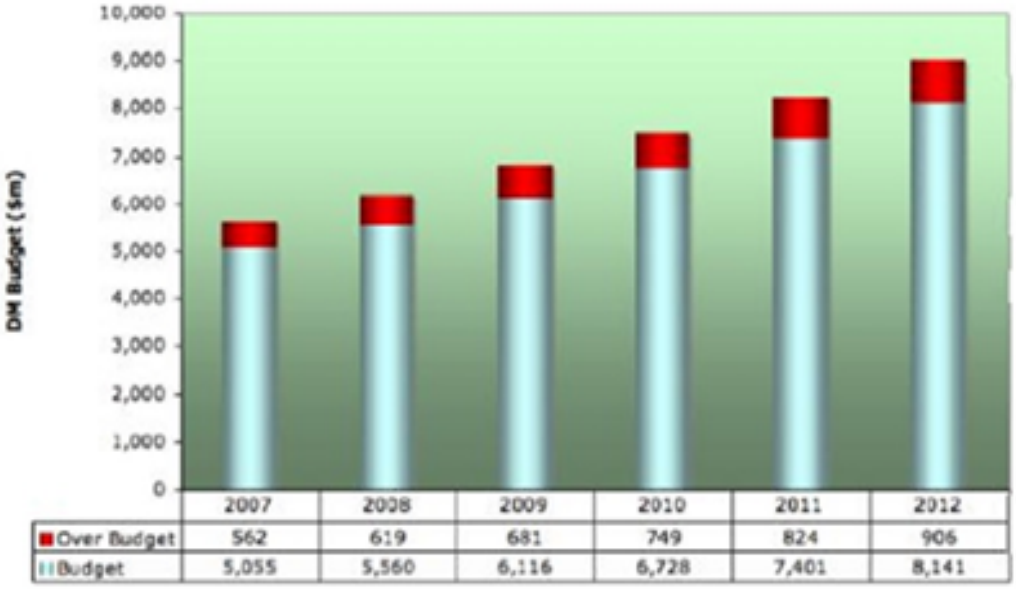


Figure 1.1: Global Data Migration forecast and overruns 2007-12

A lot time and money is spent on the migration process. A unified solution does not exist which help enterprises easily migrate to NoSQL databases. The model tries to achieve a comprehensive and a unified solution to migrate data to the required NoSQL database.

Service oriented design strategy was utilized due to the following reasons

- New NoSQL vendors come up every day and to keep up with constantly changing technology SOA is the right choice.
- Inherent support for distributed operation and scale easily
- A particular vendor might be optimized for a particular language, hence using it would be the right choice. Individual services can be in any language in SOA, thus making it a wise choice.

## 1.4 Objectives

1. To design a migration model that will help in migration of relational databases to NoSQL databases using service oriented architecture.
2. To create a web based solution for migrating various relational databases to NoSQL databases such as MongoDB, Cassandra etc.
3. To provide a multi-vendor NoSQL solution to the problem of data migration to NoSQL databases focusing on the main four types of NoSQL databases.
4. To use the principles of SOA to ensure that the solution can be
   (a) Distributed inherently
   (b) Loosely coupled components
   (c) Adapt to changes in technology rapidly
   (d) Technology/Language independence
5. Create a BPEL process to orchestrate the following web services
   (a) Services to read from relational databases
   (b) Services to analyze the schema of relational database
   (c) Services to insert into NoSQL database

## 1.5 Organization of the Thesis

The rest of the thesis is organized as follows:

1. Chapter 2: In this chapter we present the baisc concepts and definitions of utmost importance for understanding NoSQL and SOA.

2. Chapter 3: In this chapter we present the literature review where we have described some existing works on data migration.

3. Chapter 4: In this chapter we present our proposed multi-vendor data migration model and its implentation .

4. Chapter 5: In this chapter we provide a case study and generated results on migration.

# Chapter 2

# Basic Concepts

## 2.1 Basic Definitions

In this chapter we will discuss some basic concepts of NoSQL databases and Service Oriented Architecture.

## 2.2 NoSQL

Since the last decade, several database management systems have emerged. With the advent of Web 2.0 several companies such as Google, Amazon, Facebook, twitter etc. realized that the relational databases cannot handle the volume nor the velocity of information such as social networking data, logging data etc. as their structure keeps changing on a daily basis. Also the relationship between objects are not shown in relational databases clearly and a lot of join queries are required to gather all the related data. The relational databases has no native support for horizontal scaling ie. For clusters and almost all web 2.0 companies used clusters. Fitting relational database to the cluster was a very difficult task. This lead to the rise of NoSQL databases.

### 2.2.1 Aggregate Data Modeling

Information in relational databases is stored in the form of rows (tuples). A row is constrained as it can combine only a set of values, so we are not allowed nest one row inside another row nor contain a list of values inside within another .

In certain areas such as biological information it becomes necessary to have nested data. Aggregate orientation helps to solve this problem. Aggregate orientation enables us to view data in terms of complex structure rather than in terms of records. Hence nesting is possible.

The term aggregate comes from the Domain-Driven Design and it is a collection of related objects usually considered as a unit i.e. a unit for data manipulation. Seeing data as aggregates make it very easy for the databases to work on a cluster, as these aggregates are natural units for sharding and replication. Aggregates are easier for programmers than tuples.Several NoSQL databases use aggregate orientation to store their data [5].

### 2.2.2 CAP Theorem

NoSQL data stores are often associated with the CAP theorem. Consistency, Availability and Partition tolerance (CAP) theorem states that out of the three we can ensure only two at the same time.

- **Consistency:** If multiple users are viewing/updating the same data there must be a system in place to ensure all the data are the same for all users at the same time.
- **Availability:** As the name suggests, if a user at a any point in time is communicating with a node in a cluster, it must respond correctly.
- **Partition tolerance:** It refers to the fact that even if there are

communication breakages in the cluster, the cluster should function normally as multiple smaller clusters.

### 2.2.3 Types of NoSQL

There are more than 150 NoSQL vendors currently in the market. These vendors' databases can be classified into one of 4 categories are discussed below:

#### Key-value databases

The key-value data store is analogous to the hash table in Java,C# etc. except that it is stored on secondary storage. As mentioned the each data is stored in the form of keyvalue pair but the value can be multi-valued. Unlike RDBMS the value need not be same across all tuples. E.g.

```
{
Sessionid:123223
Userprofile:{
Name: Rohan,
Age: 23

    }
}
```

Figure 2.1: Example of key-value data

As shown above the key is the sessionID and value is userprofile. The value need not be an object, it can be list, set, hashes etc. One of the most popular databases is the Riak databases. Here bucket corresponds to a table, key-value corresponds to a row/tuple.

Other popular Key-value stores include Amazon DynamoDB ,project voldermort, LevelDB , BerkeleyDB, , TMemcached etc.

### Document databases

In document database the main unit is known as the document. These databases store and retrieve documents which are represented using XML (extensible Markup language), JSON (Java Script Object Notation), BSON (Binary encoded JSON). Main characteristics of these documents are

1. Self-descriptive
2. Hierarchical tree data structures
3. Contains scalar values, nested values, maps, collections etc.

MongoDB is one of the most popular document databases. In MongoDB schema corresponds to database, a collection corresponds to a table, document corresponds to a row in relational database. One of the main feature of MongoDB is the replica sets through which it ensures high availability. Replication is done in master-slave configuration in clusters. Nodes can be added easily without any downtime unlike relational databases. This is crucial for several companies.

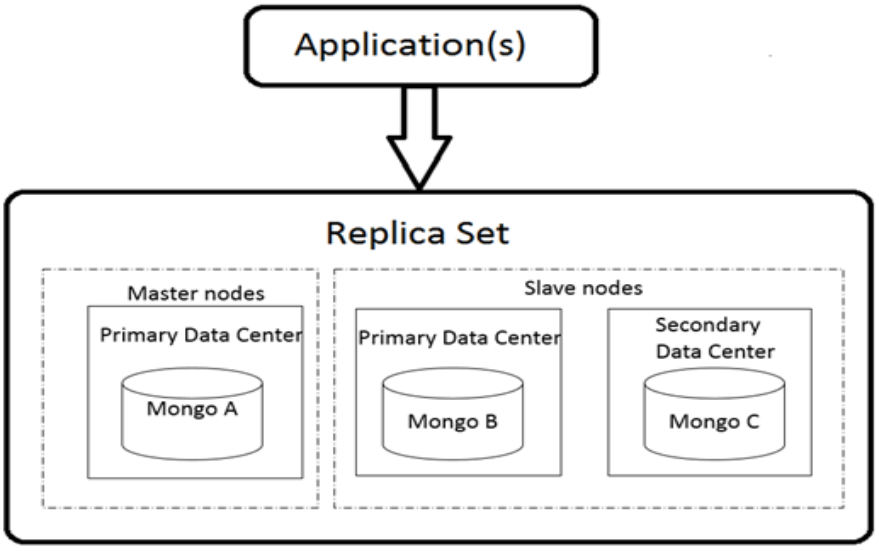


Figure 2.2: Replica Set in MongoDB in master-slave configuration

MongoDB and other document NoSQL databases are usually used for event logging, i.e. order processing event, call event, etc., content management systems, web and business analytics, real-time analytics, and also in e-commerce systems.

```
{ "name":"Rohan"
"likes": ["Skiing","Parasailing","Chess","Gaming"]
}
```

Figure 2.3: Example of data document in MongoDB

The example given above is a typical document in MongoDB represented in JSON format which corresponds to a row in a relational database. The important thing is that the next row need not have the same attributes unlike relational database. The schema is not fixed in document databases.

Other document data stores include RavenDB , CouchDB,Terrastore etc.

### Graph Databases

Relational database fails to capture the relationships between people and objects as the focus was not relationships but objects. But a lot of times it becomes necessary to extract relationships information. Graph databases enable to store data as nodes and relationship as edges. Both nodes and edges have properties. The directional significance of edges also exists. For example, in e-commerce platform when a person buys and likes the product that he just bought and his friend views the same e-commerce website, the application can recommend products based on his as will his friend's choices. This information would require complex queries in RDBMS, but in graph databases it is very easy to query the data.

Unlike RDBMS relationship is not obtained during query it persists as data, hence make it faster and easier to obtain data.Unlike other NoSQL databases, graph databases do not come under the aggregate orientation, they are more of a relationship oriented in nature. These databases can be used for routing, location-based services and recommendation services in e-commerce platform.

Among the graph databases Neo4j is quite popular. It uses the cypher query language for retrieval and storage of data in the database. Neo4j supports all ACID properties unlike other NoSQL solutions mainly because of the inherent non-aggregate orientation of these databases. Other popular graph databases include HyperGraphDB, OrientedDB etc.

**Column-Family**

Data is stored with the help of a key which is mapped to set of values and the values are gathered into several column families, where each column family is a map of the data.

One of the most popular Column-family database is Cassandra. Twitter has a portion of their data stored on Cassandra and this has led to the increase in their performance substantially. In Cassandra keyspace corresponds to a database, column family corresponds to a table, and the most vital difference is that the column is same for all rows in a relational database but in Cassandra and other column-family type data stores column can be different for each row. We store data in column family data stores when data needs to be accessed as whole not the part. Customer profile information is accessed always together, but not all orders of the customer are viewed together. Hence profile information becomes a good candidate to be stored in Cassandra and other column family stores.

The basic unit is a column, unlike relational databases where the unit is a row. The column basically is a key-value pair usually stored with a timestamp. Timestamp

plays in an important role is resolving write inconsistencies, etc. E.g.:

```
{ "name":"Rohan"
"likes": ["Skiing","Parasailing","Chess","Gaming"]
}
```

Figure 2.4: Example of column in Cassandra

The above example describes a column with the key being the first name and value being Rohan. Set of such columns forms the column family Other column family databases include HBase, Hypertable etc.

### 2.2.4 Service oriented Architecture

The concept of service orientation is nothing new. It uses the age old principle of divide and conquer and code reusing. SOA applications can be viewed as a composition of services. The only difference is that these services need not be directly owned by the same company. The solution to any problem is now viewed in terms of two keywords

1. Service
2. Messages

In order to support platform and language neutrality, the messages are in XML and uses the SOAP (Simple Object Access Protocol) protocol. Not all services are built from scratch.

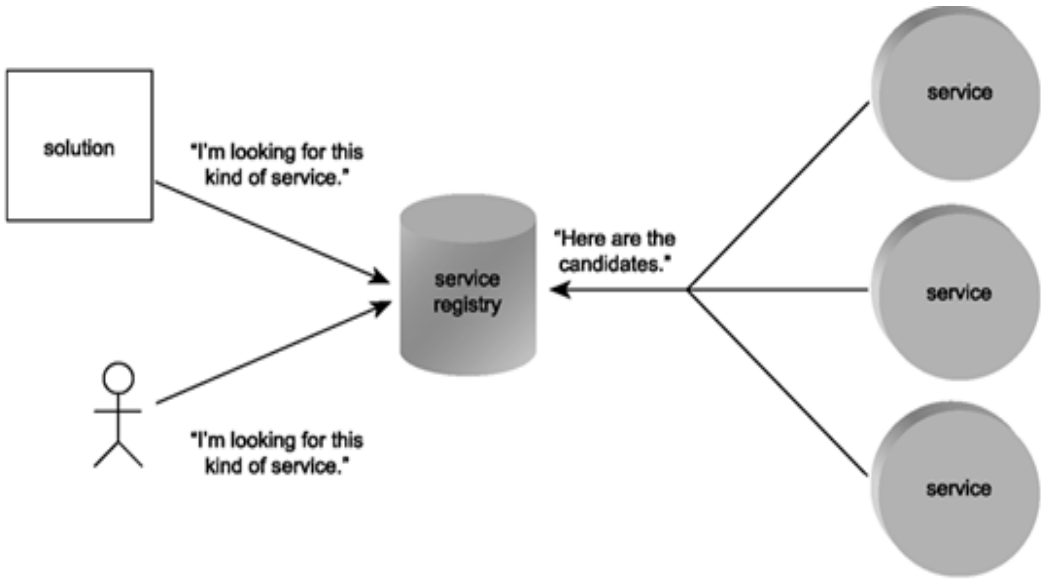


Figure 2.5: Service registry for candidate services

The Possibilities are

1. Services are built from scratch as the service is not available anywhere
2. Services are built from legacy applications
3. Services are used from other companies based on a contract

But just using services does not mean that the application is service oriented. It must follow principles of SOA design to become SOA application. Various design principles of SOA include:

1. Loose coupling
2. Granularity
3. Process Coupling
4. Service Contract

5. Abstraction
6. Reusability
7. Composability
8. Autonomy
9. Discoverability
10. Statelessness

## 2.3 Summary

In this chapter, we have discussed briefly about the need for NoSQL, various NoSQL database types, the importance of service oriented architecture and other concepts.

# Chapter 3

# Literature Review

This chapter describes the various data migration models

## 3.1 Schema Conversion

Hainaut et al [6] has defined the schema conversion process. There are two schema conversion strategies. One is the conversion of physical schema which converts any legacy database to a new target database management system as shown in figure 3.1.The second is the conversion of conceptual schema where the database is represented as a conceptual schema, refined and then conceptualized to get a new database as shown in figure 3.2.

## 3.2 Meta Modelling Approach

Jeusfled and John [7] present a meta-model technique for migration of relational database. As the source and target databases are both well understood, first a mapping from source database to destination system is done. A meta model of the source database and the destination database is created and then a relationship is created between them. Later the data is actually migrated.

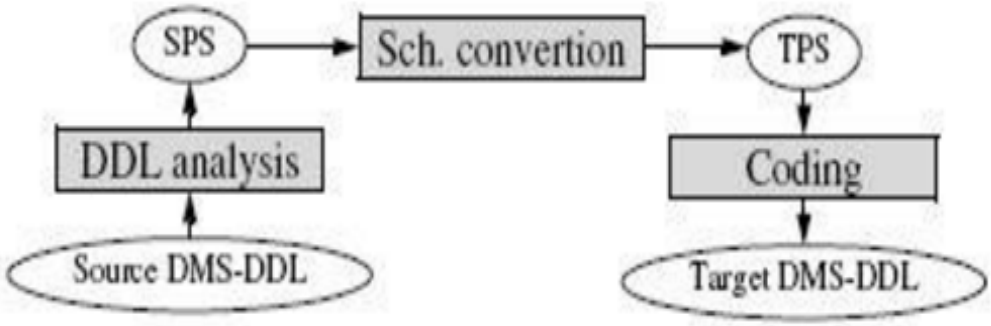


Figure 3.1: Physical Schema Conversion

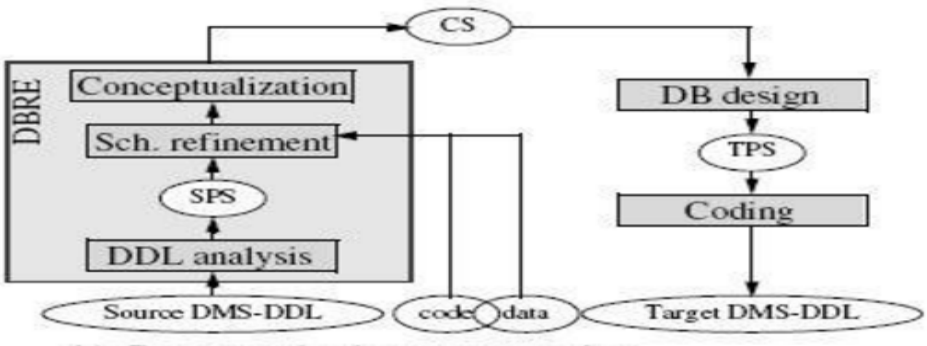


Figure 3.2: Conceptual Schema Conversion

Jahnke and Wadsack [8] describes a two-phase process for data migration. The first phase involves attaining the logical schema from the source database. Then in the second phase converts this attained logical schema into a conceptional one.

Maatuk et al. [9] describes three strategies. First strategy involves OO/XML interfaces to handle the relational data. The second technique connects relational database to multiple databases. The last approach migrates the relational data to the destination database. Figure 3.3 describes the model.

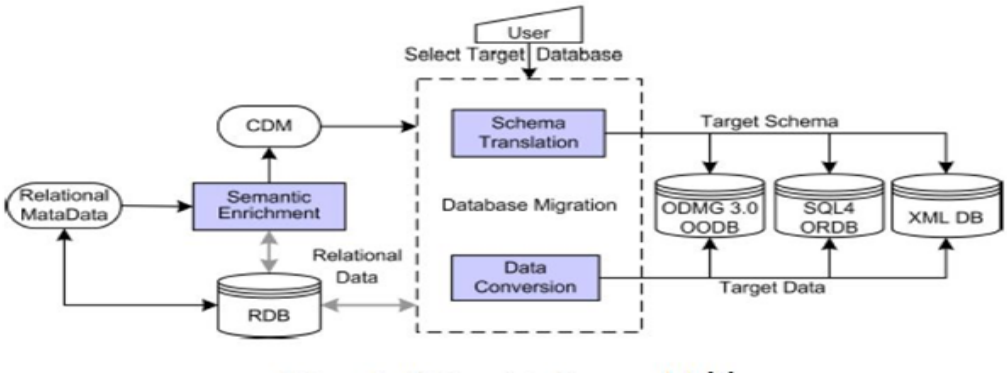


Figure 3.3: Meta modelling

## 3.3 Extract, Transform and Load

Haler et al. [10] proposed the ETL (Extract, Transform and Load) migration model. The figure describes the model. The extraction step extracts the data and places it on a different server. The filtering step filters the data. The transformation step restructures the data and does the mapping of data from source system to the target system. After the transformation step the data is uploaded on the target server.

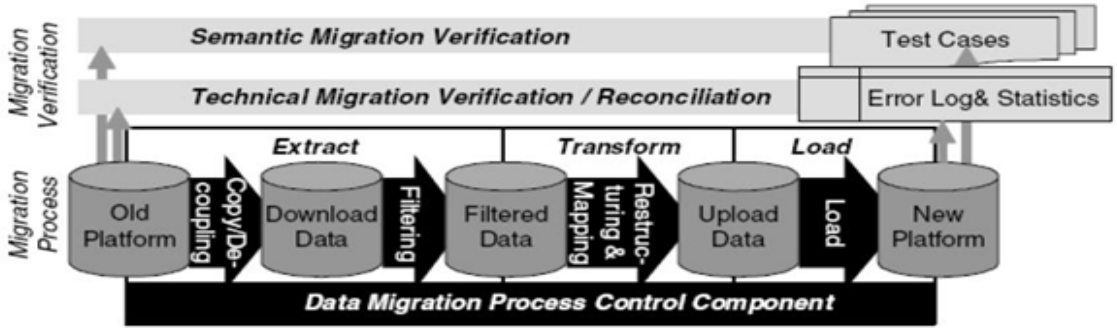


Figure 3.4: Extract, Transform Load

## 3.4 Integration Model

Border et al [11] describes a model which is integrated to the software development steps. A model generator creates a model from annotated UML model using the generic database adapter which actions as the database access layer. From the database adapter the upgrade generator takes the old model, the new model just generated and the auxiliary property file to get the upgrade program API. The upgrader program API is used for cloning of database and migration of data.

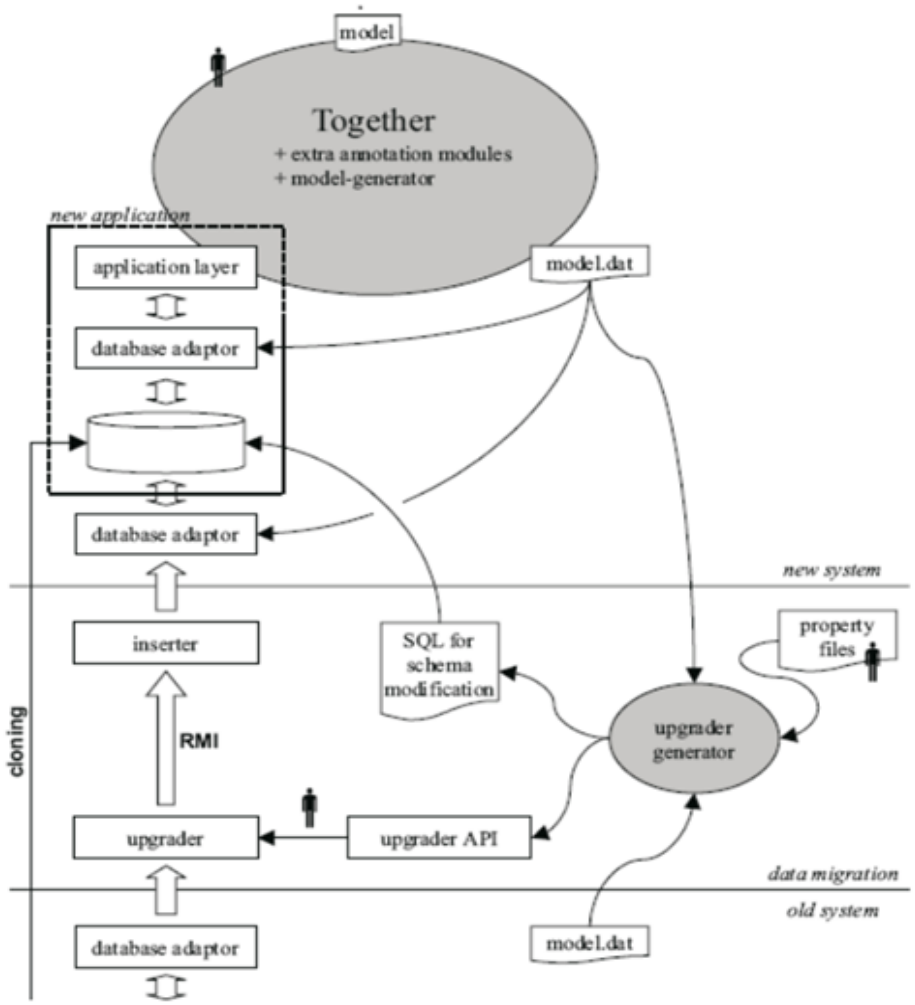


Figure 3.5: Integration Model

## 3.5 Cloud Migration model

KushalMehra et al. [1] in "Automated Migration into the Cloud "proposed a model to migrate a relational database to NoSQL databases and implemented a standalone application which works for Microsoft SQL and Amazons SimpleDB. Here they have proposed four migration approaches which migrate relational data to NoSQL databases in the cloud.

In the first method all the tables in the relational databases were migrated to a single domain (corresponds to table in relational) of the Amazon DynamoDB. The main drawback is that it was limited to 10 GB as a single domain in AmazonDB is limited to 10 GB. In the second method the tables of a particular request are migrated to a single domain and the rest of the tables are migrated to another domain. In the third method each relational table is migrated to a domain of Amazon SimpleDB. The fourth method denormalizes the data before transferring to the target system. The denormalized data is then inserted into a single domain.

Table 3.1: Comparison of various migration methods

| **Migration Method** | **Type 1** | **Type 2** | **Type 3** | **Type 4** |
|---|---|---|---|---|
| Storage <10GB | Supports | Supports | Supports | Supports |
| Storage >10 GB | Doesn't support | Supports | Supports | Supports |
| Sharding | Doesn't support | Supports | Supports | Supports |
| Joins | Limited to one domain | Limited to one domain | Cross domain | Limited to one domain |
| Denormalization | Doesn't support | Doesn't support | Doesn't support | Supports |
| Storage Cost | Nearly same of Type 2, Type 3 | Nearly same of Type 1, Type 3 | Nearly same of Type 1,Type 2 | Nearly same of Type 1,Type 2, Type 3 |
| Computation Time | Smallest | Larger than Type 1 | Highest | Larger than Type 2 |

The table 3.1 depicts the comparison of the various migration methods. An enterprise should go for the type 1 migration if the database size is less than 10GB otherwise go for any of the type 2 or type 4. Also, only type 2, type 3, type 4

has support for sharding. Type 4 is better than type 2 as type 4 denormalizes the data and goes with the aggregate orientation principles. Type 3 is needed when the target domain should have the same logical schema as that of relational databases. The computation time is highest in type 3 and least for type 1.

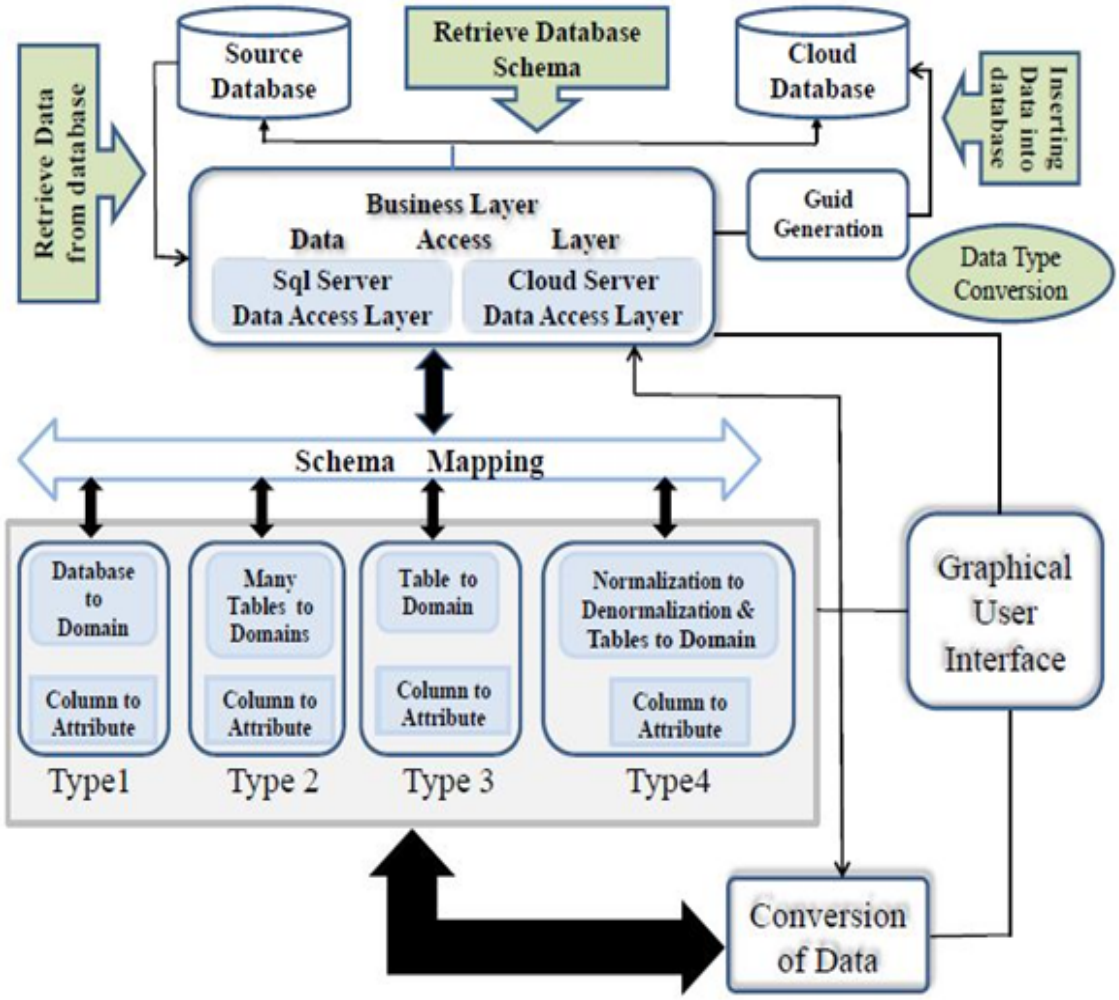


Figure 3.6: Automatic Migration Model for Amazon SimpleDB

Figure 3.6 describes the architecture of the automatic migration system. It consists of the business layer, the schema mapping layer, the data access layer , the data conversion layer and the Guid generation.

The main drawbacks of the model are

1. Not a multi-vendor approach The model supports only AmazonDB SimpleDB which is just one of the many NoSQL databases. There are several other vendors such as MongoDB, Neo4j, Cassandra. Enterprises may require a multi-vendor approach for supporting polyglot persistence.

2. Not a distributed approach In this age where the amount of computation is high and number of computers are abundant, it is imperative that we should follow the distributed approach.

## 3.6 Summary

The chapter discusses about various migration models proposed by several experts.

# Chapter 4

# Data Migration Model

## 4.1 Model description

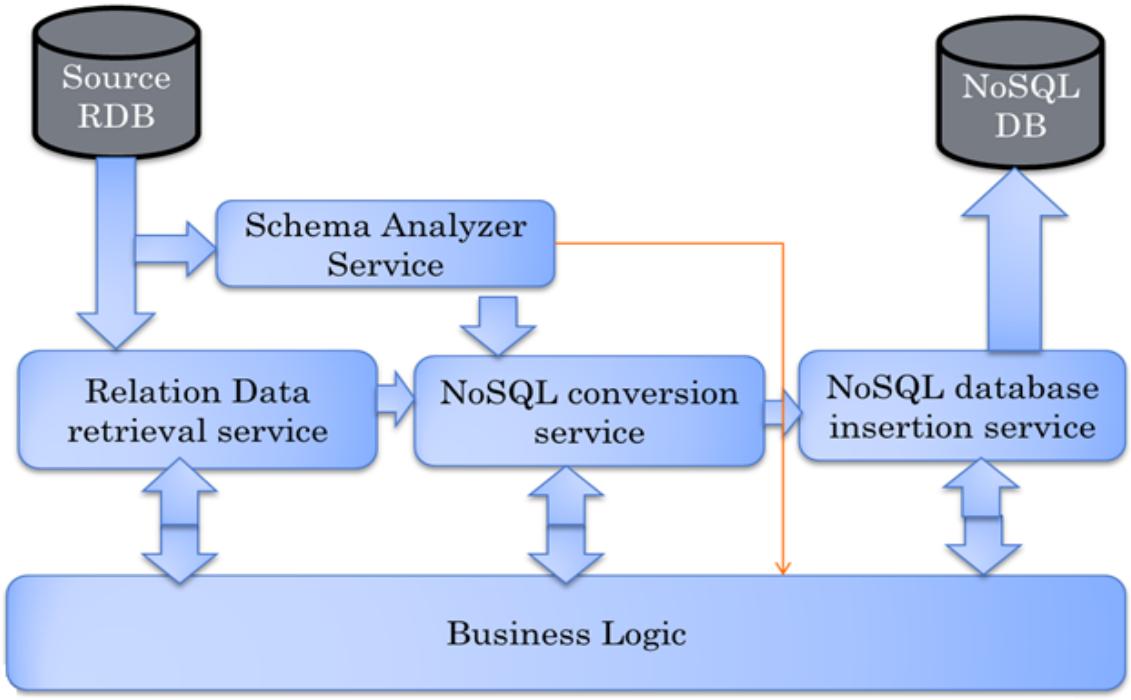


Figure 4.1: Relational-NoSQL Migration Model

The migration model is built using the Service Oriented Architecture. Multiple services are involved in the model. They include

1. Schema Analyzer
2. Relational Data retrieval service
3. NoSQL conversion service
4. NoSQL database insertion service

All services are orchestrated using the business process enterprise logic (BPEL). The most important is the schema analyzer service. This service analysis the schema of the given relational database. This service finds all the tables and relational schema mappings. The data retrieval service collects all the data from selected tables for migration. The NoSQL conversion service is a composite service. This service is composed of MongoDB insertion service, Cassandra insertion service, Neo4j insertion service and Amazon DynamoDB insertion service.

Based on the required NoSQL database the corresponding NoSQL database conversion service and NoSQL database insertion service is invoked. The NoSQL conversion service converts all the data collected by the data retrieval service into the intermediary format in JSON. The database insertion service inserts the converted data into the NoSQL database. All services communicate through the Business Logic/ Enterprise bus. Business Process Execution Language (BPEL) defines a notation for specifying business process behavior based on Web Services.

## 4.2 Algorithms

This section discusses the various algorithms used in the model.

**Algorithm 1** ReadFromDB web service

***Input***: U //Source table URL location, L //List of Tables

***Output***: S // database in JSON XS:String format

**Begin**

1: Analyze tables to find whether join operation can be applied on list of tables using primary-foreign key relationship information from source table
2: **for** each table T in L **do**
3: **if** If primary-foreign key exists in T with rest of tables of L **then**
4: Execute join select query on table and store in resultSet D
5: **else**
6: Excecute select query on table and store in resultSet D
7: **end if**
8: Add column header information of T into D
9: **end for**
10: Convert D into JSONArray format J
11: Convert J into XS:String format for BPEL transfer
12: Exit

The algorithm 1 analyzes the schema of the relational database and finds out if primary key - foreign key relationalship exists. If such a relationship exists then combine the tables involved in the relationship and store in a dataset otherwise directly store in dataset. This dataset is converted into JSON string for web transportation.

**Algorithm 2** InsertIntoMongoDB web service

***Input***:X table content in JSON XS:String format

***Output***: O // MongoDB data in BSON format

**Begin**

1: Convert X to JSONArray J
2: **for** each JSONObject O in J **do**
3: Convert JSONObject into BSON object
4: Insert BSON object into MONGO DB using the column header information from X
5: **end for**
6: Exit

The algorithm 2 converts individual JSON objects from the input json string into BSON object. The BSON object along with header information is inserted into mongoDB data store.

**Algorithm 3** InsertIntoDBCassandra web service

***Input***:X table content in JSON XS:String format

***Output***: O // Cassandra data in JSON format

**Begin**

1: Convert X to JSONArray J
2: **for** each JSONObject O in J **do**
3: Insert JSONObject with the column header information from X into cassandra using CQL
4: **end for**
5: Exit

The algorithm 3 inserts individual JSON objects from the input json string into BSON object along with header information is inserted into cassandra data store using CQL.

**Algorithm 4** InsertIntoAmazonDB web service

***Input***:X table content in JSON XS:String format

***Output***: O // Amazon DynamoDB data in JSON format

**Begin**

1: Convert X to JSONArray J
2: **for** each JSONObject O in J **do**
3: Insert into JSONObject and column header information using Amazon DynamoDB API into Amazon DynamoDB.
4: **end for**
5: Exit

The algorithm 4 inserts individual JSON objects from the input json string into BSON object along with header information is inserted into Amazon dynamoDB using amazon dynamoDB API.

**Algorithm 5** InsertIntoNeo4j web service

***Input***:X table content in JSON XS:String format

***Output***: O // Neo4j nodes and edges information in JSON format

**Begin**

1: Convert X to JSONArray J
2: **for** each JSONObject O in J **do**
3: Insert into JSONObject and column header information using Cypher Query Language
4: **end for**
5: Exit

The algorithm 5 inserts individual JSON objects from the input json string into BSON object along with header information is inserted into Neo4j data store using Cypher Query Language.

## 4.3 Summary

In this chapter, the migration model was discussed in detail. The model uses various services. Algorithms of various services were also discussed.

# Chapter 5

# Implementation of Migration Model

## 5.1 Service oriented Model

As proof of concept we have implemented a system which takes in Apache Derby relational database and migrates data to MongoDB, Cassandra, Amazon DynamoDB or Neo4j data store based on user choice.

The Figure 5.1 shows the implementation of the system using OpenESB v 2.3. It shows the BPEL (Business Process Enterprise Logic) of the system. Initially user input is taken and then the readDB service is invoked to read the Database. Based on the users choice either MongoDB insertion service, Cassandra insertion service, AmzonDB insertion service or Neo4j insertion service is invoked. These services need not be on the same server. The Figure 5.2 shows the source code of the BPEL in XML.

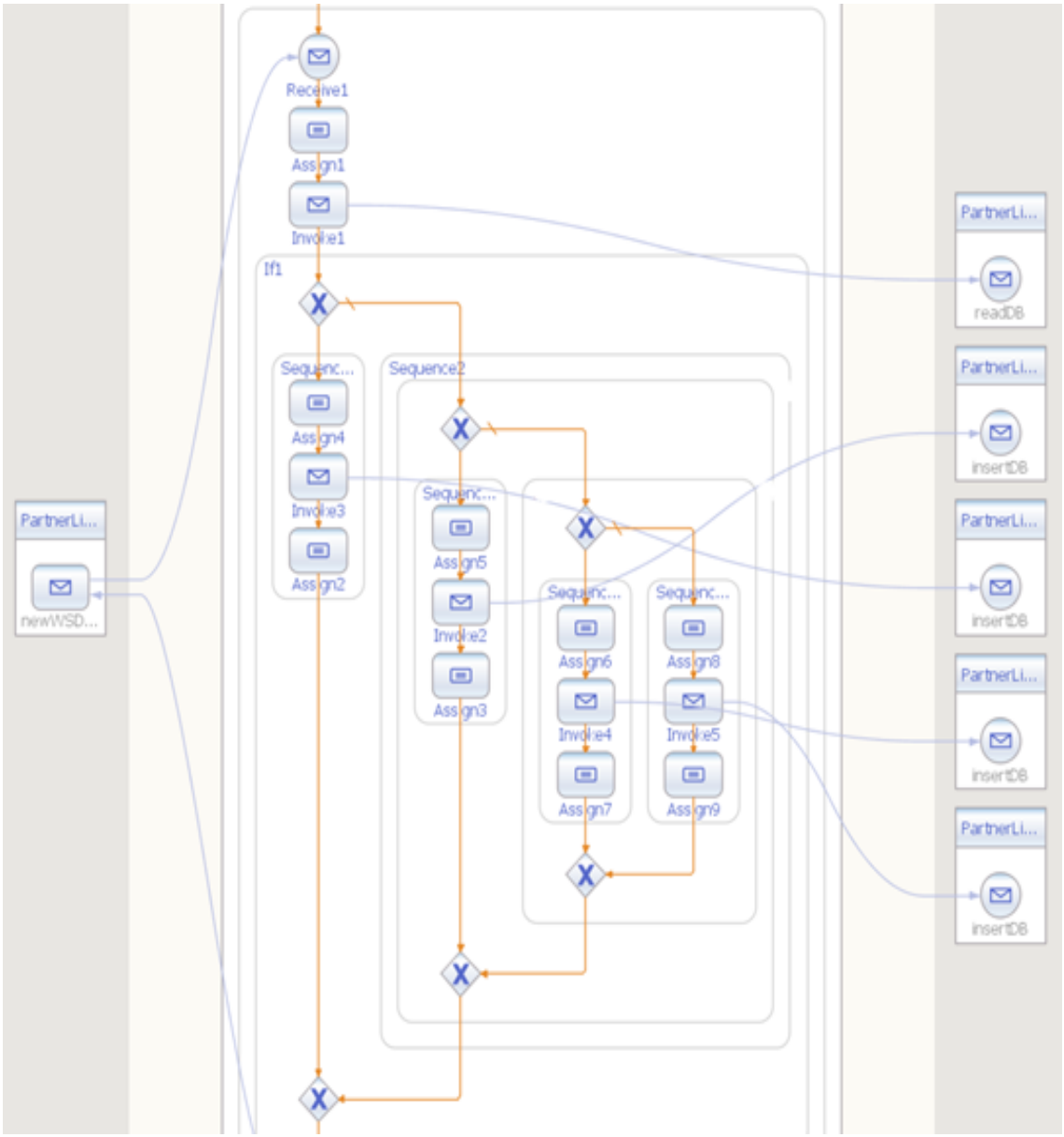


Figure 5.1: Design of migration models BPEL

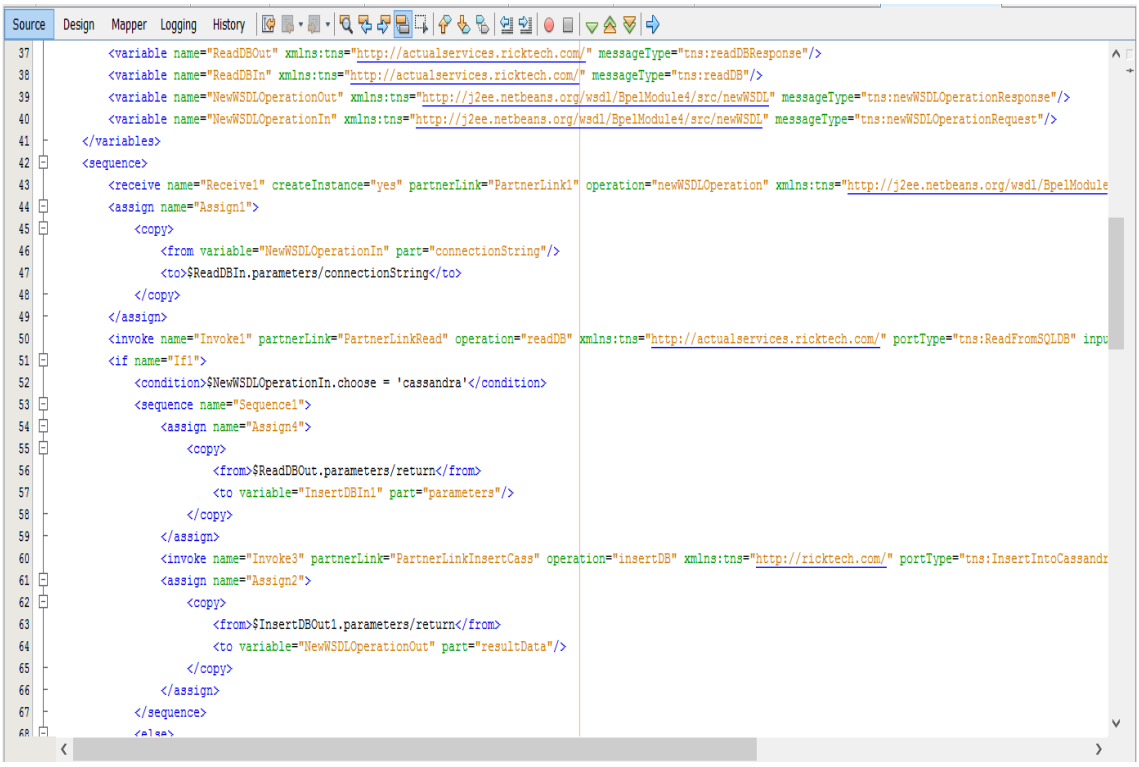


Figure 5.2: XML Source of migration models BPEL

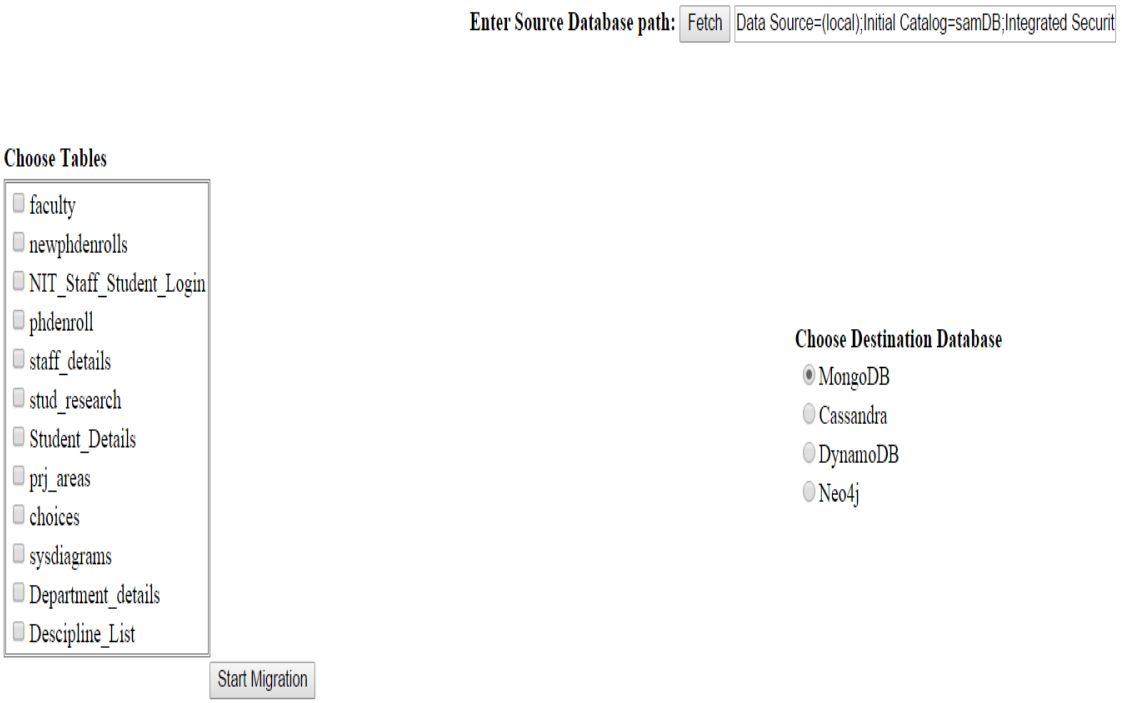


Figure 5.3: GUI of the migration model implementation


Figure 5.4: Textbox for source database input

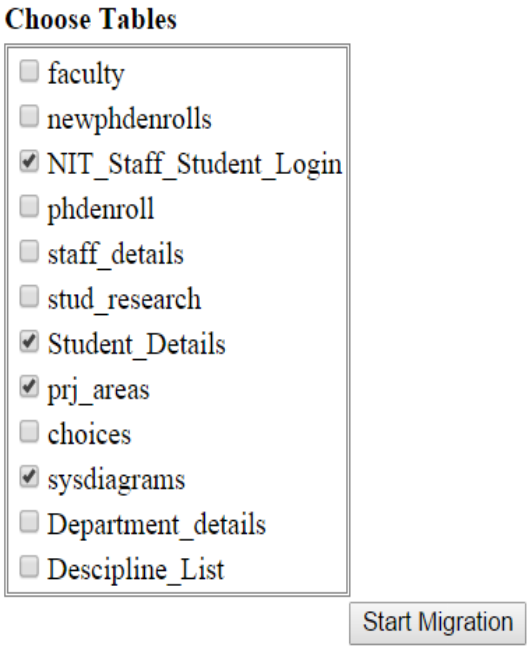


Figure 5.5: Checkbox for selecting Input Tables from database

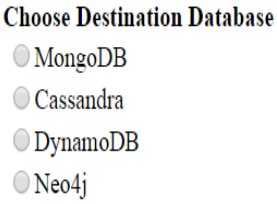


Figure 5.6: RadioButton for choosing target NoSQL vendor data store

The figure 5.3 shows the graphical user interface(GUI) for the migration model. The front end was developed in ASP. Net and c#. The model is implemented in such a way that it can be accessed from any computer provided the IP address is

known. The GUI has several sections. The first section involves the textbox where the path of the relational database is entered as shown in figure 5.4. On clicking on the fetch button the tables in the database are listed in the second section as shown in figure 5.6. Once the required tables are selected the target NoSQL vendor is selected using RadioButton as shown in figure 5.5. There are four options:

1. MongoDB
2. Cassandra
3. Neo4j
4. Amazon Dynamo DB

The start migration button causes the selected tables to migrate to the specified target data store and the result is shown in the next web page.

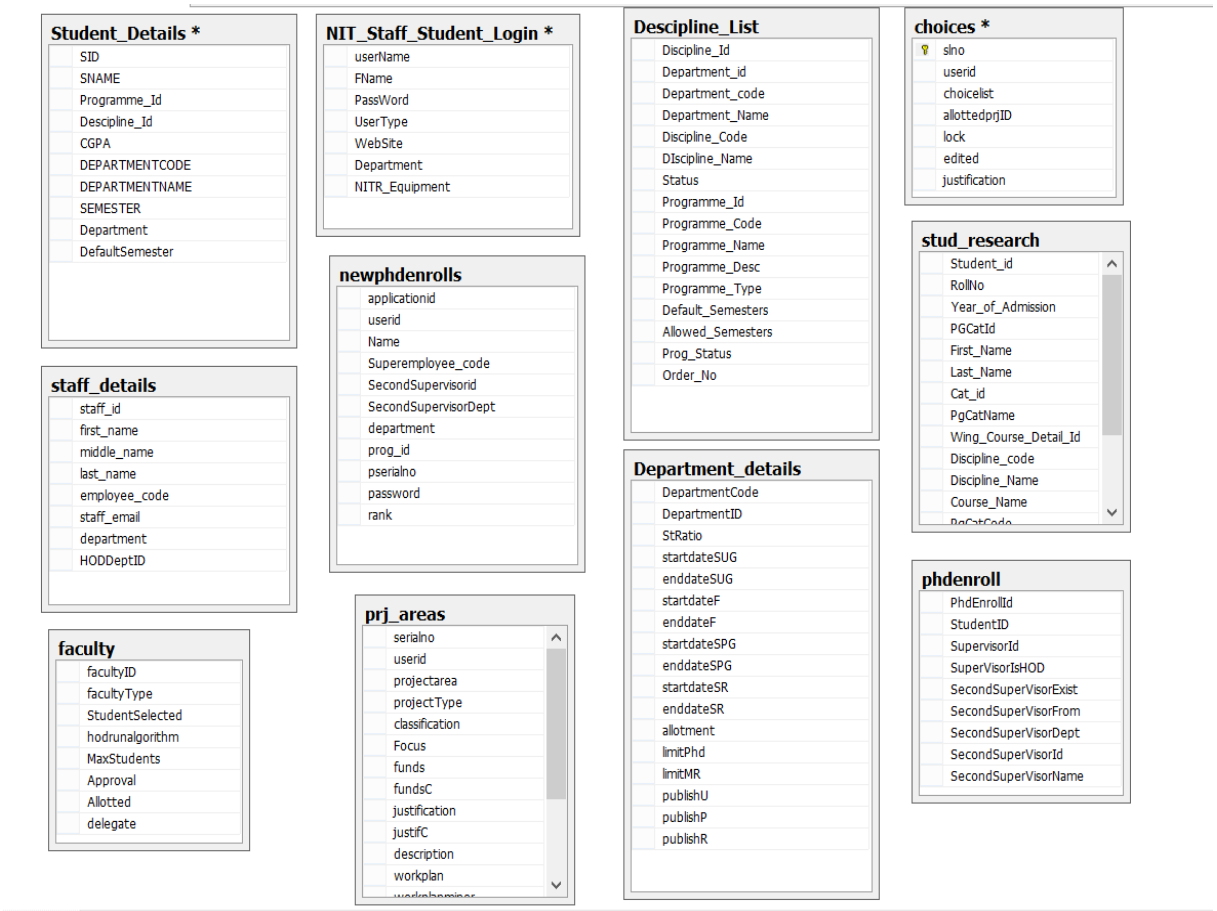


Figure 5.7: Input tables used for migration

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:InsertCassanraResponse xmlns:ns2="http://ricktech.com/">
         <return>


         StaffProfile = {


    Archana = {emailAddress:"archanam@nitrkl.ac.in", employee_code:1090912,department:16},
    Samit = {emailAddress:"samit@nitrkl.ac.in", employee_code:1090916",department:16},
    Dinabandhu = {emailAddress:"samit@nitrkl.ac.in", gender:1090916", department:16},
    Monalisa ={emailAddress:"pattnaikm@nitrkl.ac.in", employee_code:1121052,department:16}
    Manoj = {emailAddress:"pattnaikm@nitrkl.ac.in", employee_code:1110965,department:16}
    Sunipa = {emailAddress:"bhattacharyyas@nitrkl.ac.in", employee_code:1110957,department:16}
    Sudip = {emailAddress:"bhuyanp@nitrkl.ac.in", employee_code:1110963,department:16}
    Prasanta = {emailAddress:"", employee_code:1110958,department:16}
    Ratnakar = {emailAddress:"ratnakar@nitrkl.ac.in", employee_code:1110960,department:16}
    Natraj = {emailAddress:"yedlan@nitrkl.ac.in", employee_code:1110961,department:16}
    Pitamber = {emailAddress:"pitam@nitrkl.ac.in", employee_code:1110966,department:16}
    Ashok = {emailAddress:"mondala@nitrkl.ac.in", employee_code:1110970,department:16}
    Indranil = {emailAddress:"banerjeei@nitrkl.ac.in", employee_code:1110976,department:16}
```

Figure 5.8: Cassandra data store migration output

The Figure 5.7 shows the input table from an existing project. The table consists of 11 tables. Information is about student and staff details of an institute with 21 departments. Each department has information of both under graduate and undergraduate students. There more than 2000 records of students and staff members. Staff details were migrated to cassandra and Amazon DynamoDB as shown in Figure 5.8 and Figure 5.9

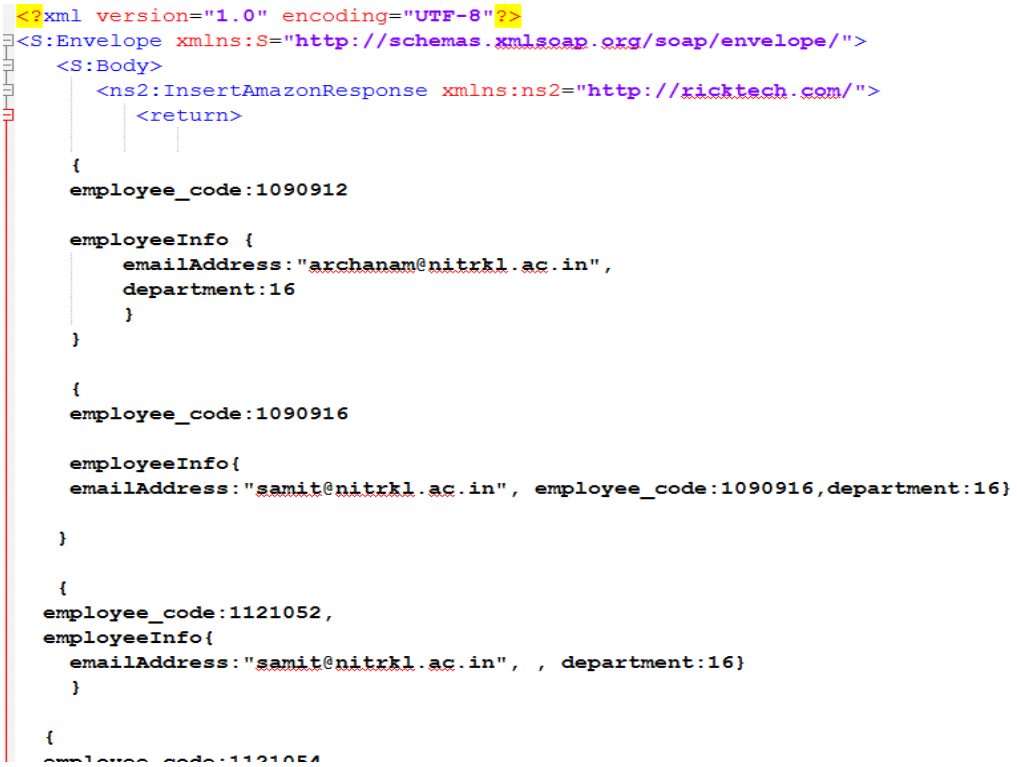


Figure 5.9: Amazon DynamoDB data store migration output

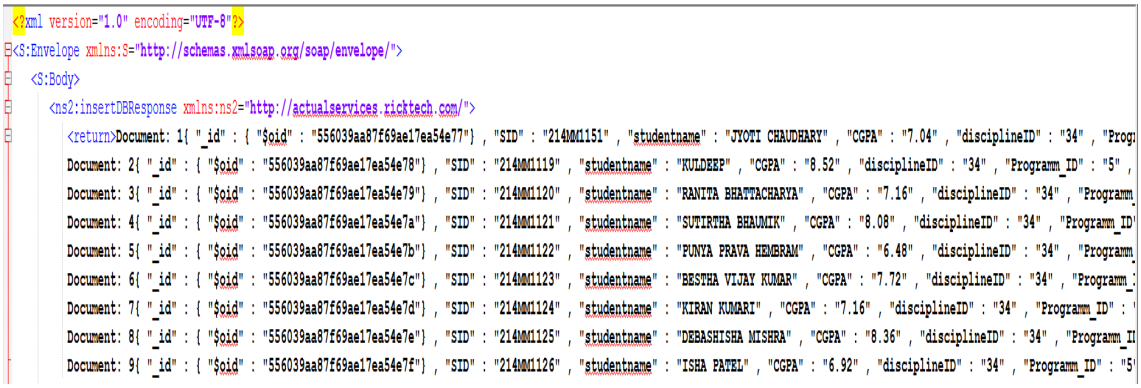


Figure 5.10: MongoDB data store migration output

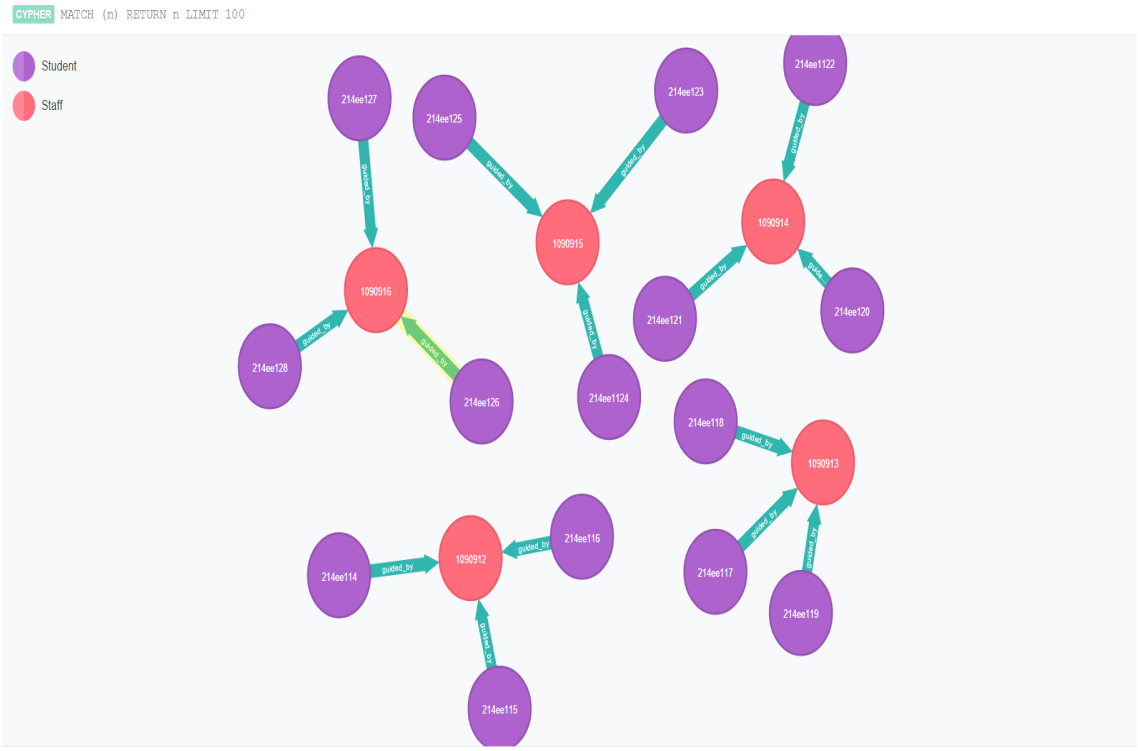


Figure 5.11: Graphical visualization of Neo4j data store migration output

Figure 5.10 shows the output of MongoDB data store after the migration process. Several documents of containing information of the students stored in MongoDB BSON format is shown in the figure.

Figure 5.11 shows graphical visualization of Neo4j data store output after the migration process. There are two types of nodes. Faculty node is represented in red colour and the student node is represented in purple. The relationship is 'guided by' indicating which students are guided by which faculty.

## 5.2 Summary

In this section the implementation of the migration model was described in detail. The interface was also briefly discussed.

# Chapter 6

# Conclusion and Future Work

Over the last decade, distributed computing has been a successful paradigm for web applications. The cloud computing is a billion-dollar industry. DBMSs store and serve data for an application, hence data becomes critical and central to a web application. The goal of this thesis is to propose a model and develop a system which migrates relational databases to most popular NoSQL databases. This report provides techniques and a model which will help software industries to migrate their existing relational databases to the NoSQL vendors using service oriented Architecture. In effect a layer of abstraction is created for easy migration to NoSQL databases. Web services were created to analyze schema of Relational database and apply schema conversion automatically. We also added support for other NoSQL databases using web services to achieve multi-vendor support such as MongoDB, Cassandra, Neo4j and Amazon DynamoDB.

There are more than 150 NoSQL vendors. We intend to provide support for other popular data stores. This process is easy as the model was created using service oriented architecture which is a loosely coupled system. The system currently takes in users choice of target NoSQL data store. But the choice is not very easy. We plan to use neural networking to assist users in the choice of target database in our future research work.

# Bibliography

[1] K.Mehra, Y.Yan and D.Lemure, Automatic data migration to the cloud in the Sixth International workshop on Cloud Data Mangement (CloudDB 2014)

[2] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In Pervasive computing and applications (ICPCA), 2011 6th international conference on , IEEE, 2011

[3] P. Howard and C. Potter., Bloor research: Data migration in the global 2000 - research, forecasts and survey results

[4] Andre Calil and Ronaldo dos Santos Mello, Simplesql: a relational layer for simpledb In Advances in Databases and Information Systems, Springer,2012 .

[5] Sadalage P.J and Fowler .M, 2013, NOSQL Distilled, Pearson, p.99-109

[6] J . Henrard , M. Hick, P. Thiran , and J. Hainaut . Strategies for data reengineering . In Reverse Engineering, 2002. Proceedings. Ninth Working Conference on, pages 211220. IEEE, 2002

[7] M. A. Jeusfeld and U.A. Johnen. An executable meta model for re-engineering of database schemas. Springer, 1994.

[8] J. H. Jahnke and J. Wadsack. Varlet: Human-centered tool support for database reengineering. In Proc. of Workshop on Software-Reengineering, 1999.

[9] A . Maatuk, A. Ali , and N. Rossiter . Relational database migration : A perspective. In Database and Expert Systems Applications, pages 676683. Springer,2008.

[10] K. Haller. Towards the industrialization of data migration: Concepts and patterns for standard software implementation projects. In Advanced Information Systems Engineering, pages 6378. Springer, 2009.

[11] B. Bordbar, D. Draheim, M. Horn, I. Schulz, and G. Weber. Integrated modelbased software development, data access, and data migration. In Model Driven Engineering Languages and Systems, pages 382396. Springer, 2005

[12] Amazon. Amazon DynamoDB. http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/. Retrieved on November 2014.

[13] Apache Cassandra. http://docs.datastax.com/en/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html. Retrieved on December 2014.

[14] Neo4j graph database. http://neo4j.com/developer/get-started/. Retrieved on December 2014.

[15] MongoDB. http://docs.mongodb.org/manual/. Retrieved on June 2014.

[16] A. Thakar and A. Szalay. Migrating a (large) science database to the cloud. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC 10, pages 430434, New York, NY, USA, 2010.ACM.