AUTOMATIC DATA MIGRATION INTO THE CLOUD

Kushal Mehra

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science

(Software Engineering)

Concordia University

Montréal, Québec, Canada

January 2014

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Kushal Mehra**

Entitled: **Automatic Data Migration into the Cloud**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Software Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair

Dr. T. Popa

______________________________ Examiner

Dr. T. Eavis

______________________________ Examiner

Dr. N. Shiri

______________________________ Supervisor

Dr. Y. Yan

______________________________ Co-supervisor

Dr. D. Lemire

Approved by ______________________________

Chair of Department or Graduate Program Director

____

Dr. Christopher W. Trueman, Interim Dean

Faculty of Engineering and Computer Science

Date ______________________________

Abstract

Automatic Data Migration into the Cloud

Kushal Mehra

Relational databases have been used for decades to store data. Using scale up, relational databases require a bigger and bigger server with more CPUs, more memory, and more disk storage to keep all the tables to support more concurrent users. However, big servers tend to be highly complex, proprietary, and disproportionately expensive, unlike the low-cost, commodity hardware. Therefore, it becomes important to store data efficiently and compute with massive amount of data, providing high scalability, providing high performance and availability at low costs. This leads to the invention of cloud databases, for instance NoSQL databases. NoSQL databases have many advantages such as reading and writing data quickly, supporting massive storage and low cost. The scaling approach in cloud databases is scale out, which is used to add multiple servers, and the data structure of storage is in the form of key-value pairs. However, it can be a challenge for enterprises to migrate existing relational databases to highly scalable NoSQL databases on clouds.

In this thesis, we propose an automatic data migration model which will assist enterprises to migrate their relational databases efficiently and transparently to the cloud databases. We propose four migration methods to migrate data in four different ways. Each migration method is independent of the others and stores the migrated relational database in different formats in the cloud database.

We design a system to implement the automatic data migration model. As a proof of concept, we successfully migrated a relational database from Microsoft SQL Server to a cloud database Amazon SimpleDB using four different migration methods. Furthermore, we have conducted extensive experiments on Amazon SimpleDB to evaluate the performance of our model in terms of computational time, storage cost, sharding and redundancy. Based on these experiments and detailed analysis of each migration method, our system allows enterprises to determine which method is suitable for their data migration. Furthermore, our experimental evaluation shows that our solution is promising and can migrate data from the relational databases to the cloud databases.

Acknowledgments

With immense gratitude , I acknowledge my advisors, Professors, Dr. Yan and Dr. Lemire for providing continuous support, mentoring, guidance over the course of my master's study. I heartily thank my advisors for helping me to complete this work.

I am very thankful to Dr. Lemire for being such a wonderful mentor, for guiding me and providing me with in-depth feedback on various aspects of my work. Dr. Yan's guidance and mentoring has helped me improve on various aspects, such as ability to critically evaluate my own work, the ability to find insights into the big problems and hence finding a solution for those problems.

I am also thankful to all my friends during my master's program. Special thanks go to Min Chen and Jing Li, for the funny and wonderful moments we shared in the lab. I greatly thank Min Chen, PhD student, for helping me understand difficult aspects of my work and in writing my thesis.

Most importantly, I have immense gratitude for my family, for providing me with an amount of great support guidance and sacrifice during the harder times. My father and mother are the main source of my inspiration and motivation. My sisters, Karuna and Gayatri, are my first teachers in my life and their continuous guidance and support has helped me achieve my goals. I don't have enogh words to thank my parents and my sisters .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Cloud Computing

Cloud Computing has emerged as a ubiquitous paradigm where infrastructure and solutions are provided as a service. The cloud computing market is continuously growing. Analysts estimate the global cloud computing market is worth billions of dollars [24]. The major features of cloud computing are: pay-per-use, i.e., does not have an upfront cost and completely based on as usage; elasticity ability to scale up resources, and on demand capacity.

There are three cloud models which are popular. Infrastructure as a Service (IaaS), in which infrastructure (CPU, storage, network, etc.) is provided as a service. Amazon web services (http://aws.amazon.com/) and Rackspace (http://www.rackspace.com/)- are IaaS provider examples. Platform as a Service (PaaS) provides a custom platform to build cloud applications. Then, the application is deployed in the cloud and can be scaled. Microsoft Azure (http://www.microsoft.com/windowsazure/), Google App Engine (http://code.google.com/appengine/), Force.com (http://www.force.com/), and Facebook's developer platform (https://developers.facebook.com/) are examples of PaaS providers. In the Software as a Service (SaaS) model, cloud operators install the application in the cloud, and cloud users access the software from the cloud. Salesforce.com (http://www.salesforce.com/), Google Apps for Business and Enterprises (http://www.google.com/apps/intl/en/business/index.html), and Microsoft Dynamics CRM (http://crm.dynamics.com/en-us/home) are examples of SaaS providers.

1.2 Motivations and Challenges

Data is critical and central to every application. Big organizations are collecting data at different possible levels, resulting in massive and evergrowing data repositories. Therefore, database management systems (DBMS) become critical components for handling and manipulating data. Cloud computing provide a number of advantages for the deployment of data intensive applications. One is pricing (pay-as-use) and another is unlimited throughput by adding servers as load increases. Relational database management systems (RDBMS) are examples of online transaction processing (OLTP) database systems. Some of the important features of RDBMS are:

- Data Consistency: working concurrently and performing transactions, changing the system from one consistent state to another state.

- Rich Functionality: handling diverse applications using a relational data model and declarative query language.

- Durability: ensures persistency of data even in the case of power failure, system crashes.

Supporting transaction is a key feature of the RDBMS which leads to increased use of RDBMS. Although RDBMS is widely adopted and used by large organizations, RDMS is not suitable for cloud [38]. The reason is that scaling on demand and providing high availability is difficult in case of failures.

Consider a web application stack in Figure 1 [32]. The client's request passed through the load balancer to the machine which comprises of web server and the application server. The application server takes care of application logic (e.g., in C# with embedded SQL) and the web server handles HTTP requests. Queries and data storage are handled by the DB Server. Most applications start with a small set of servers. With an increase in the number of clients and requests except the database server, all the different layers in the stack can be scaled easily by adding more servers

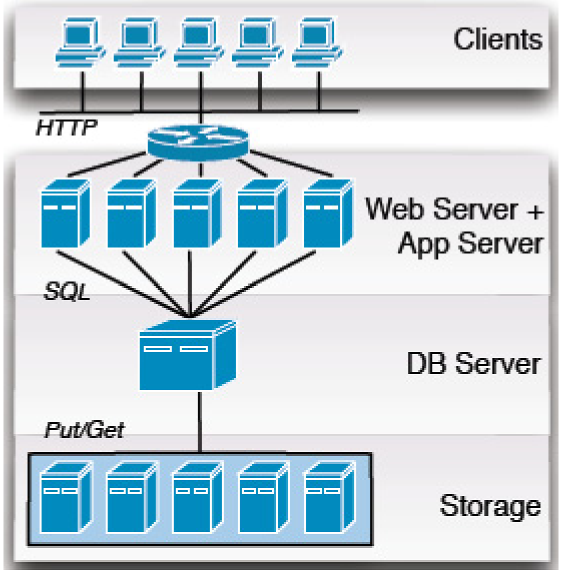


Figure 1: The classic software stack of web application [32]

to distribute the load across a larger number of servers. If the database server is overloaded, the only other option is to buy a bigger machine. A bigger machine that acts as the database server is costly. Therefore, the above architecture has limitations in terms of cost and scalability.

To scale, there are two approaches: scaling up and scaling out.

- Scaling up: This approach uses multiple processors that share the same memory space. They are typically used in enterprise infrastructures to scale up databases. However, scaling up is not preferred in cloud because the cost of hardware increases non-linearly, which is undesirable.

- Scale out: This approach involves adding multiple machines and is the preferred approach in the cloud. Scaling out reduces the overall system cost and provides pay–per-use pricing. RDBMS cannot be scaled out because of expensive distributed synchronization.

Many large organizations are looking forward to large scale operation and they depend upon a system called key-value pair, which is an architecture that is easy to scale out. Google's Big-table [9], Yahoo!'s PNUTS [12], Amazon's Dynamo [14] are examples. The data in these systems are distributed geographically in the form of key-value pairs. These key-value pairs can be scaled out to large servers and can provide low latency and high availability. The major problem with these systems is their lack of transaction guarantees. RDBMS provides strong consistency but is limited in its ability to scale out, while the form of key-value pairs provides scale out, but lacks support for transaction processing.

With the continuous growth of the Internet, databases need to store and process data efficiently. Moreover, databases need to provide high performance while reading and writing. Relational databases are challenged much, and appear inadequate in large scale operations and concurrent applications [22]. This leads to the development of NoSQL databases.

With many advantages of NoSQL databases such as wanting high performance during reading and writing, some enterprises seek to migrate their massive relational databases to NoSQL databases. The process of transferring data between storage types, formats, or computer systems is called data migration [44]. The technology used in developing the software becomes obsolete in a number of years due to the continuous evolution of IT [15]. A new version of a product requires updating of the old data model and moving the client data from their present data model to the data model used by the new version. Hence, the product requires data migration.

Thakar et al. [42] and Chanchary et al. [27] migrated a large relational database to cloud database. [42], shows that it is impossible to migrate even smaller data without changing the schema and the settings (e.g., the inability to migrate a spatial indexing library and several other user-defined functions and stored procedures). Both Thakar et al. [42] and Chanchary et al. [27] lack a migration model. Moreover, the data model of a relational database and NoSQL database are completely different.

Howard et al. [25] estimated that the data migration market would reach $906 million by 2012, as compared to $562 million in 2007. This shows a great demand in the requirement of data migration. Figure 2 presents the estimated data migration market [25] .

Banking, diversified financials and utilities are the sectors predicted to drive most of the spend on a global basis as shown in Table 1 [25].

In order to manage complex data migration processes, methods and models with different phases and activities need to be identified. Such methods should be applicable in practice and help software companies provide complete data migration

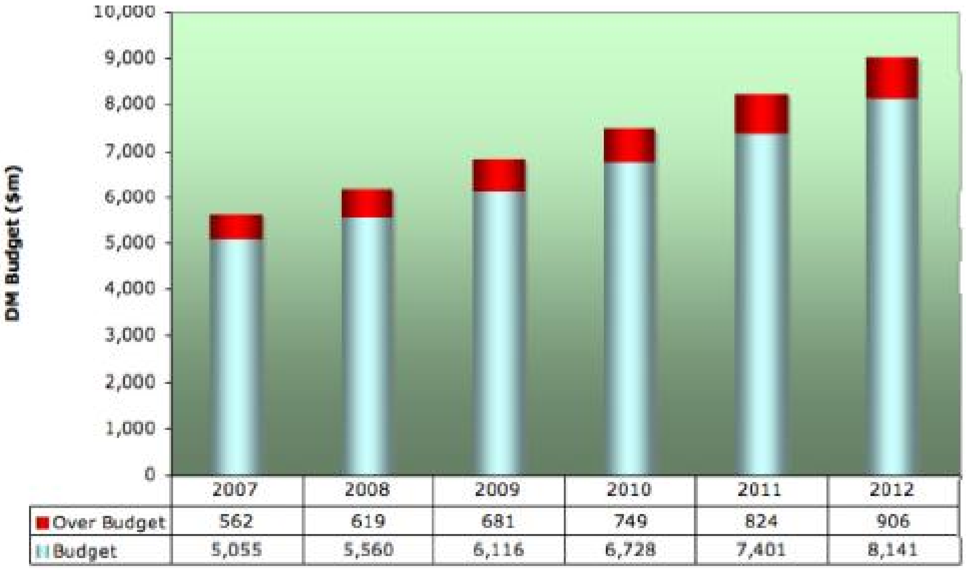


Figure 2: Global DM forecast and overruns 2007–2012 [25]

solutions.

Table 1: Global DM budget forecast leading 3 industry sectors in $m: 2007–2012 [25]

Industry Sector	2007	2008	2009	2010	2011	2012
Banking	783	862	948	1043	1147	1262
Diversified Financials	414	456	502	552	607	668
Utilities	296	325	358	394	433	476

1.3 Thesis Overview

The overarching goal of this thesis is to propose a model, methods, and paradigms to develop a system which migrates relational databases to cloud databases. This thesis proposes four diverse methods to migrate the relational databases to cloud databases. Each method is independent of the others.

- Type 1: complete relational database to one domain.

- Type 2: multiple tables to one domain.

- Type 3: a table to one domain.

- Type 4: normalization to denormalization and tables to domain

Cloud databases such as Amazon SimpleDB consists of domains. The domain corresponds to a database table. Type 1 Migration Method migrates all the tables in a relational database to a domain in the cloud database. Presently, the limit of a domain in Amazon SimpleDB is 10GB. Therefore, Type 1 can only migrate a relational database of size up to 10GB. Type 2 and Type 4 migration methods migrate tables in the relational databases to a domain where users frequently perform joins. Denormalization is defined as the process used to optimize the read performance of a database by adding redundant data or by grouping data [45]. Type 4 Migration Method denormalizes the tables and then migrates the data, whereas Type 2 migrates normalized database. Type 3 Migration Method migrates a table in a relational database to a domain in the cloud database. Each Migration Method migrates the whole relational database to cloud database. Join in relational databases combine records from two or more tables but cloud databases lack joins in order to have high performance and scalability. The thesis provides a way of handling joins in each approach. Finally, we propose an interface which generates code with respect to cloud API and helps in code re-factoring during application migration to the cloud.

In this thesis, we use Amazon SimpleDB as an example. The other NoSQL databases can be studied in the same way.

1.4 Contributions and Impact

We propose an automatic data migration model to migrate relational databases to the cloud. We present the mechanism for automatic data migration and discuss in detail each component and its significance. We present four different approaches of data migration from relational databases to cloud databases. We present the join functionality in cloud database. Finally we present an interface which generates code automatically with respect to cloud API and helps in code re-factoring during application migration to the cloud.

We have developed a prototype for all four techniques to demonstrate feasibility of ideas and techniques proposed for migrating a relational database to the cloud. A detailed analysis of each Migration Method allows enterprises to make a decision as to which method is more suitable for migration based on insights from this thesis.

1.5 Organization

In Chapter 2 we review the state of the art in scalable and distributed database system and study previous work. Chapter 3 presents the design principles and formulates the problem for data migration. We propose four migration strategies. Chapter 4 presents a data migration model and discusses each component of the model in detail. Chapter 5 presents the experimental evaluation of the model. Chapter 6 presents a performance evaluation of our model. Finally, Chapter 7 concludes the thesis, provides some future directions, and presents some open challenges.

Chapter 2

The State of the Art

This chapter presents the state of the art regarding the topic of data migration. First, we present the background of distributed systems. Then, we present the definition and characteristics of the data migration process. Different data migration methods are presented. The methods are categorized into technical and process based throughout the existing literature.

2.1 Background of Distributed Database Systems

Distributed DBMSs (DDBMS) such as R* [34] and SDD-1 [39] and Parallel DBMSs (PDBMS) such as Gamma [16] and Grace [19] are early attempts at providing scalable database systems. DDBMS and PDBMS are used for managing databases whose storage is distributed over a network of computers. The PDBMS allows updates but it is mainly used for analytics, while DDBMS are primarily designed and intensively used for updates.

DDBMS is limited due to the overhead of distributed transactions. Two reasons which limit transactions are:

- Higher response time: transactions are distributed across high latency networks, resulting in higher response times, spanning across multiple database servers.
- Limited availability of systems: guaranteeing transactions in the presence of failures limited the availability of these systems.

Apart from these reasons, there are some other concerns which prevent widely acceptance of distributed database systems. This leads to scaling up the DBMSs rather than using distributed database systems.

One approach to scale up DBMS is data sharding. Database sharding provides a method for scalability across independent servers, each with their own CPU, memory and disk. Examples of such systems are Oracle Real Application Clusters [8], Oracle Rdb (formerly DEC Rdb) [35], and IBM DB2 data sharing [30].

The basic concept behind sharding is taking a large database, and breaking it into a number of smaller databases across servers. Figure 3 presents database sharding.

Some advantages of data sharding are:

- Smaller databases are easier to manage.

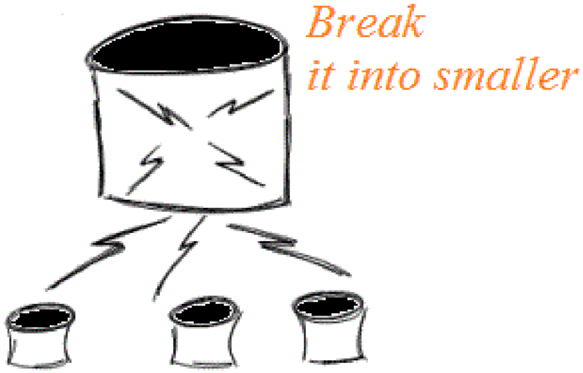


Figure 3: Data sharding takes large databases and breaks them down into smaller databases.

- Smaller databases are faster.

- Data sharding can reduce cost.

2.2 Cloud Data Management

Many Internet companies such as Google, Yahoo! and Amazon, face the challenges of handling millions of concurrent users. This leads to the development of key-value storages such as Google's Bigtable [10], Yahoo!'s PNUTS [11], and Amazon's Dynamo [14]. Key-value scales out to millions of commodity servers, replicating data across geography, and ensuring high availability of user data in case of failures.

2.2.1 Key-value Storage

Bigtable [10] is designed to scale up to petabytes of data across thousands of machines. Bigtable has achieved the following goals:

- Wide applicability
- Scalability
- High performance
- High availability

Bigtable is used by a large number of google products and projects, such as Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth.

A Bigtable cluster has a set of servers that serve the data. Each such server is known as a tablet server, which takes care of parts of the tables each called as a tablet. A tablet is semantically represented as a key range and physically represented as a set of Sorted Strings Tables (SSTables). A tablet forms a unit of distribution. Data is maintained into three dimensions: rows, columns, and timestamps. Data from the tables is persistently stored in the Google File System (GFS) [10] which is used to store data from the tables and provides scalable, consistent, fault-tolerant storage. Master and chubby cluster [4] are used to handle communication and synchronization between servers and meta-data management.

PNUTS [11] is designed by Yahoo! with a primary goal of providing read access among geographically distributed users. Data is organized in the form of attributes having record values. PNUTS performs explicit replication across different data centers. Yahoo! Message Broker (YMB) is responsible for handling replication of data. PNUTS uses centrally managed databases, processing the updates. Updates are first written in the master database and then replicated geographically. PNUTS uses asynchronous replication to ensure low latency updates.

Dynamo [14] has been the underlying storage technology to support the core services in Amazon's e-commerce business. It is able to scale even in busy shopping days to handle large amount of loads. Dynamo does not replicate data implicitly, instead it replicates explicitly. Updates can be done to any of the replicas. Data is partitioned and replicated using consistent hashing [31], and consistency is facilitated by object versioning [33]. Dynamo uses a quorum of servers to handle and serve write and read requests. ACID guarantees make poor availabilities, acknowledged by academic and industry [18]. Dynamo does not provide any isolation guarantees, and Dynamo only supports eventual replica consistency [43].

Amazon SimpleDB is a highly available and flexible non-relational data store that offloads the work of database administration. It is able to scale to handle a large amount of loads. SimpleDB architecture is characterized as a combination of partitioning and replication. Amazon SimpleDB is optimized to provide high availability and flexibility. Amazon SimpleDB automatically manages infrastructure provisioning, hardware and software maintenance, replication and indexing of data items, and performance tuning. SimpleDB does not provide any isolation guarantees and supports only eventual replica consistency [2].

2.2.2 Structure Selection

Although diverse databases that have key-value storages have one overarching goal, they have fundamental differences with respect to their design. Cooper et al. [13] describes the performance of different databases. We now discuss differences in design and the implications of the different databases.

2.2.2.1 Data Models

Key-value stores are distinguished by their data models. A Bigtable is a sparse, distributed, multidimensional sorted map. The map is indexed by a timestamp, column key, row key. Each value is an uninterpreted array of bytes [10]. Amazon SimpleDB stores key-value pairs in the Amazon Web Services (AWS). The entire table is represented as the domain. Each row is known as an item and every row has a unique identifier. PNUTS provides a flat row-like structure which resembles the relational model.

Key-value storage systems are different from RDMS when the failure of one component leads to system unavailability. The data manipulation in key-value pairs is done on the single node only. This leads to key-value pairs which can scale to a billion values with horizontal partition. The failure of data which is being served is limited to failure node only. Rest nodes can serve the request independently.

2.2.2.2 Data Replication and Fault Tolerance

Key-value storage systems replicate data in many commodity servers which ensure high availability and low latency. Yahoo! Message Broker (YMB) provides fault tolerance and replication in PNUTS. The response message is sent to the client once data is replicated. Fault tolerance is handled by Yahoo! Message Broker (YMB) using logs and guaranteed delivery, and replication of data. If one of the master nodes gets down or fails, another master node will be selected automatically.

Amazon SimpleDB uses asynchronous based replication. Amazon SimpleDB supports two types of read consistency:

- Eventually Consistent Reads (Default): An eventually consistent read might not reflect the results of a recently completed write. Repeating a read after a short time should return the updated data [2].

- Consistent Reads: A consistent read returns a result that reflects all writes that received a successful response prior to the read [2].

In case of failure, if one node gets down or fails, other nodes will handle requests.

In Bigtable data, replication is handled by the Google File System (GFS). GFS replicate all the data geographically, and provides strong and consistent replication storage. The write logs are saved in Bigtable and allow data to be recovered in case of server failure. Once a tablet server failure is detected by the master, the master will replace the failed server with another tablet server.

2.2.3 Data Distribution

All system data is distributed over the servers. Amazon SimpleDB supports hash partition. PNUTS deals with hash and range partition while Bigtable supports range partition.

2.3 Data Migration Methods and Characteristics

Data migration is studied widely but there are only a few publications which provide the meaning of data migration. Most authors define data migration as process of one aspect. This section will cover some definitions in the literature.

Matthes & Schulz [37] describes data migration as, "Tool-supported one-time process which aims at migrating formatted data from a source structure to a target data structure whereas both structures differ on conceptual and/or physical levels."

Drumm et al. [17] defines data migration as, "The task of transforming and integrating data originating from one or multiple legacy applications or databases into a new one."The data needs to be extracted from the source, transformed and loaded into the target during the data migration process.

Haller [20] defines data migration as the process of migrating data out of one schema to a new schema. The new schema and exiting schema can have completely different structures.

Data migration refers to two important aspects:

- the restructuring of data
- the actual transfer of data from source to target

The restructuring of data is discussed by Sockut & Goldberg [41] in early 1979. They view about restructuring as changing logical and physical structures of data. They described this process as restructuring and reformatting.

2.3.1 Relational-Cloud Mapping

Calil et al. [5] proposed SimpleSQL, a relational layer over Amazon SimpleDB, which implements relational-cloud mapping. However, the relational-cloud mapping does not migrate the relational databases to the cloud databases. SimpleSQL provides an access layer over Amazon SimpleDB that converts a SQL request to SimpleDB API and returns data in a relational format. Amazon SimpleDB is a famous document-oriented cloud database. Calil et al. [5] provides four traditional manipulation operations: INSERT, UPDATE, DELETE and SELECT. SimpleSQL supports complex queries. SimpleSQL performs the following steps to perform joins:

- Split: the command is split into simple SELECTs, i.e., SELECTs without JOIN;
- Access: each individual SELECT command is submitted to SimpleDB;
- Transform: the resulting set of each individual command is transformed to the relational schema;
- Join: the transformed tables are combined to generate the resulting table according to the join condition.

2.3.2 Data Migration Methods

In this section we study the literature of on data migration models. Idu [26] categorized migration models into two types:

- The Technical Model: This is technical based data migration. This method addresses how practically data migration can be done. It emphasizes on the development of technical solutions to migrate databases.
- The Process based Model: This migration model involves the management of

various phases of data migration. The migration model also manages the development of technical solutions.

Firstly, we present technical models followed by process based models .

2.3.2.1 Technical Models

1. **Schema Conversion**

   Hainaut et al. [23] define schema conversion as a process of extracting the source physical schema from the legacy application system of the underlying database source physical schema (SPS) and transforming it into a target physical schema for the target Data Management Systems (DMS). They provide two migration strategies at the database layer.

   - Physical schema conversion: it simulates the structure of the legacy system into the target DMS.
   - Conceptual schema conversion: the complete semantics of the legacy database are retrieved and represented in the conceptual schema (CS). Then schema refinement and data structure conceptualization is performed to develop a new database.

   Mapping is used to define the transformations that are required to migrate the data. Two types of mapping can be defined: Structural mapping which modifies the schema, and instance mapping which explains the transfer of source data into the target. Figure 4 represents both schemas [23].

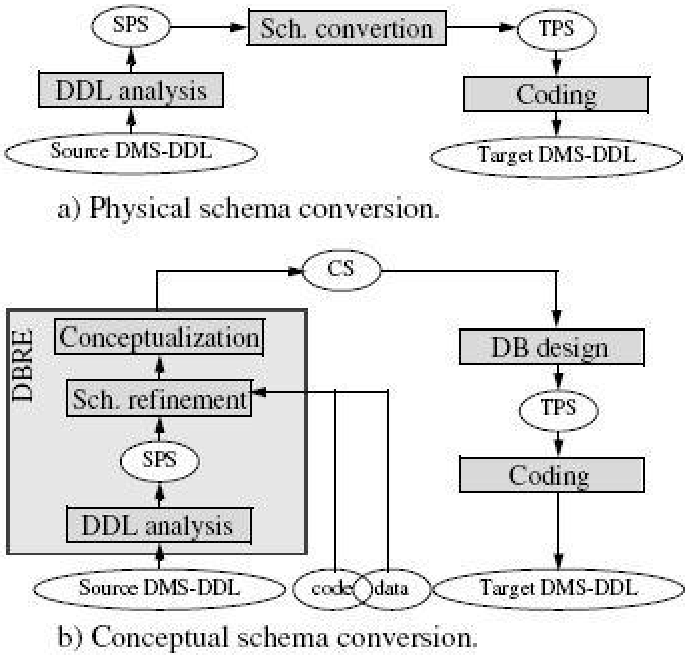


Figure 4: The two schema conversion strategies [23]

2. **Meta-Modeling Approach**

Jeusfeld and Johnen [29] present a meta-model method for relational database migration. According to this method, source and target DMS are well known. Mapping is created between the source and the target system. A meta model of the source data model and target model is developed as an interlink between the two and then migration is done.

Jahnke and Wadsack [28] present a two-phase process of migration. In the first phase, logical schema is gathered from the source database. The second phase involves converting a logical schema into conceptual schema. This conceptual schema forms a mapping between the source and the target DMS and thus executes the data migration.

Maatuk et al. [36] discuss three approaches to database conversion. The first approach is for handling data stored in RDBs through OO/XML interfaces. The second approach is connecting an existing RDBs to different database systems. The final approach is migrating RDB into the target system. Figure 5 presents their model of migration [1].

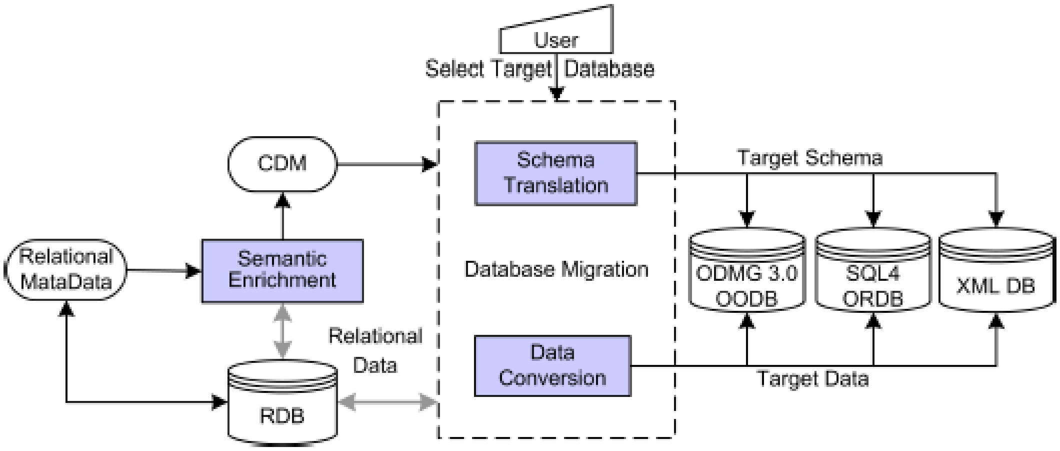


Figure 5: Data migration model [1]

3. **Extract Transform Load**

Haller [21] presents a migration model known as ETL. The extraction step extracts data and copies it to a different server. It filters data needed to be migrated. The transformation step involves the matching of the schema from the old system and the target system. If a schema differs, then restructuring of schema needs to be done. Once the transformation is completed, data is loaded into the target system. This completes the migration from the source to the destination. Figure 6 illustrates the ETL model [21].

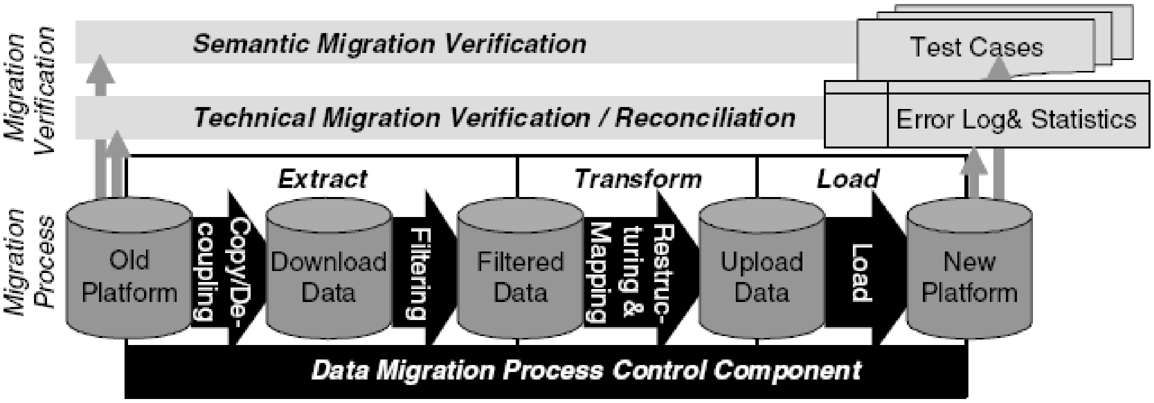


Figure 6: Generic migration architecture [21]

4. **Integrated Model**

Bordbar et al. [3] proposes an integrated model for data migration, closely related to the software development of the target system. This approach provides access to a relational database and performs data migration. The data migration is performed with the help of a model generator by providing the source and target models as inputs. The generator creates a model representation of the annotated UML model as an input to generic database adapter and upgrader generator. Generic database adapter acts as a database access layer and exploits the information in the model by generating necessary SQL queries. Upgrader generator generates upgrader program API by taking an old model representation, a new model representation, and an auxiliary property file. Finally, the upgrader program API is used for database cloning, schema evolution, and data migration.

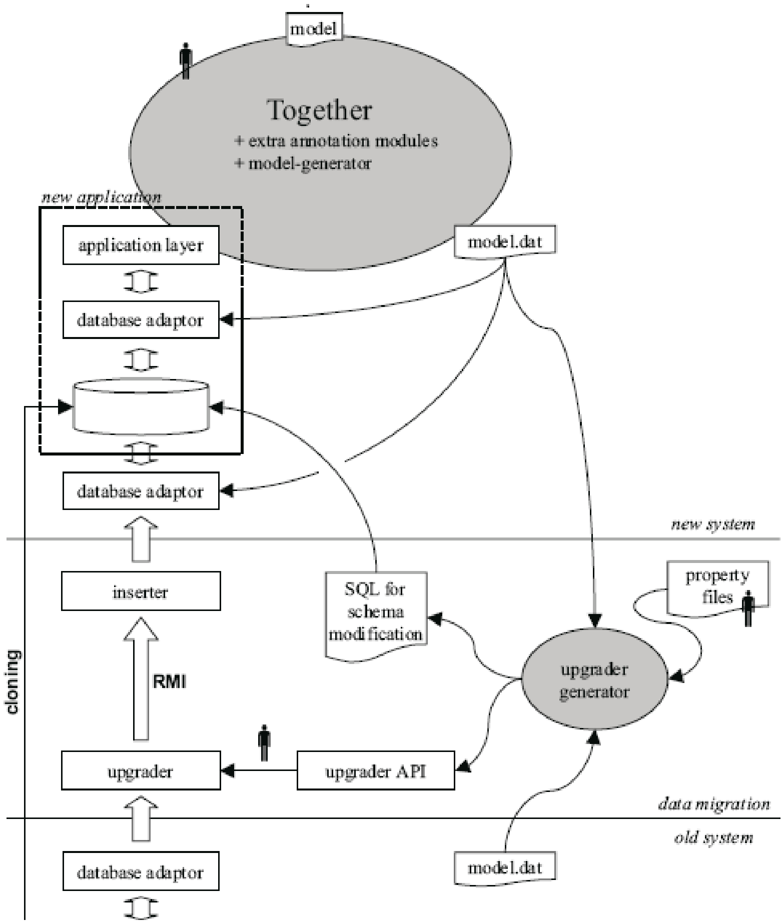


Figure 7: The Data migration approach [3]

The integrated model is presented in Figure 7 [3].

2.3.2.2 Process based Models

1. **Process model**

Matthes et al. [37] presents an iterative and incremental process model of fourteen phases as given in Figure 8 [37].

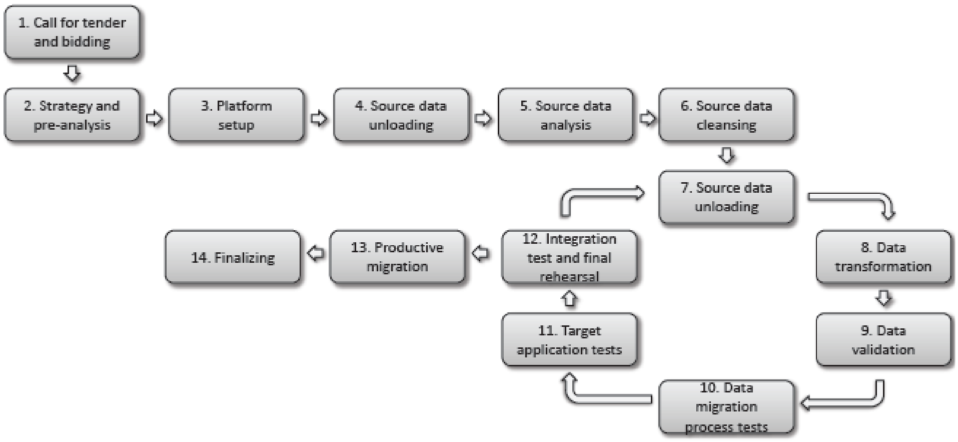


Figure 8: Process model [37].

The first phase, call for tender and bidding, indicates whether a migration project is an internal project or from outside sources. The second phase, strategy and pre-analysis, explains the project scope and migration road map. These phases determine what data is to be migrated and the business concept tables and attributes to be migrated. The technical and business view is depicted in Figure 9 [37].

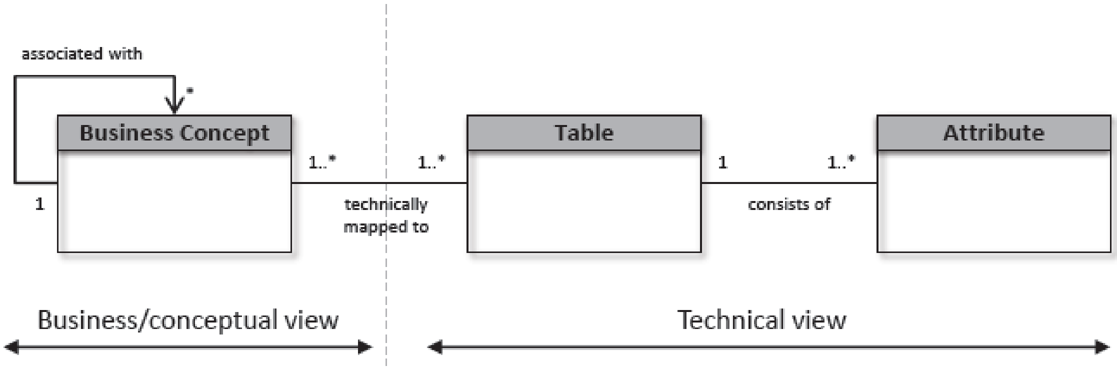


Figure 9: Business concept and technical relation [37]

Two scenarios are encountered between the source and the target systems. Firstly, source and target systems have different technical implementations. Secondly they may have a different business concept implementation. Both scenarios are depicted in Figure 10 [37].

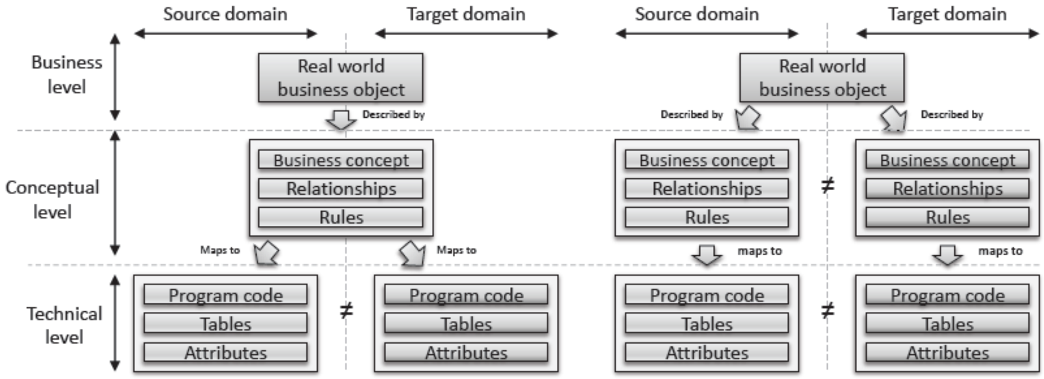


Figure 10: Migration scenarios on business, conceptual and technical levels [37]

The third phase is more technical to support the migration. Complex programs and source databases are used in this stage.

The next four phases consist of the implementation of the data migration program. Data unloading means extracting the relevant data from the source. Then it involves analysis of the structures of tables and attributes that need to be migrated. Source data cleansing means filtering and improving data quality. Finally, data transformation refers to continuously improving the rules and logic of the methods that migrate data from source to destination. The next six phases consist of testing overall data migration process. Testing is categorized into two types. Migration run testing which processes activities of data migration. This involves integration testing. The testing focus on the overall performance of the process. The second type is data validation. Validation refers to consistent testing, completeness and type correspondence. A report is prepared and used in the next iteration in order to solve any issues between the source and the target system.

The last three phases perform the final data migration. These stages make sure that testing has been achieved in previous phases before migration on the live data. At last, the target system is loaded with data.

2. **The Cyclical Process Model**

Russom [40] proposes a cyclic process for data migration. The development model is iterative and has five phases with a preliminary phase. The first phase is solution pre-design. This phase deals with requirement gathering, developing a project plan, timeline and data profiling. Solution design, the second phase refers to the splitting of data migration tasks based on their dependencies.

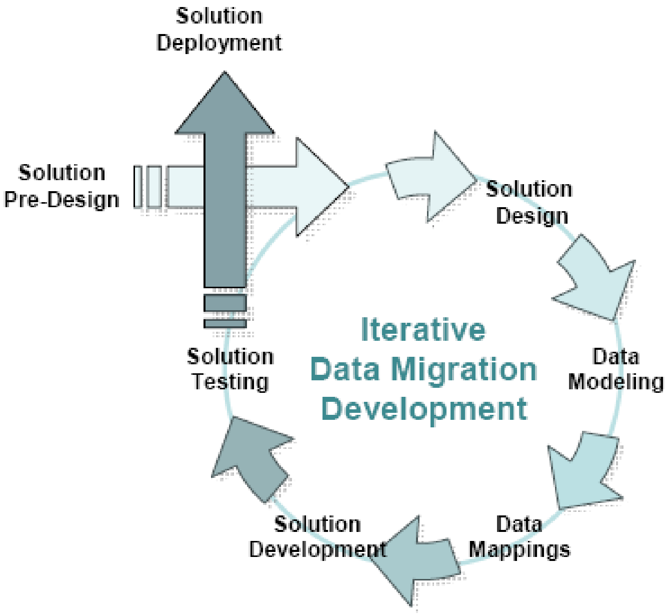


Figure 11: Data migration solution is a cyclical process [40]

.

The third phase, data modeling, refers to the building of the target database. The next phase, data mapping, maps the legacy system attributes to the target system attributes. In the fifth phase, solution development, uses mapping to migrate the data from source to target system using migration programs. The final stage is solution testing which involves an amount of data required to develop a solution and test it. The model is depicted in Figure 11 [40].

2.3.3 Data Migration Levels

Haller [20] provides different levels of methodologies for different migration projects. Table 2 [20] depicts the level from top to bottom.

Table 2: Data migration methodology levels [20]

Level	**Examples**
Project Management	Critical Path method, Expected Return of Investment of Projects
IT Project Management	Rational Unified Process
Data Migration Management and Controlling	Key Performance Indicators, Butterfly Approach
Real Life IT View	Deployment
Data migration Tools and Architecture	SAP solution, Butterfly Approach
Lab View	Implementation in PL/SQL

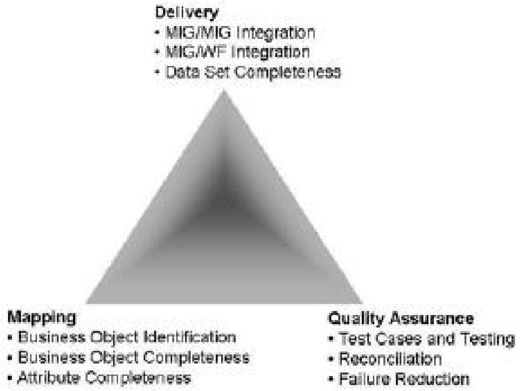


Figure 12: Dimension triangle [20]

.

Haller [20] groups all data migration tasks into three dimensions and forms the **migration triangle**. These three dimensions provide priorities during data migration. Figure 12 depicts dimension triangle [20]. The delivery dimension consists of three parts:

- Migration-Migration-Integration (MIG/MIG-integration) makes sure different mappings are running together.

- Migration-Customization-Integration (MIG/CUS-integration or MIG/WF-integration): it ensures that workflow, domain value tables and data migration work to integrate with each other.

- Data Set Completeness: it ensures migration completes on time and is tested.

Similarly, quality assurance has three tasks :

- Test Case and Testing: it involves running test cases for successful data migration.

- Reconciliation: it refers to make sure that all data is migrated successfully.

- Failure Reduction: it refers to fixing any issues involved in the above two cases.

The third dimension mapping also has three parts:

- Business Object Identification: it involves picking and identifying a business that is required to be migrated.

- Business Object Completeness: it migrates all business objects as identified in the above tasks.

- Attribute Completeness: it migrates complete information about business objects.

2.4 Summary

In this chapter, we studied the topic of data migration. We reviewed the background of distributed systems. Finally, we studied different technical and process based data migration methods.

Chapter 3

Data Migration Methods

Data migration methods play an important role in the architecture of data migration from a relational database to a cloud database, because it decides whether the migration procedure migrates data successfully. In this chapter, we propose four data migration methods based on three mapping strategies. From the many different famous cloud service providers are available,e.g., Amazon, Microsoft, Oracle, we choose Amazon service as our target system and Amazon SimpleDB as our target database. The analysis method developed in this thesis can be used for developing migration techniques on other NoSQL databases as well.

Firstly, we present building up principles of RDBMS and key-value storage. In section 3.2, we discuss problems faced by the relational database. Section 3.3 covers basic terminology, structure and characteristics of Amazon SimpleDB. In Section 3.4, we discuss characteristics of cloud databases. In Section 3.5.2, we give the definition of variables as a basis for the theoretical analysis of our data migration methods. In Section 3.5.3, we define three mapping strategies. Based on these mapping strategies, we propose four migration methods. The formulation of four migration methods as well as illustrative examples for these methods is given in Section 3.6. In Section 3.7, we present joins functionality in each migration method. In Section 3.10, we compare different migration methods and provide recommendation for when to use each migration method.

3.1 Design Principles

Although RDMBS and key-value pairs based cloud DBMS have different architectures, they are widely used for storing bulk data. Some of the common design principles are carried forward in designing DBMS for cloud platforms. The following are design principles:

- Incremental Scalability : These systems scale out to one storage node. Operations to a single node allow execution of the operations without the need for distributed synchronization. This makes other nodes work during failure without affecting others.
- Symmetry: Every node has the same responsibility. No node performs any

extra task as compared to another node.

- Synchronization: These systems limit the distributed synchronization and are used only when needed.

3.2 Problem Description

Relational databases support shared-everything architecture. Using a scale up approach, relational databases require a bigger and bigger server with more CPUs, more memory, and more disk storage to ensure that all the tables can support more concurrent users or store more data. However, big servers tend to be highly complex, proprietary, and disproportionately expensive, unlike the low-cost, commodity hardware. Therefore, it becomes important to store data efficiently and compute the massive amount of data, providing high scalability, and providing high performance and availability at low costs. This leads to the invention of NoSQL databases which scale out efficiently and store large amount of data in the form of key-value pairs. Key-value storages use a cluster of physical or virtual servers to support database operations and store data. To scale, additional servers are added to the cluster, database operations and the data are spread across the larger cluster. Some of the disadvantages of RDBMS are:

- Scale Out Transaction: RDBMS suffers due to the costly operation of scaling out and providing transactions at the same time.

- Administrative Overhead: it becomes difficult to manage large RDMBS installations with a large number of partitions. This is the biggest overhead of administration because doing a partition is difficult. To do this, it is required to make the data offline, complete the needed partition, and then bring it back online.

- Static partition: as partition and mapping are done to one node only, whenever a node fails or a load increases to capacity, the database needs to repartition which is highly complex and results in down time. This is highly inefficient in the banking system and online shopping websites.

- Modification: whenever partition is done, the application needs to modify again, in order to adjust.

Enterprises and legacy systems suffered from above drawbacks of RDBMS. These systems require migration of their relational databases onto the cloud databases and migration and integration of applications with cloud databases. Key-value pairs provide efficient solutions to overcome these issues.

In this thesis, we propose four **data migration methods** which will assist enterprises to migrate their relational databases efficiently and transparently to cloud databases.

We use a relational database for an online bookstore application to evaluate our Data Migration Methods and perform an experimental evaluation. The sample database consists of thirteen tables and sample data. We use **Amazon SimpleDB** as our cloud database and **Microsoft SQL Server** as our relational database for performing experiments. Figure 13 presents an online bookstore schema.

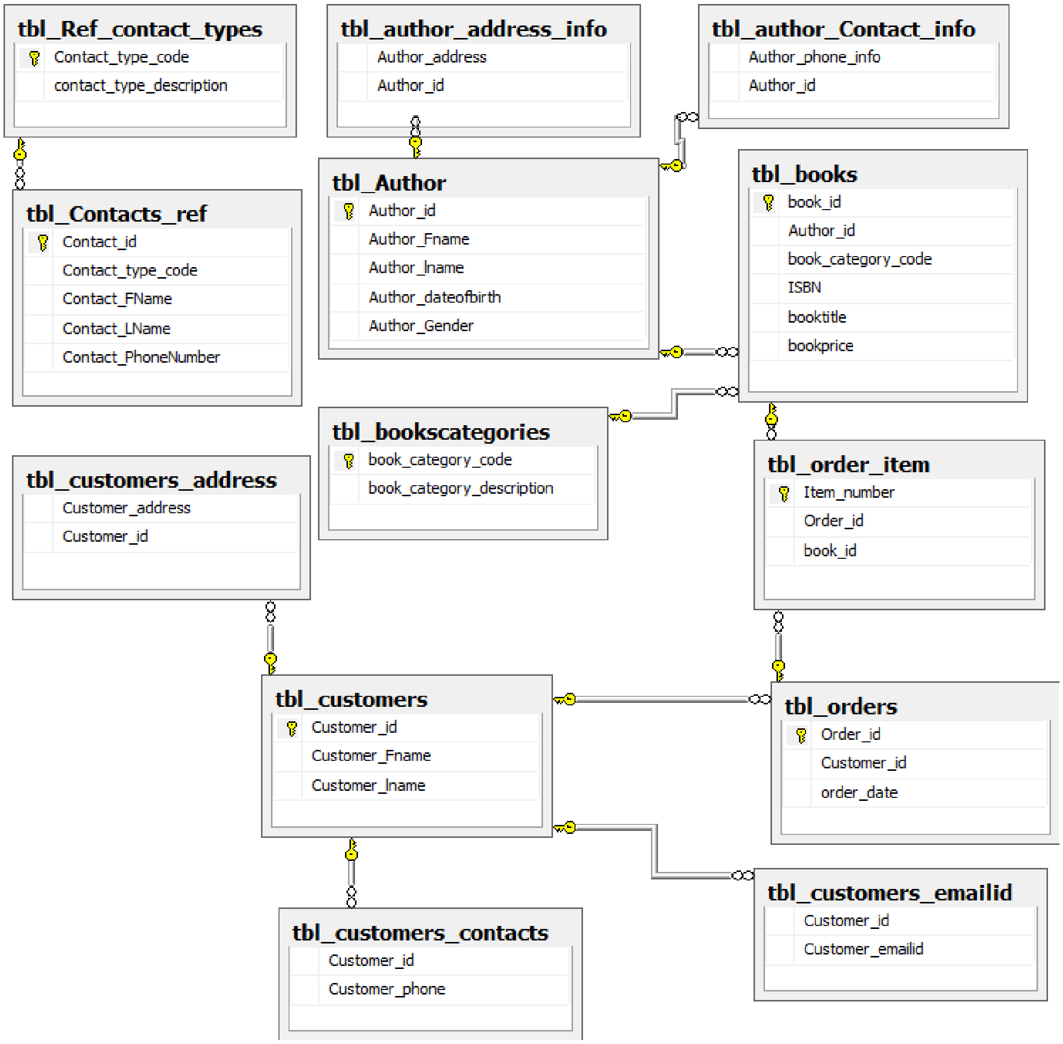
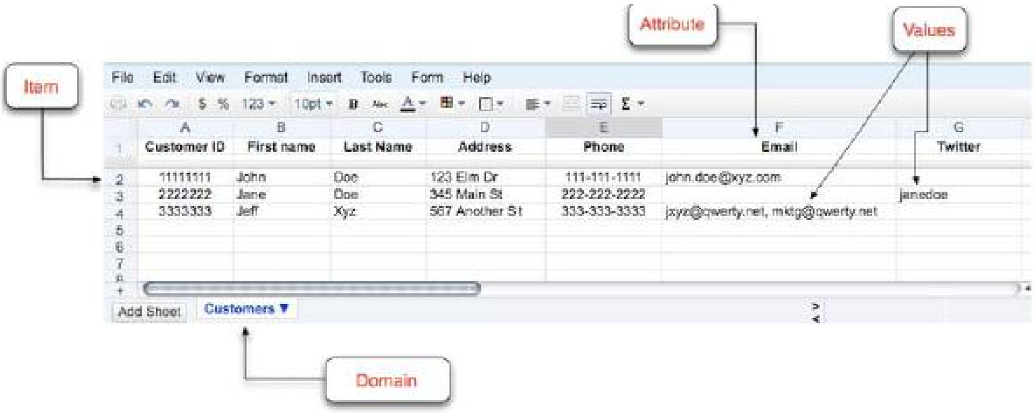


Figure 13: Relational database schema used in the experiments

3.3 Amazon SimpleDB

SimpleDB is a web service which provides structured data storage in the cloud, supported and backed by clusters of Amazon-managed database servers. The data is stored securely in the cloud and has no schema. The data is stored as key-value pairs. SimpleDB is an Amazon solution for handling data that follow document-oriented model [6]. It acts as a service and replication of data is done geographically depending upon the region selected during setup.

SimpleDB is composed of domains, items, attributes and values. SimpleDB looks like a spreadsheet that contains structured data. Figure 14 provides an overview of the SimpleDB structure [7].



	A	B	C	D	E	F	G
1	Customer ID	First name	Last Name	Address	Phone	Email	Twitter
2	11111111	John	Doe	123 Elm Dr	111-111-1111	john.doe@xyz.com	
3	2222222	Jane	Doe	345 Main St	222-222-2222		janedoe
4	3333333	Jeff	Xyz	567 Another St	333-333-3333	jxyz@qwerty.net, mktg@qwerty.net	
5							
6							
7							

Figure 14: SimpleDB customer structure [7]

- **Domain**: The entire customer table is represented as the domain customer in SimpleDB as shown in Figure 14. Each domain consists of a set of items. Data stored in a domain can be retrieved and modified by making a query against the domain. The user can have up to 250 domains and every domain can grow up to 10GB.

- **Item**: Every customer is identified by a unique Customer ID. Items are similar to rows in a database table. Each item is uniquely identified and contains data in the form of key-value attributes. The item name is similar to the primary key in a database table which identifies each item uniquely.

- **Attributes**: Attributes are synonymous with columns in database tables. Each customer attributes in Figure 14 is associated with value. A name, an address, and a phone number are three attributes of customer domain. Each item in SimpleDB can have an array of items. The customer can have multiple phone numbers and all phone numbers can be stored as multi-value attributes.

- **Values**: Every customer attribute is associated with a value. The customer's first name is an attribute and John is a value in the Customer domain.

Table 3 summarizes the equivalence relation between the Amazon SimpleDB and the database table.

Table 3: Relational database and SimpleDB equivalence

Relational Database	SimpleDB
Table	Domain
Row	Item
Column	Attribute
Value	Value(s)

Table 4 shows a comparison between SimpleDB and Relational Database.

Table 4: SimpleDB vs Relational Database

Relational Database	SimpleDB
Tables are organized in databases	No Databases, all domains are loose in AWS account
Schemas to define table structure	No predefined structure, variable attributes
Tables, records and column	Domain, items and attributes
Columns have only one value	Attributes have multiple values
Define indexes manually	All attributes are automatically indexed
Data is normalized	Data is not always normalized
joins are used to denormalize	No joins, either duplication or multiple queries
Transactions are used to guarantee consistency	Eventual consistency, consistent read, conditional put and conditional delete

In this section we have given an overview of Amazon SimpleDB. Also we have discussed a data model of Amazon SimpleDB which distinguishes it from relational database. In the next section we present characteristics of cloud databases which distinguish them from relational databases.

3.4 Characteristics of Cloud Databases

This section gives an overview of some distinguishing characteristics which separate cloud databases from relational databases.

- **No Normalization**: Cloud databases do not follow any normalization forms, and tend to be completely de-normalized. This provides a lot flexibility in the model without having normalization and enables the user to use multi-value attribute property or store multi-attribute data in the form of arrays.

- **No Joins**: Cloud databases do not support cross domain queries or cross table queries,sacrificing complex queries and join functionality as compared to a relational database. Cloud databases provide the ability to store multi values for an attribute or key and to some extent avoids the necessity of joins. The joins are avoided in the cloud databases in order to achieve high performance.

- **Schemaless**: Cloud databases does not support any schema. There is no need to maintain a schema and migrate the schema to a new version. Cloud databases stores data in the form of key-value pairs.

- **String Type**: SimpleDB stores all data as an UTF-8 string. As a result, SimpleDB creates an indexing on the data and retrieves data quickly. Other cloud databases support JSON types.

Table 5 presents some of the cloud databases that have the same data model and characteristics.

Table 5: Cloud databases Characteristics

Cloud Database	No Normalization	No Joins	Schemaless	Data type
Amazon SimpleDB	✓	✓	✓	String
MongoDB	✓	✓	✓	JSON types
CouchDB	✓	✓	✓	JSON types
Oracle NoSQL	✓	✓	✓	Binary, JSON types

3.5 Data Migration Methods

It is challenging to migrate the relational databases to the cloud databases as the structure and schema of the both databases is completely different.

3.5.1 Existing Migration Methods

The existing data migration models as explained in Chapter 2 are not sufficient to migrate the relational databases to the cloud databases:

- Thakar et al. [42], migrated a (large) science database to Amazon EC2 and Microsoft SQL Azure but lacks a migration strategy.

- Calil et al. [5] proposed a SimpleSQL, a relational layer over Amazon SimpleDB, which implements relational-cloud mapping. The relational-cloud mapping does not migrate relational databases to cloud databases. The following reasons make a SimpleSQL not sufficient for a cloud database:

  1. Migration Strategy: SimpleSQL does not provide a data migration strategy to migrate the relational database to the Amazon SimpleDB. Instead, the SimpleSQL provides an access layer over Amazon SimpleDB that converts the SQL request to the SimpleDB. Our Data Migration Methods provide different ways of migrating the relational databases to the cloud databases.

2. Sharding: SimpleSQL maps the data to one domain only in Amazon SimpleDB. It does not consider queries to other domains. Hence, once the domain crosses the limit of 10GB, query performances will decrease and it may take more time to return back the result. Hence, the user has limited sharding capability. Our system provides the sharding of data by migrating data in different domains depending upon the semantics. Hence, the database can scale out easily.

3. Application adaptation: SimpleSQL provides an access layer over Amazon SimpleDB that converts the SQL request to the SimpleDB. However, the SimpleSQL has limited capabilities. It provides an extra layer which increases query time to fetch data from the SimpleDB. Hence, it deteriorates the application performance. We propose an interface which will assist the developer to generate code automatically. This includes the basic usage of select, insert, delete and update queries. Therefore, applications can easily adapt to the migrated database without lacking in query performance.

4. Joins: SimpleSQL supports complex queries limited to one domain only. We provide a way of handling joins at the application level. Our different migration strategies support a different way of handling joins. Furthermore, we make use of the multi-value attribute property of Amazon SimpleDB and make the joins easier.

We propose four different data migration methods to migrate data from the relational databases to the cloud databases. Our data migration methods convert the data, preserve the schema of the relational databases, and overcome all the above challenges.

3.5.2 Definition of Variables

In this section, we give the definition of variables needed for the theoretical analysis of our data migration methods.

We denote a relational database by RDB. The list ST of variables associated with RDB is defined as follows:

- $ST(RDB) = \{T_k | 1 \leq k \leq n_{ST}\}$ is a set of tables in RDB where T_k is the k^{th} table of RDB and n_{ST} is the number of tables.

- $C(T_k) = \{c_j^{T_k} | 1 \leq j \leq n_C\}$ is a set of column names, corresponding to a set of columns of T_k and n_C is the number of column names, i.e.columns of T_k.

- $R(T_k) = \{r_i^{T_k} | 1 \leq i \leq n_R\}$ is a set of records of T_k where n_R is the number of records of T_k.

- $r_i^{T_k} = (v_{i1}^{T_k}, \ldots, v_{i|C(T_k)|}^{T_k})$ is a tuple of values $v_{ij}^{T_k}$ $(1 \leq j \leq |C(T_k)|)$ where $v_{ij}^{T_k}$ represents the value at the i^{th} record and the j^{th} column of table T_k. $|C(T_k)|$ is the number of columns of T_k.

- Q_l is the l^{th} request from a user to select a set of tables in RDB. The selected set of tables will be migrated to one domain in a cloud database.

- $QT(Q_l) = \{T_k | T_k \in ST(RDB)$, where T_k is requested by $Q_l\}$ is a set of tables requested by Q_l and some of the tables T_k have a primary-foreign key relationship.

- $QC(Q_l, T_k) = \{c_j^{T_k} | c_j^{T_k} \in C(T_k)$, where $c_j^{T_k}$ is requested by $Q_l\}$ is a set of column names of T_k requested by Q_l.

Accordingly, we denote a cloud NoSQL database by $NOSQLDB$. The list of variables associated with $NOSQLDB$ are defined as follows:

- $SD(NOSQLDB) = \{D_h | 1 \leq h \leq n_{SD}\}$ is a set of domains in $NOSQLDB$ where n_{SD} is the number of domains.

- $S \subseteq SD(NOSQLDB)$ is a subset of domains in $NOSQLDB$.

- $M(D_h) = \{m_p^h | 1 \leq p \leq n_h\}$ is a set of items and n_h is the number of items in D_h.

- $m_p^h = ((name_p^h, GUID_p^h), (c_{j1}^{T_{k_1}}, v_{i_1 j_1}^{T_{k_1}}), \ldots, (c_{jn}^{T_{kn}}, v_{i_n j_n}^{T_{kn}}))$ is the p^{th} item of D_h represented as a tuple of key-value pairs where

  - $(name_p^h, GUID_p^h)$ is the first key-value pair fixed for m_p^h to uniquely identify m_p^h, where $GUID_p^h$ is a globally unique identifier of item m_p^h and $name_p^h$ is the attribute name of $GUID_p^h$.

  - $(c_{j_h}^{T_{k_h}}, v_{i_h j_h}^{T_{k_h}})$ $(1 \leq h \leq n)$ is a key-value pair of a column name and a corresponding value in T_{k_h}.

- $\Pi_{attributelist}\sigma(D_h)$ are the set of key-value pairs in the attribute list of domain D_h such that :

  - The attribute list is a set of keys, i.e., $c_1, \ldots, c_m$ where $c_i(1 \leq i \leq m)$ is a key exists in an item of $M(D_h)$.
  - $\Pi_{c_1,\ldots,c_m}(D_h) = \{((c_1, v_1), \ldots, (c_m, v_m)) | \exists m_p^h \in M(D_h), (c_j, v_j)(1 \leq j \leq m)\}$ is a key value pair in m_p^h.
  - σ is a condition that is applied to test each item m_p^h in domain D_h.The condition is an expression built from $=, <, >, \leq, \geq, \wedge, \vee, \neg$. The conditional operator refines the $\Pi_{attributelist}\sigma(D_h)$ result.

- $CQ = \Pi_{attributelist}\sigma_\psi(D_h)$ is a cloud query where ψ is a “where ”notation in the query. The CQ must have at least one condition in ψ.

3.5.3 Mapping Strategies

In order to make data migration suitable for different sizes of relation databases and flexible with the users’ requirements, we propose three mapping strategies when mapping tables in a RDB to domains in a CDB.

Definition 1 ***Mapping Strategy One** (**MS1**) is a mapping strategy that maps a table $T_k \in ST(RDB)$ to a domain $D_h \in SD(NOSQLDB)$, denoted $T_k \overset{MS1}{\longmapsto} D_h$, such that:*

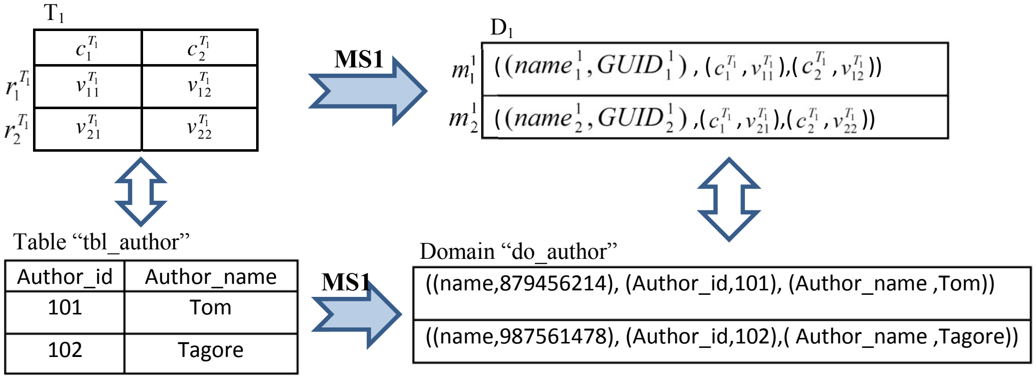
- *For all $r_i^{T_k} = (v_{i1}^{T_k}, \ldots, v_{i|C(T_k)|}^{T_k}) \in R(T_k)$, there exists*

  $$m_p^h = ((name_p^h, GUID_p^h), (c_1^{T_k}, v_{i1}^{T_k}), \ldots, (c_{|C(T_k)|}^{T_k}, v_{i|C(T_k)|}^{T_k}))$$

  where $|C(T_k)|$ is the number of columns of T_k.

- *The number of items of D_h equals the number of records of T_k, i.e.$|M(D_h)|$ $= |R(T_k)|$.*

Mapping Strategy One maps a table T_k in a RDB to a domain D_h in a $NOSQLDB$ with the strategy that a record $r_i^{T_k}$ of T_k corresponds to an item m_p^h of D_h. An example to explain Mapping Strategy One is given in Example 1.



T_1

	$c_1^{T_1}$	$c_2^{T_1}$
$r_1^{T_1}$	$v_{11}^{T_1}$	$v_{12}^{T_1}$
$r_2^{T_1}$	$v_{21}^{T_1}$	$v_{22}^{T_1}$

D_1

m_1^1	$((name_1^1, GUID_1^1), (c_1^{T_1}, v_{11}^{T_1}), (c_2^{T_1}, v_{12}^{T_1}))$
m_2^1	$((name_2^1, GUID_2^1), (c_1^{T_1}, v_{21}^{T_1}), (c_2^{T_1}, v_{22}^{T_1}))$

Table “tbl_author”

Author_id	Author_name
101	Tom
102	Tagore

Domain “do_author”

((name,879456214), (Author_id,101), (Author_name ,Tom))
((name,987561478), (Author_id,102),(Author_name ,Tagore))

Figure 15: Mapping Strategy One

Example 1 *Consider a relational database (RDB) " **Bookstore**"and a cloud database (NOSQLDB) " **Amazon SimpleDB**". "**Bookstore**"has a table "**tbl_author**"as shown in the bottom left of Figure 15. After applying Mapping Strategy One on table "**tbl_author**", domain "**do_author**"is created in "**Amazon SimpleDB**"as a mapping of table "**tbl_author**"and the data in table "**tbl_author**"is migrated to yhe domain "**do_author**"as shown in the bottom right of Figure 15. Also, correspondences between the definitions and tables/domains are presented in Figure 15.*

Definition 2 ***Mapping Strategy Two** (**MS2**) is a mapping strategy that maps a set of tables $QT(Q_l) \subseteq ST(RDB)$ to a domain $D_h \in SD(NOSQLDB)$, denoted $QT(Q_l) \overset{MS2}{\longmapsto} D_h$, such that:*

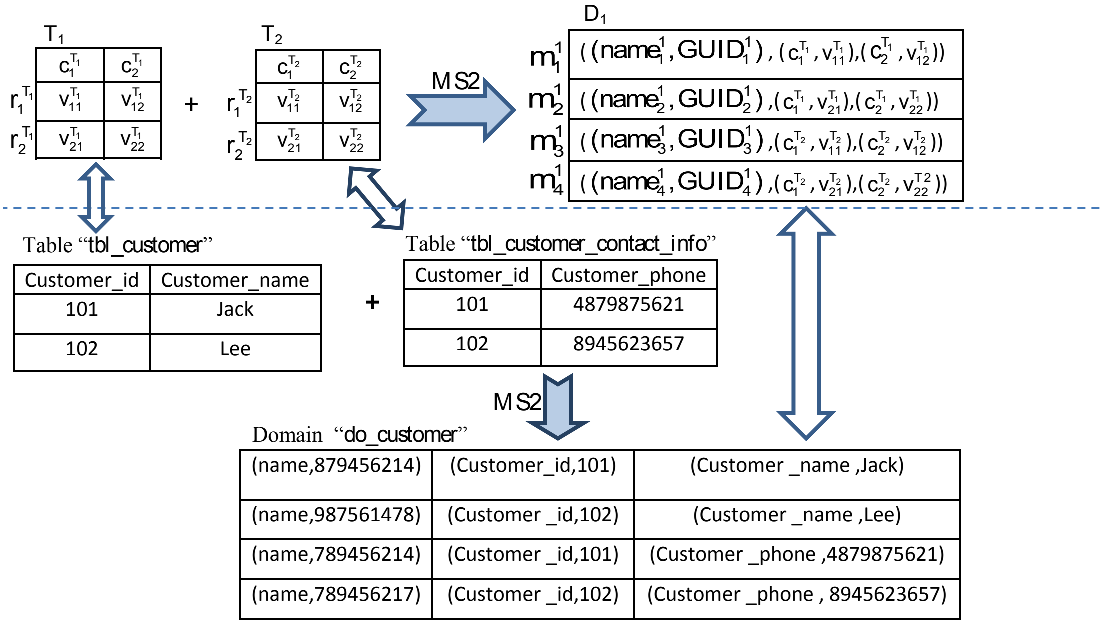
- *For all $T_k \in QT(Q_l)$ and $r_i^k = (v_{i1}^{T_k}, \ldots, v_{i|C(T_k)|}^{T_k}) \in R(T_k)$, there exists*

  $$m_p^h = ((name_p^h, GUID_p^h), (c_1^{T_k}, v_{i1}^{T_k}), \ldots, (c_{|C(T_k)|}^{T_k}, v_{i|C(T_k)|}^{T_k}))$$

  where $|C(T_k)|$ is the number of columns of T_k.

- *The number of items of D_h is the summation of records of tables in Q_l, i.e. $|M(D_h)| = \sum_{T_k \in QT(Q_l)} |R(T_k)|$.*

According to a user's request, Q_l, Mapping Strategy Two maps a set of tables $QT(Q_l)$ in a RDB to a domain D_h in a $NOSQLDB$ with the strategy that a record $r_i^{T_k}$ of $T_k \in QT(Q_l)$ corresponds to an item m_p^h of D_h. An example to explain Mapping Strategy Two is given in Example 2.



T_1

	$c_1^{T_1}$	$c_2^{T_1}$
$r_1^{T_1}$	$v_{11}^{T_1}$	$v_{12}^{T_1}$
$r_2^{T_1}$	$v_{21}^{T_1}$	$v_{22}^{T_1}$

T_2

	$c_1^{T_2}$	$c_2^{T_2}$
$r_1^{T_2}$	$v_{11}^{T_2}$	$v_{12}^{T_2}$
$r_2^{T_2}$	$v_{21}^{T_2}$	$v_{22}^{T_2}$

D_1

m_1^1	$((name_1^1, GUID_1^1), (c_1^{T_1}, v_{11}^{T_1}), (c_2^{T_1}, v_{12}^{T_1}))$
m_2^1	$((name_2^1, GUID_2^1), (c_1^{T_1}, v_{21}^{T_1}), (c_2^{T_1}, v_{22}^{T_1}))$
m_3^1	$((name_3^1, GUID_3^1), (c_1^{T_2}, v_{11}^{T_2}), (c_2^{T_2}, v_{12}^{T_2}))$
m_4^1	$((name_4^1, GUID_4^1), (c_1^{T_2}, v_{21}^{T_2}), (c_2^{T_2}, v_{22}^{T2}))$

Table “tbl_customer”

Customer_id	Customer_name
101	Jack
102	Lee

Table “tbl_customer_contact_info”

Customer_id	Customer_phone
101	4879875621
102	8945623657

Domain “do_customer”

(name,879456214)	(Customer_id,101)	(Customer _name ,Jack)
(name,987561478)	(Customer _id,102)	(Customer _name ,Lee)
(name,789456214)	(Customer _id,101)	(Customer _phone ,4879875621)
(name,789456217)	(Customer _id,102)	(Customer _phone , 8945623657)

Figure 16: Mapping Strategy Two

Example 2 *Consider a relational database (RDB) “ **Bookstore**”and a cloud database (NOSQLDB) “**Amazon SimpleDB**”. A user requests Q_1 has two tables “**tbl_customer**”and “**tbl_customer_contact_info**”to be migrated to “**Amazon SimpleDB**”together. We assume each customer has only one phone number in the table “**tbl_customer_contact_info**”. Applying Mapping Strategy Two, a new domain “**do_customer**”is created and the data in both tables “**tbl_customer**”and “**tbl_customer_contact_info**”are migrated into this domain. The mapping produced is presented in Figure 16.*

Mapping Strategy Two regards each table independently and migrates the data to a domain table by table. However, we can take advantage of the primary key and foreign key relations between tables in a relational database to make a mapping strategy. Hence, we define a virtual table that can be obtained from a set of requested tables. We call it a virtual table because this table does not exist in a relational database and is only generated for a temporary use.

Definition 3 *For given RDB and Q_l, a* joined table $T^{\bowtie}_{Q_l}$ *is generated and* $T^{\bowtie}_{Q_l}$ *satisfies:*

- *The column names of the joined table is the union of column names of all tables in $QT(Q_l)$, i.e.$C(T^{\bowtie}_{Q_l}) = \cup_{T_k \in QT(Q_l)} QC(Q_l, T_k)$.*
- *Suppose $T^*_{Q_l} \in QT(Q_l)$ is the table with a primary key in the request Q_l and the table has a foreign key relationship with other tables. For all $r_i^{T^*_{Q_l}} = (v_{i1}^{T^*_{Q_l}}, \ldots, v_{i|C(T^*_{Q_l})|}^{T^*_{Q_l}}) \in R(T^*_{Q_l})$, there exists*

$$r_i^{T^{\bowtie}_{Q_l}} = (v_{i1}^{T^{\bowtie}_{Q_l}}, \ldots, v_{in}^{T^{\bowtie}_{Q_l}}) \in R(T^{\bowtie}_{Q_l})$$

  where

  - *The number of columns equals the number of the union of columns of all tables in $QT(Q_l)$, i.e.$n = |C(T^{\bowtie}_{Q_l})|$*
  - *for $1 \leq j \leq n$,*

$$v_{ij}^{T_{Q_l}^{\bowtie}} = \begin{cases} 1 \\ \\ \\ 2 \end{cases}$$

$1\ v_{ij'}^{T^*_{Q_l}},\ \textit{if}\ c_j^{T_{Q_l}^{\bowtie}} = c_{j'}^{T^*_{Q_l}} \in C(T^*_{Q_l});$

$2\ \{v_{i'j'}^{T_k} | \forall i', v_{i'j''}^{T_k} = v_{ij'''}^{T^*_{Q_l}}\ \textit{where}\ c_{j''}^{T_k} = c_{j'''}^{T^*_{Q_l}} \wedge c_{j'''}^{T^*_{Q_l}}\ \textit{is}$

$\quad \textit{the primary key of}\ T^*_{Q_l}\},\ \textit{if}\ c_j^{T_{Q_l}^{\bowtie}} = v_{j'}^{T_k} \in C(T_k)$

$\quad \textit{where}\ T_k \in QT(Q_l) - \{T^*_{Q_l}\}.$

- *The number of records of the joined table* $T_{Q_l}^{\bowtie}$ *equals the number of records of table* $T^*_{Q_l}$, *i.e.*$|R(T_{Q_l}^{\bowtie})| = |R(T^*_{Q_l})|$.

Based on the definition of $T_{Q_l}^{\bowtie}$, we propose Mapping Strategy Three.

Definition 4 ***Mapping Strategy Three****(MS3) is a mapping strategy that maps a set of tables* $QT(Q_l) \subseteq ST(RDB)$ *to a domain* $D_h \in SD(NOSQLDB)$ *through a joined table* $T_{Q_l}^{\bowtie}$, *denoted* $QT(Q_l) \xmapsto[T_{Q_l}^{\bowtie}]{MS3} D_h$, *such that:*

- *For all* $r_i^k = (v_{i1}^{T_{Q_l}^{\bowtie}}, \ldots, v_{i|C(T_{Q_l}^{\bowtie})|}^{T_{Q_l}^{\bowtie}}) \in R(T_{Q_l}^{\bowtie})$, *there exists*

  $$m_p^h = ((name_p^h, GUID_p^h), (c_1^{T_{Q_l}^{\bowtie}}, v_{i1}^{T_{Q_l}^{\bowtie}}), \ldots, (c_{|C(T_{Q_l}^{\bowtie})|}^{T_{Q_l}^{\bowtie}}, v_{i|C(T_{Q_l}^{\bowtie})|}^{T_{Q_l}^{\bowtie}}))$$

  where $|C(T_{Q_l}^{\bowtie})|$ *is the number of columns of* $T_{Q_l}^{\bowtie}$.

- *The number of items of D_h equals the number of records of $T^{\bowtie}_{Q_l}$, i.e.$|M(D_h)|$ $= |R(T^{\bowtie}_{Q_l})|$.*

The Mapping Strategy Three makes use of the primary key and foreign key relations between tables to map a set of tables requested by the users in a domain. An example to explain Mapping Strategy Three is given in Example 3.

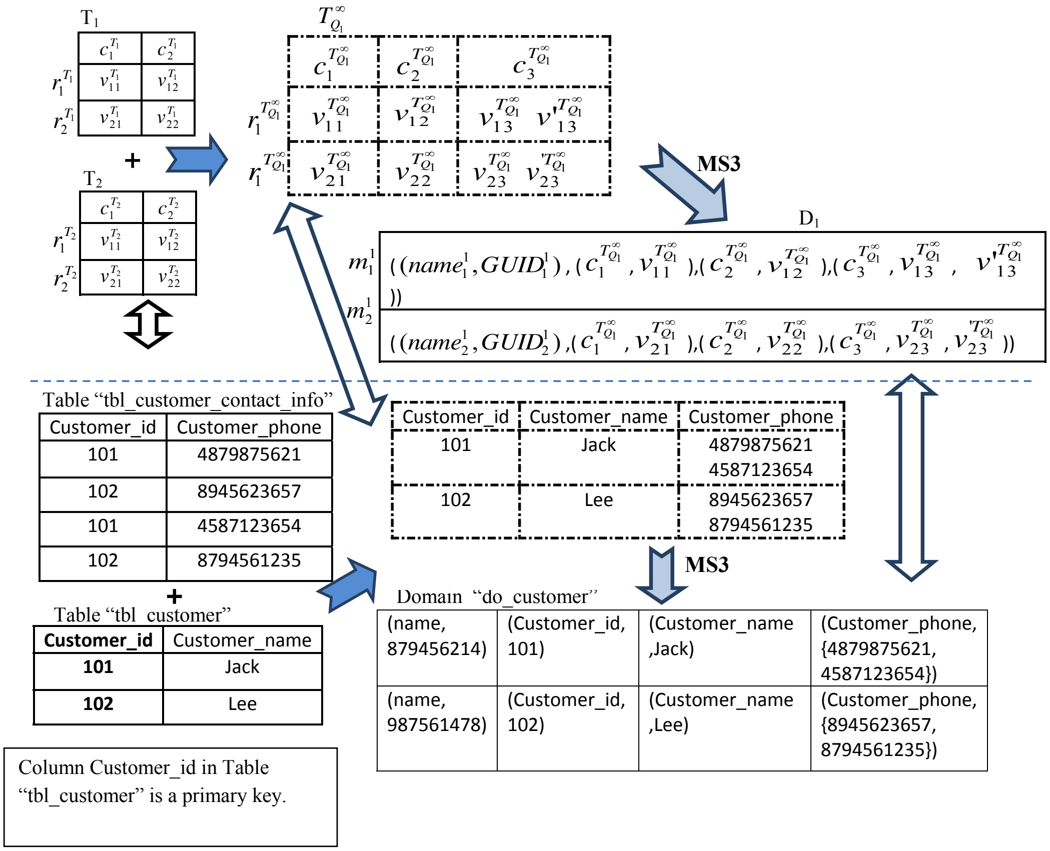

T_1
$c_1^{T_1}$ $c_2^{T_1}$
$r_1^{T_1}$ $v_{11}^{T_1}$ $v_{12}^{T_1}$
$r_2^{T_1}$ $v_{21}^{T_1}$ $v_{22}^{T_1}$
+
T_2
$c_1^{T_2}$ $c_2^{T_2}$
$r_1^{T_2}$ $v_{11}^{T_2}$ $v_{12}^{T_2}$
$r_2^{T_2}$ $v_{21}^{T_2}$ $v_{22}^{T_2}$
$T^{\infty}_{Q_1}$
$c_1^{T^{\infty}_{Q_1}}$ $c_2^{T^{\infty}_{Q_1}}$ $c_3^{T^{\infty}_{Q_1}}$
$r_1^{T^{\infty}_{Q_1}}$ $v_{11}^{T^{\infty}_{Q_1}}$ $v_{12}^{T^{\infty}_{Q_1}}$ $v_{13}^{T^{\infty}_{Q_1}}$ $v'^{T^{\infty}_{Q_1}}_{13}$
$r_1^{T^{\infty}_{Q_1}}$ $v_{21}^{T^{\infty}_{Q_1}}$ $v_{22}^{T^{\infty}_{Q_1}}$ $v_{23}^{T^{\infty}_{Q_1}}$ $v_{23}^{'T^{\infty}_{Q_1}}$
MS3
D_1
m_1^1 $((name_1^1, GUID_1^1), (c_1^{T^{\infty}_{Q_1}}, v_{11}^{T^{\infty}_{Q_1}}), (c_2^{T^{\infty}_{Q_1}}, v_{12}^{T^{\infty}_{Q_1}}), (c_3^{T^{\infty}_{Q_1}}, v_{13}^{T^{\infty}_{Q_1}}, v'^{T^{\infty}_{Q_1}}_{13}))$
m_2^1 $((name_2^1, GUID_2^1), (c_1^{T^{\infty}_{Q_1}}, v_{21}^{T^{\infty}_{Q_1}}), (c_2^{T^{\infty}_{Q_1}}, v_{22}^{T^{\infty}_{Q_1}}), (c_3^{T^{\infty}_{Q_1}}, v_{23}^{T^{\infty}_{Q_1}}, v_{23}^{'T^{\infty}_{Q_1}}))$
MS3

Table “tbl_customer_contact_info”

Customer_id	Customer_phone
101	4879875621
102	8945623657
101	4587123654
102	8794561235

+

Table “tbl_customer”

Customer_id	Customer_name
101	Jack
102	Lee

Customer_id	Customer_name	Customer_phone
101	Jack	4879875621 4587123654
102	Lee	8945623657 8794561235

Domain “do_customer”

(name, 879456214)	(Customer_id, 101)	(Customer_name ,Jack)	(Customer_phone, {4879875621, 4587123654})
(name, 987561478)	(Customer_id, 102)	(Customer_name ,Lee)	(Customer_phone, {8945623657, 8794561235})

Column Customer_id in Table “tbl_customer” is a primary key.

Figure 17: Mapping Strategy Three

Example 3 *Consider a relational database (RDB) " **Bookstore**"and a cloud database (NOSQLDB) "**Amazon SimpleDB**". A user requests Q_1 two tables "**tbl_customer**"and "**tbl_customer_contact_info**"to be migrated to "**Amazon SimpleDB**"together. Since the primary key "**Customer_id**"of table "**tbl_customer**"is the foreign key of table "**tbl_customer_contact_info**", a temporary table $T^{\bowtie}_{Q_1}$ is generated. Applying Mapping Strategy Three, a new domain "**do_customer**"is created and the data in both tables "**tbl_customer**"and "**tbl_customer_contact_info**"are migrated into this domain. The mapping produce is presented in Figure 17.*

3.6 Migration Methods

Making use of three mapping strategies, i.e.$MS1$, $MS2$, and $MS3$, we propose four migration methods, i.e.Type 1 Migration, Type 2 Migration, Type 3 Migration, and Type 4 Migration. Figure 18 presents the relationship between mapping strategies and data migration methods. Type 1 migration method uses migration strategy 2 (MS2). Type 2 migration method uses migration strategy 2 (MS2) and migration strategy 1 (MS1). Type 3 migration method uses migration strategy 1 (MS1). Type 4 migration method uses migration strategy 3 (MS3) and migration strategy 1 (MS1). The four data migration methods are proposed depending upon our analysis of the characteristics of both the cloud database and the relational database. There could be more possible data migration methods beyond our consideration for migrating relational databases to cloud databases. For example, in Type 4 Migration Method, we do not denormalize and migrate data of tables whose primary key is composed of more than one column, which cause more redundancies and make data migration complicated.

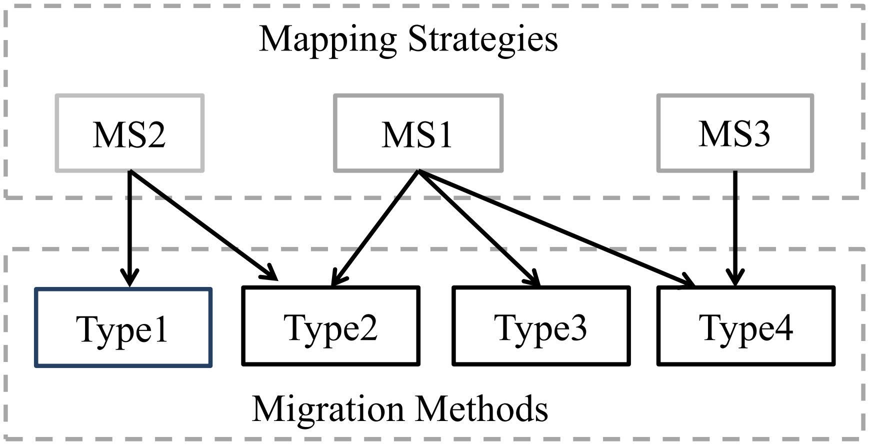


Figure 18: Migration Strategies - Migration Methods Relationship

3.6.1 Type 1 Migration

Making use of Mapping Strategy Two, Type 1 Migration migrates all the tables in a relational database to a domain in a cloud database. Example 4 gives an example of Type 1 Migration.

Definition 5 *For given RDB and NOSQLDB, **Type 1 Migration** migrates all the data from a relational database (RDB) to one domain D_h in a cloud database (NOSQLDB), i.e.*$ST(RDB) \overset{MS2}{\longmapsto} D_h$*, such that there is only one domain in a cloud* *database (NOSQLDB), i.e.*$|SD(NOSQLDB)| = 1$.

Example 4 *Consider a relational database (RDB) "**Bookstore**"and a cloud database (NOSQLDB) "**Amazon SimpleDB**". " **Bookstore**"has only two tables "**tbl_customer**"and "**tbl_customer_contact_info**"as shown in Example 2. A user requests this two tables can be migrated to "**Amazon SimpleDB**"together, i.e.* $QC(Q_1) = ST(RDB) = \{$ *"**tbl_customer**", "**tbl_customer_contact_info**"*$\}$. *After applying Type 1 Migration to "**Bookstore**", "**Amazon SimpleDB**"contains one and only one domain "**do_customer**"as show in Figure 16.*

3.6.2 Type 2 Migration

Making use of both Mapping Strategy One and Mapping Strategy Two, Type 2 Migration migrates the set of tables included in one request to one domain.The other single table and their data is migrated to another single domain in cloud database. Example 5 gives an example of Type 2 Migration.

Definition 6 *For given RDB, NOSQLDB and a set of requests* $\{Q_1, \ldots, Q_n\}$, ***Type 2 Migration** migrates the data from a relational database (RDB) to a cloud database (NOSQLDB) such that*

- *For all* $Q_l \in \{Q_1, \ldots, Q_n\}$ *there exists one and only one domain* $D_h \in SD(NOSQLDB)$ *such that* $QT(Q_l) \overset{MS2}{\longmapsto} D_h$.

- *For all $T_k \in ST(RDB) - \sum_{l=1}^{n} QT(Q_l)$ there exists one and only one domain $D_h \in SD(NOSQLDB)$ such that $T_k \xmapsto{MS1} D_h$.*

- *The number of domains in $NOSQLDB$ equals the number of requests plus the number of tables in RDB that are not requested by any request, i.e. $|SD(NOSQLDB)| = n + |ST(RDB) - \sum_{l=1}^{n} QT(Q_l)|$.*

Example 5 *Consider a relational database (RDB) "**Bookstore**"and a cloud database (NOSQLDB) "**Amazon SimpleDB**". "**Bookstore**"has three tables two of which, "**tbl_customer**"and "**tbl_customer_contact_info**", are as shown in Figure 16 and the left table "**tbl_author**"is shown in Figure 15. As shown in Figure 16, "**tbl_customer**"and "**tbl_customer_contact_info**"are migrated into domain "**do_customer**", denoted D_1, using Mapping Strategy Two. As shown in Figure 15, table "**tbl_author**"is migrated into domain "**do_author**", denoted D_2, using Mapping Strategy One. After migration, "**Amazon SimpleDB**"only contains two domains D_1 and D_2.*

3.6.3 Type 3 Migration

Making use of Mapping Strategy One, Type 3 Migration migrates each table in a relational database to a domain in a cloud database. Example 6 gives an example of Type 3 Migration.

Definition 7 *For given RDB and NOSQLDB,* ***Type 3 Migration*** *migrates the data from a relational database (RDB) to a cloud database(NOSQLDB) such that*

- *For all* $T_k \in ST(RDB)$*, there exists one and only one* $D_h \in SD(NOSQLDB)$ *such that* $T_k \xmapsto{MS1} D_h$.
- *The number of domains in NOSQLDB equals the number of tables in RDB, i.e.*
  $|SD(NOSQLDB)| = |ST(RDB)|$.

Example 6 *Consider a relational database (RDB) "**Bookstore**"and a cloud database (NOSQLDB) "**Amazon SimpleDB**". "**Bookstore**"has only one table "**tbl_author**"as shown in Figure 15. Applying Mapping Strategy One, table "**tbl_author**"is migrated into domain "**do_author**". Figure 15 also shows the migration process. After migration, "**Amazon SimpleDB**"has only one domain "**do_author**".*

3.6.4 Type 4 Migration

Denormalization is defined as the process to optimize the read performance of a database by adding redundant data or by grouping data [45]. This migration first denormalizes the relational database based on the semantics. Once semantics are provided by the user, it will automatically denormalize the database and then migrate the data to cloud database. The denormalization process is important and will utilize the multi-value attribute property of Amazon SimpleDB and migrate data in the form of arrays. The multi-value attributes will provide join functionality in SimpleDB. Normally, denormalized tables have a primary-foreign key relationship. Example 7 gives an example to explain Type 4 migration process.

Definition 8 *For given RDB, NOSQLDB and a set of requests* $\{Q_1, \ldots, Q_n\}$*,* ***Type 4 Migration*** *migrates the data from a relational database (RDB) to a cloud database (NOSQLDB) such that*

- *For all* $Q_l \in \{Q_1, \ldots, Q_n\}$ *there exists one and only one domain* $D_h \in SD(NOSQLDB)$ *such that* $QT(Q_l) \xmapsto[T^{\bowtie}_{Q_l}]{MS3} D_h$.
- *For all* $T_k \in ST(RDB) - \sum_{l=1}^{n} QT(Q_l)$ *there exists one and only one domain* $D_h \in SD(NOSQLDB)$ *such that* $T_k \xmapsto{MS1} D_h$.
- *The number of domains in NOSQLDB equals the number of requests plus the number of tables in RDB that are not requested by any request, i.e.* $|SD(NOSQLDB)| = n + |ST(RDB) - \sum_{l=1}^{n} QT(Q_l)|$.

Example 7 *Consider a relational database (RDB) "**Bookstore**"and a cloud database (NOSQLDB) "**Amazon SimpleDB**". "**Bookstore**"has three tables two of which, "**tbl_customer**"and "**tbl_customer_contact_info**", are shown in Figure 17 and the left table "**tbl_author**"is shown in Figure 15. As shown in Figure 17,* *“**tbl_customer**”and “**tbl_customer_contact_info**”are migrated into domain “**do_customer**”, denoted D_1, using Mapping Strategy Three. As shown in Figure 15, table “**tbl_author**”is migrated into domain “**do_author**”, denoted D_2, using Mapping Strategy One. After migration, “**Amazon SimpleDB**”only contains two domains D_1 and D_2.*

3.7 Joins

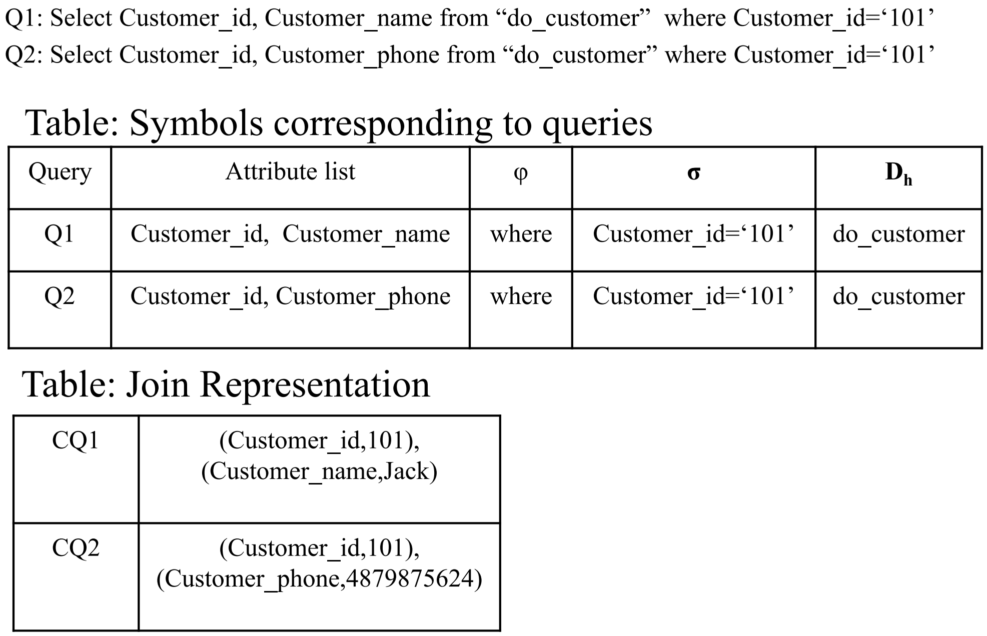
Cloud databases does not support the concept of joins. In this section, we formalize the joins in the four migration methods.

3.7.1 Join in Type 1 and Type 2

Definition 9 *After migrating data from RDB to NOSQLDB using Type 1 or Type 2 migration methods. $J^{\bowtie}(S) = \{CQ_l | l >= 2\}$ is a join consisting of a set of cloud queries to S in NOSQLDB where S contains only one domain, i.e., $|S| = 1$.*

In Type 1 or Type 2 migration methods, a join is performed to one domain only because the data of the tables where the user is likely to make joins is migrated to one domain only. In the Type 1 migration method, whole database data is migrated to one domain. In the Type 2 migration method, the data of the tables where user like to make joins is migrated to one domain. The join is performed by making simultaneous queries to a domain in the cloud database.

As shown in Example 5, the domain "**do_customer**"is formed by merging tables "**tbl_customer**"and "**tbl_customer_contact_info**". The join to the domain "**do_customer**"is shown in Figure 19. The "**Q1**"and "**Q2**"are two queries to the cloud and their symbol representation is presented in "**Table: Symbols corresponding to queries**"in Figure 19. "**Table: Join Representation"** in Figure 19 presents the join representation and result of the queries. Developers can exploit data at the application level.



Q1: Select Customer_id, Customer_name from "do_customer" where Customer_id='101'
Q2: Select Customer_id, Customer_phone from "do_customer" where Customer_id='101'

Table: Symbols corresponding to queries

Query	Attribute list	φ	**σ**	**D_h**
Q1	Customer_id, Customer_name	where	Customer_id='101'	do_customer
Q2	Customer_id, Customer_phone	where	Customer_id='101'	do_customer

Table: Join Representation

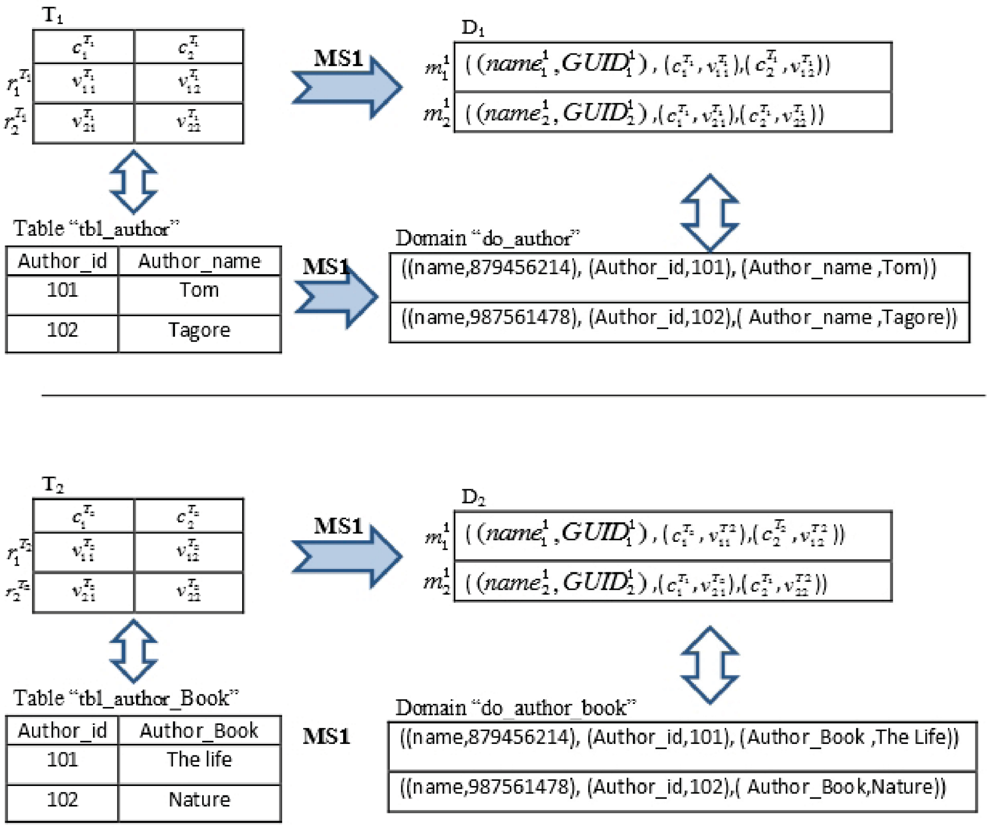
CQ1	(Customer_id,101), (Customer_name,Jack)
CQ2	(Customer_id,101), (Customer_phone,4879875624)

Figure 19: Join in Type1 or Type 2

3.7.2 Join in Type 3

Definition 10 *After migrating data from RDB to NOSQLDB using Type 3 migration method.* $J^{\bowtie}(S) = \{CQ_l | l >= 2\}$ *is a join consisting of a set of cloud queries to* S *in NOSQLDB where* S *contains more than one domain, i.e.,* $|S| > 1$.

Figure 20 presents the two tables "**tbl_author**"and "**tbl_author_book**". Making use of Mapping Strategy One, Type 3 Migration migrates each table in a relational database to a domain in a cloud database. The table "**tbl_author**"corresponds to a domain "**do_author**"and the table "**tbl_author_book**"corresponds to a domain "**do_author_book**".



	$c_1^{T_1}$	$c_2^{T_1}$
$r_1^{T_1}$	$v_{11}^{T_1}$	$v_{12}^{T_1}$
$r_2^{T_1}$	$v_{21}^{T_1}$	$v_{22}^{T_1}$

m_1^1	$((name_1^1, GUID_1^1), (c_1^{T_1}, v_{11}^{T_1}), (c_2^{T_1}, v_{12}^{T_1}))$
m_2^1	$((name_2^1, GUID_2^1), (c_1^{T_1}, v_{21}^{T_1}), (c_2^{T_1}, v_{22}^{T_1}))$

Author_id	Author_name
101	Tom
102	Tagore

((name,879456214), (Author_id,101), (Author_name ,Tom))
((name,987561478), (Author_id,102),(Author_name ,Tagore))

	$c_1^{T_2}$	$c_2^{T_2}$
$r_1^{T_2}$	$v_{11}^{T_2}$	$v_{12}^{T_2}$
$r_2^{T_2}$	$v_{21}^{T_2}$	$v_{22}^{T_2}$

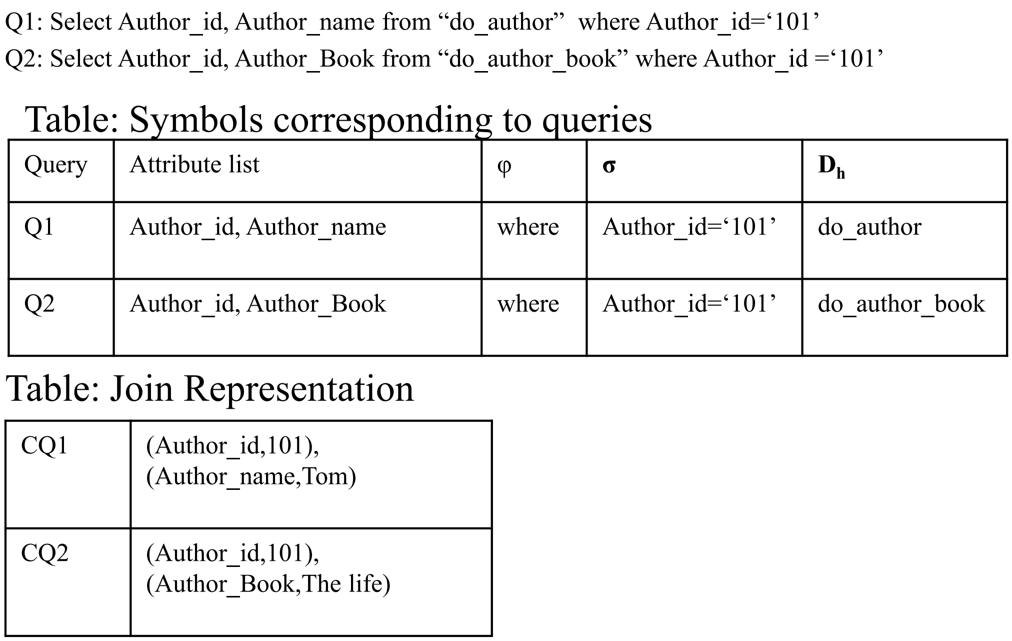
m_1^1	$((name_1^1, GUID_1^1), (c_1^{T_2}, v_{11}^{T2}), (c_2^{T_2}, v_{12}^{T2}))$
m_2^1	$((name_2^1, GUID_2^1), (c_1^{T_1}, v_{21}^{T_2}), (c_2^{T_1}, v_{22}^{T2}))$

Author_id	Author_Book
101	The life
102	Nature

((name,879456214), (Author_id,101), (Author_Book ,The Life))
((name,987561478), (Author_id,102),(Author_Book,Nature))

Figure 20: Type 1 Data Migration

In Type 3, the join is performed by querying the multiple domain and exploiting the data at the application level. As shown in Figure 21, the join is performed by query the domain "**do_author**"and "**do_author_book**". Figure 21 illustrates the join functionality. The "**Q1**"and "**Q2**"are two queries to the cloud and their symbol representation is presented in "**Table: Symbols corresponding to queries**". "**Table: Join Representation**"in Figure 21 presents the join representation and the result of the queries. Developers can exploit data at the application level.



Q1: Select Author_id, Author_name from “do_author” where Author_id=‘101’
Q2: Select Author_id, Author_Book from “do_author_book” where Author_id =‘101’

Table: Symbols corresponding to queries

Query	Attribute list	φ	**σ**	**D_h**
Q1	Author_id, Author_name	where	Author_id=‘101’	do_author
Q2	Author_id, Author_Book	where	Author_id=‘101’	do_author_book

Table: Join Representation

CQ1	(Author_id,101), (Author_name,Tom)
CQ2	(Author_id,101), (Author_Book,The life)

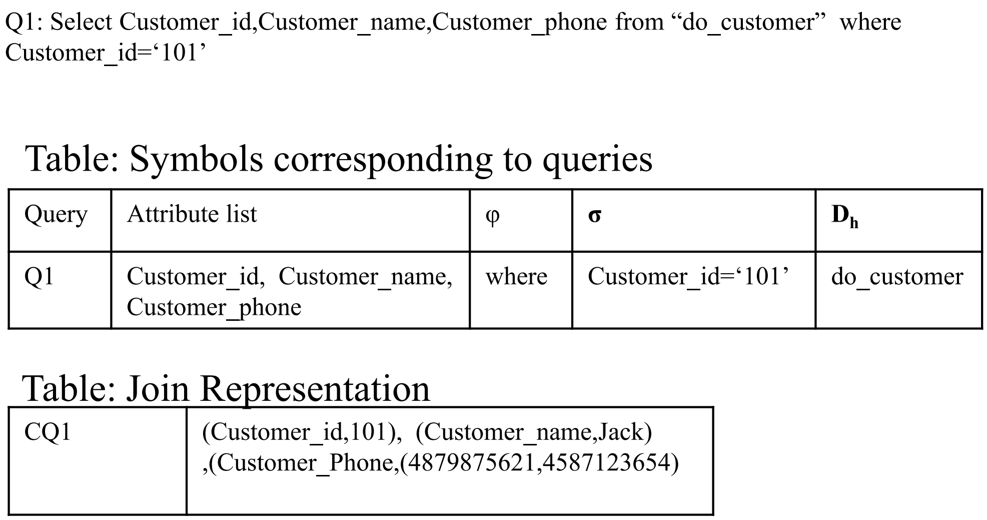
Figure 21: Join in Type 3

3.7.3 Join in Type 4

Definition 11 *After migrating data from RDB to NOSQLDB using Type 4 migration method.* $J^{\bowtie}(S) = \{CQ_l | l >= 1\}$ *is a join consisting of a cloud query or set of queries to* S *in NOSQLDB where* S *contains only one domain, i.e.,* $|S| = 1$.

In the Type 4 migration method, the join is performed to one domain only because the data of the tables where the user is likely to make joins is migrated to one domain only. The data is in the denormalized form.

As shown in Example 7, the domain “**do_customer**”is formed by merging tables “**tbl_customer**”and “**tbl_customer_contact_info**”. The join to the domain “**do_customer**”is shown in Figure 22.



Q1: Select Customer_id,Customer_name,Customer_phone from "do_customer" where Customer_id='101'

Table: Symbols corresponding to queries

Query	Attribute list	φ	**σ**	**D_h**
Q1	Customer_id, Customer_name, Customer_phone	where	Customer_id='101'	do_customer

Table: Join Representation

CQ1	(Customer_id,101), (Customer_name,Jack) ,(Customer_Phone,(4879875621,4587123654)

Figure 22: Join in Type 4

The "**Q1**"is a query to the cloud. The symbol representation is presented in "**Table: Symbols corresponding to queries**". "**Table: Join Representation**"in Figure 22 presents the join representation and results of the query. Developers can exploit the data at the application level.

3.8 Sharding

As the data size increases, a single domain may not be sufficient to store the data nor provide an adequate read and write throughput. In this case, we need to shard the data. Sharding is the process of storing data records across multiple domains in SimpleDB. Sharding is important for scaling up in SimpleDB. Every domain has limited throughput, so it becomes important to spread the data across multiple domains.

The Type 1 Migration method migrates a relational database to a domain in the cloud database. Therefore, the cloud database can have only one domain. As a result, the Type 1 Migration Method does not support sharding. The read and write throughput in the Type 1 Migration method may decrease as domain reaches its capacity.

The Type 2, Type 3 and Type 4 Migration methods migrate the tables of a relational database to the domains in the cloud database. Therefore, the Type 2, Type 3 and Type 4 Migration Methods support sharding. Hence, it is easy to scale domains in the Type 2, Type 3 and Type 4 Migration Methods. So, rather than storing all the data in one SimpleDB domain, we recommend splitting domains up into smaller number of chunks to increase the throughput and get a high performance. Sharding data across domains decreases query time, but it increases box usage. Box usage is directly related to the cost of querying the data. Hence, performance-improving measures result in a higher bill.

3.9 Redundancy

Data redundancy is the superfluity of data. In computer data storage system, data redundancy is a common issue. The major disadvantages of data redundancy are:

- Increases the size of the database

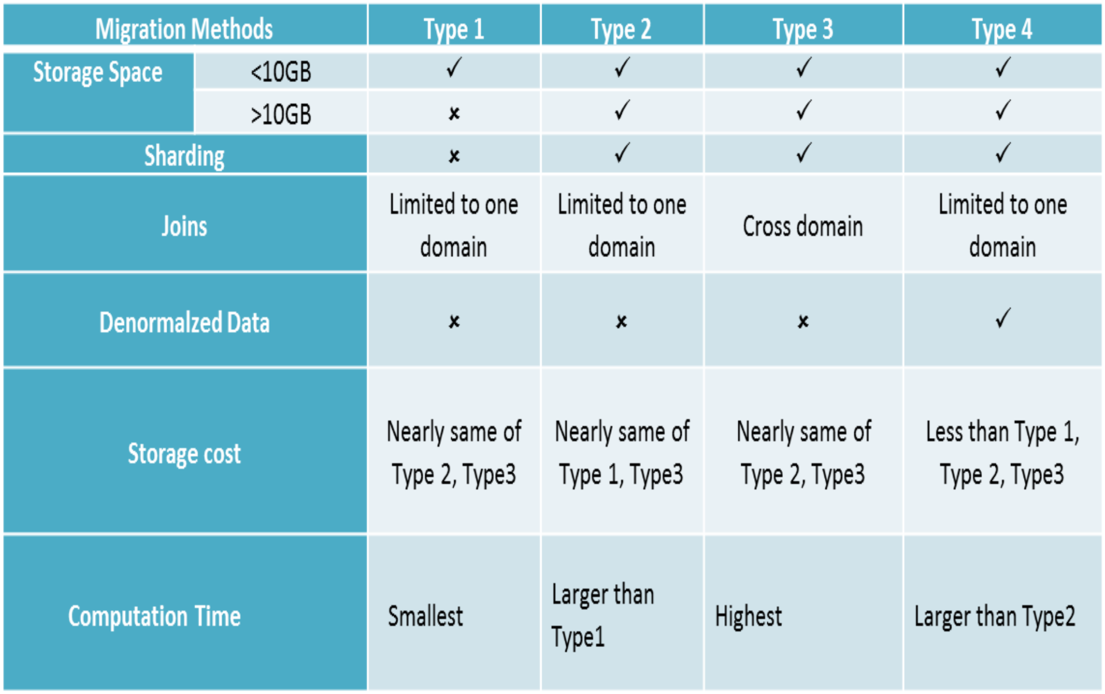
- Causes data inconsistency
- Decreases efficiency of the database
- May cause data corruption

Type 1 Migration migrates all the tables in a relational database to a domain in the cloud database. The tables migrated in a domain may have redundant data. The Type 2 Migration migrates tables which have primary-foreign key relationship to a domain in the cloud database. Therefore, the Type 2 Migration method stores data in the redundant form. This redundant form causes inconsistency. The query to a domain in the Type 1 Migration method and the Type 2 Migration method may bring back the result in a redundant form. As a result, the query time for a domain in the Type 1 Migration method and the Type 2 Migration method may increase. This leads to a decrease in the efficiency of the domain.

The Type 3 Migration migrates each table in a relational database to a domain in a cloud database. Each table in a relational database is in its normalized form. Therefore, normalized data is migrated to a domain in the cloud database. Hence, Type 3 Migration method stores consistent data and is not redundant. In the Type 4 Migration method, the relational database is migrated in denormalized form. The denomalization process uses the multi-value attributes property of a cloud database and then migrates data in the form of arrays. Therefore, storing the data in the multi-value attribute form decreases the redundancy and stores the data in the consistent form. As a result, the query time for a domain in the Type 4 Migration decreases.

3.10 Comparison of Migration Methods

we compare our different migration methods and provide recommendations is for when to use these methods. Figure 23 presents a comparison of different migration methods.



Migration Methods		Type 1	Type 2	Type 3	Type 4
Storage Space	<10GB	✓	✓	✓	✓
	>10GB	×	✓	✓	✓
Sharding		×	✓	✓	✓
Joins		Limited to one domain	Limited to one domain	Cross domain	Limited to one domain
Denormalzed Data		×	×	×	✓
Storage cost		Nearly same of Type 2, Type3	Nearly same of Type 1, Type3	Nearly same of Type 2, Type3	Less than Type 1, Type 2, Type3
Computation Time		Smallest	Larger than Type1	Highest	Larger than Type2

Figure 23: Comparison of Migration Methods

A software developer or enterprises should use the Type 1 Migration Method when the database size should not increase more than 10GB. A domain in Amazon SimpleDB can only store data up to 10GB. However, if the size of the relational databases increases more than 10GB, we recommend the software developer or enterprises must use the Type 2, Type 3 or Type 4 Migration methods. Moreover, the Type 2, Type 3 and Type 4 Migration methods support sharding. Therefore, the Type 2, Type 3 or Type 4 Migration methods provide high scalability. The Type 2 and Type 4 Migration limit the joins to only one domain. We recommend the use of the Type 4 Migration method over Type 2 Migration method because Type 4 Migration Method denormalizes the data. By denormalizing, the Type 4 Migration method saves the renting cost and makes the joins easier as compared to the Type 2 Migration method. The Type 2 Migration is useful when there is no need for data denormalization. The Type 3 Migration Method is useful when the user needs the same structure of the data in the cloud database as is needed in the relational database.

Chapter 4

Data Migration Model

To realize our data migration methods, we propose an Automatic Data Migration model. This model uses four migration methods as an important component of our migration system to migrate data in four different ways.

In Section 4.1, we propose an Automatic Data Migration model. The system Migrates the relational database into the NoSQL cloud database. In Section 4.2, we discuss each component of our model in detail. In Section 4.3, we present objectives of the migration model. Finally, in Section 4.4, we present the limitations of our model.

4.1 Architecture of the Migration Model

Figure 24 presents the architecture of the automatic data migration model. The model migrates the relational databases to the cloud databases.

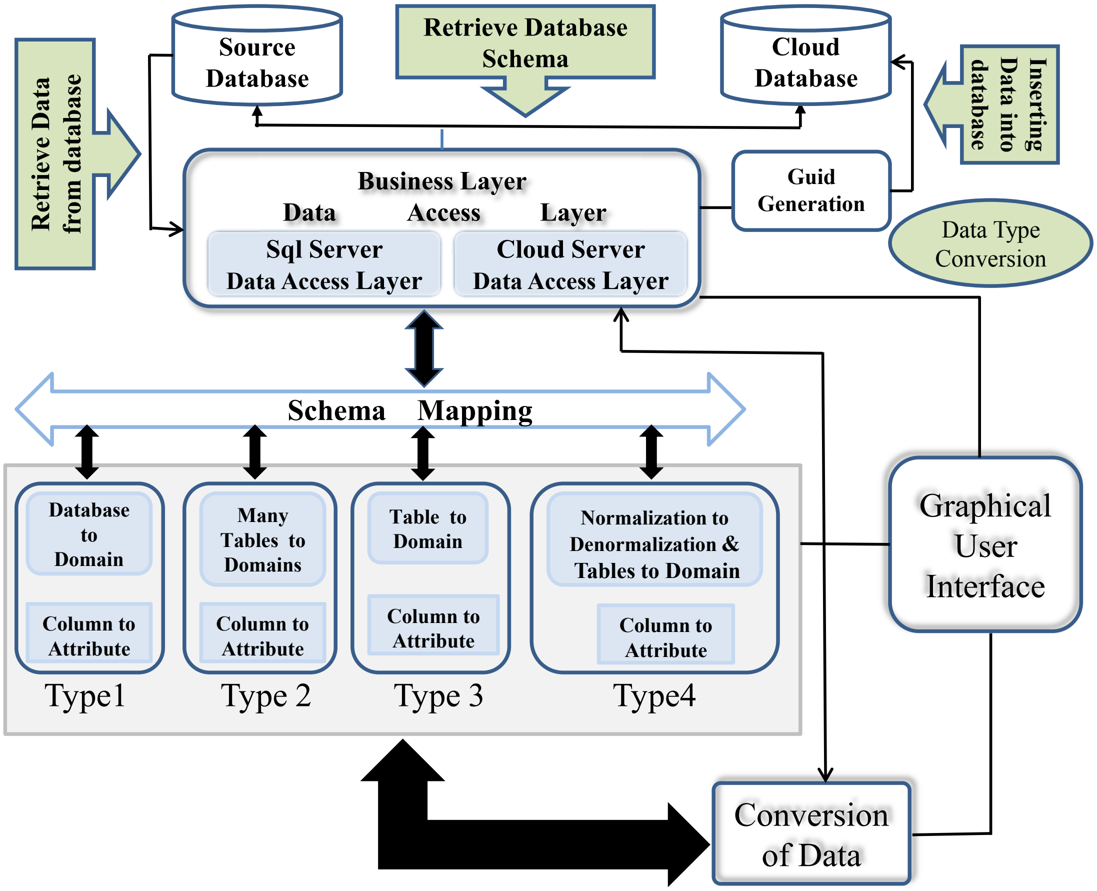


Figure 24: Relational-Cloud Migration

The model maps the data between the source system and the target system. The model migrates data from one system to another system automatically whenever it is required. Only semantic knowledge is required at some point.

The model first loads all available relational databases. Depending upon the selected database, it loads all the tables, and schema, and provides the relationship between the tables. The relationship refers to the primary-foreign key relationship. The user can see all the available tables in the present database. The user has complete control over the data migration process through the user interface. The model automatically fetches the underlined schema, and data from the source database and migrate this data to target system (aka Amazon SimpleDB.).

The model is composed of many specialized modules that interact with each other during migration and maintain a continuous flow of data. The main modules of the model are:

- Business Layer
- Data Access Layer
- Schema Mapping
- Conversion of Data
- Guid Generation

All the modules are controlled by the user interface.

4.2 Implementation Details

The implementation of the data migration model supports the migration of data from a source to a destination without having prior semantic knowledge of the structure of the data. As we are migrating a relational database to a cloud database, our source system can be Oracle, MySQL or Microsoft SQL Server. Any system which has a relational database can act our source system. Our destination system is a cloud database which supports key-value pairs. In this thesis, we migrate relational data to **Amazon SimpleDB**.

We use the **Microsoft SQL Server** to store our relational database. To implement our model, we use Microsoft .Net Framework 3.5, Microsoft IIS 7.0 and Microsoft SQL Server 2008 R2. We use Amazon web service SDK for Microsoft Visual Studio. We use the C# library of SimpleDB to perform all necessary action for migrating the data.

The migration system makes use of inbuilt Microsoft libraries to fetch the data from the Microsoft SQL Server in the data table. After that, it converts data in the form of key-value pairs. The key-value is the basic data structure supported by Amazon SimpleDB. Amazon SimpleDB identifies every item (row in table) with a unique id which is known as an itemname. In order to make each item uniquely identified, we generate a globally unique identifier (GUID) for every item to be inserted. Before inserting an item into Amazon Simpledb, key-value pairs are converted into string data type because Amazon SimpleDB recognizes data in the form of strings only. Finally, the data is inserted into SimpleDB.

In the following section, we discuss each component of our model in detail.

4.2.1 Business Layer

The business layer store variables required by the application while moving from one graphical user interface to another graphical user interface. The current processing database, and tables information is maintained by the business layer. Each interface is linked to a previous interface for displaying the tables, and the databases and other information related to the database. The business layer also stores the Amazon SimpleDB secret key and the access key behind the scenes so that users do not need to input access keys again and again. By providing a business layer, the model is more interactive, user friendly, and flexible and decreases complication.

4.2.2 Data Access Layer

A data access layer (DAL) is a layer which provides simplified access to the data stored in persistent storage of the source database. In this case, our source database is the Microsoft SQL Server. In the following section we present two data access layers:

- (Microsoft/Oracle/Mysql) Data Access Layer: This access layer forms a basis of the model. The layer accesses various databases available in the system. It fetches the underlying schemas for the tables. During the migration it fetches the data from the respective table and transfers the data to the cloud server data access layer. The layer uses standard data table provided by the Microsoft .Net Framework to store data.

The layer automatically tests the connection with the server. It automatically opens and closes the connection with the server. Once the model is connected to a particular server, it automatically fetches the underlying databases, tables and data from the server. The layer also handles exceptions which may be generated during migration.

- The cloud Server Data Access Layer: The Amazon SimpleDB service provides small groups of API calls that supports the process of data migration from relational databases. Every cloud service provider provides its own API for data access. The methods used by us from Amazon SimpleDB are:
  - PutAttributes: Provides core concepts of inserting data into Amazon SimpleDB.
  - BatchPutAttributes: Provides a concept of inserting bulk data inside Amazon SimpleDB. It can insert up to twenty five items at one time.
  - ListDomains: Lists all the domains (table) that are available in Amazon

SimpleDB.

- CreateDomain: Provides a way of creating domains (table) in the Amazon SimpleDB.

The (Microsoft/Oracle/Mysql) data Access layer retrieves databases, tables, schemas and transfers table data and schemas to the cloud server data access layer. The cloud server data access layer inserts the data retrieved by the (Microsoft/Oracle/Mysql) data access layer into Amazon SimpleDB.

4.2.3 GUID Generation and Conversion of Data

By running Amazon SimpleDB instance, we are inserting relational data into SimpleDB. The items are similar to the rows in the database table. Each item identifies a single object and has data for a single item in the form of key-value attributes. Each item is recognized by a unique key or an identifier, a primary key in a database table. In order to insert the data into a cloud server, we generate a globally unique identifier (GUID), for every item (row, in traditional terminology). This removes the chances of duplication and prevents us from getting exceptions during migration and makes the migration process smooth.

SimpleDB stores all the data as an UTF-8 string. Thus it becomes easy for SimpleDB to automatically index the data and retrieve the data quickly. All other kinds of data types such as numbers, and dates must be converted into strings. Developers must ensure the correct encoding of the data, before storing data into SimpleDB. In Figure 24, module **Data Type Conversion** converts all the data types into string and ensures that every key value inserted into SimpleDB is of a string type.

4.2.4 Graphical User Interface

The user interface controls all the modules indirectly. The user interface plays a vital role in migration. It performs many functions.

1. Authentication of the source and the target database: In order to connect to SimpleDB, the migration model must receive the following information from the user:

    - Access Key: Access key of the user to its SimpleDB account.
    - Secret Access Key: Secret access key of the user to its SimpleDB account.

    The above keys are required to connect to Amazon SimpleDB. Amazon SimpleDB automatically opens and closes the connection. The interface is depicted in Figure 25.

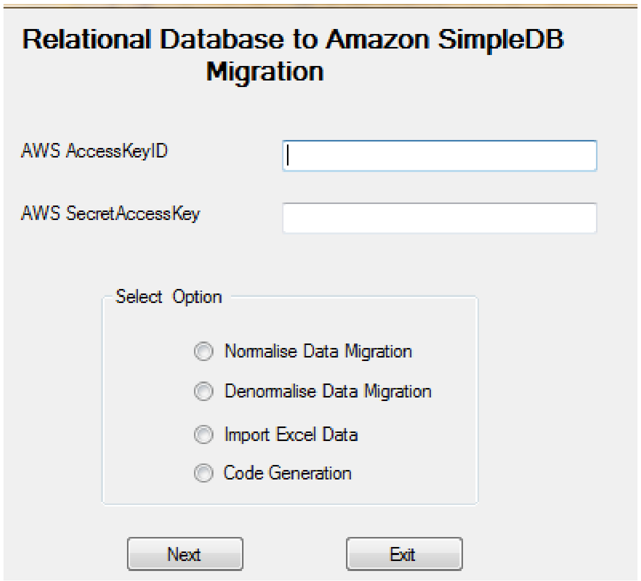
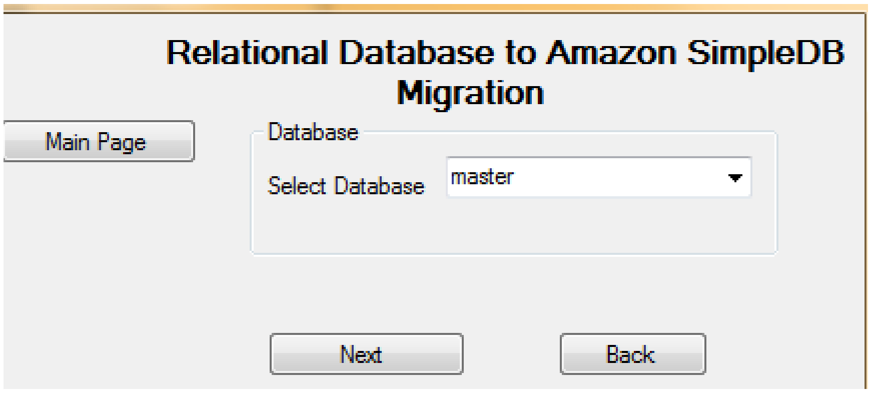

Relational Database to Amazon SimpleDB Migration
AWS AccessKeyID
AWS SecretAccessKey
Select Option
○ Normalise Data Migration
○ Denormalise Data Migration
○ Import Excel Data
○ Code Generation
Next
Exit

Figure 25: Authentication Amazon SimpleDB

Similarly, the source system provides authentication. Once authentication is done, the system proceeds to the next interface.

2. Display and Selection of Database: After authentication is done with the source and target system, the interface will display existing databases in the system. The interface is presented in Figure 26.



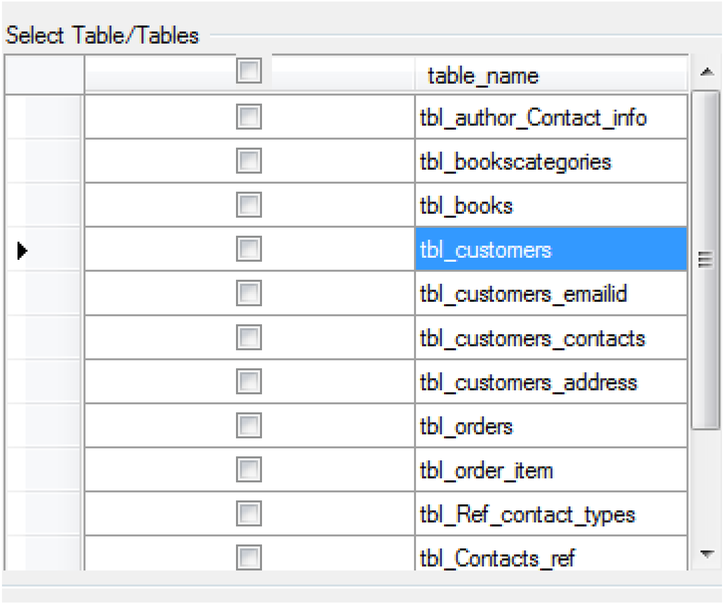
Relational Database to Amazon SimpleDB Migration

Database

Select Database: master

Figure 26: Database selection

3. Display of Tables: Once the source database to be migrated is selected, then interface displays available tables from the database. The user can select and migrate tables or tables data to the cloud database. Figure 27 presents the interface.



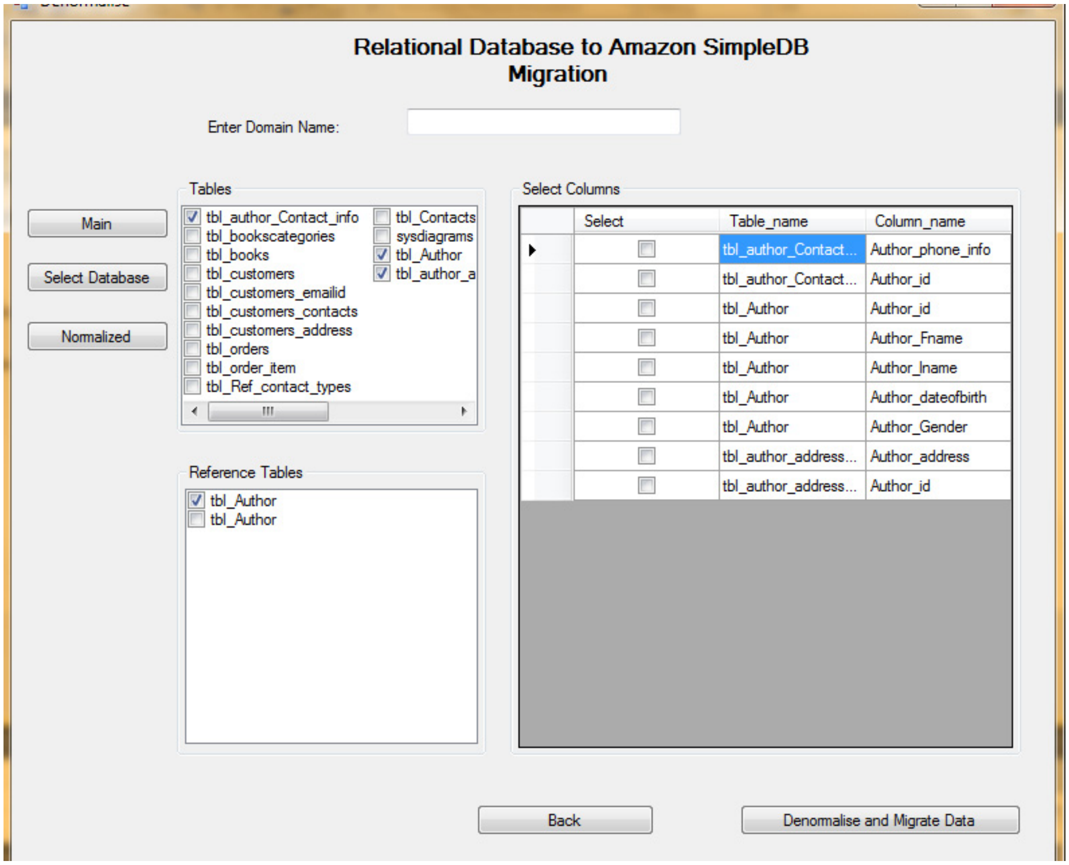
Select Table/Tables

	□	table_name
	□	tbl_author_Contact_info
	□	tbl_bookscategories
	□	tbl_books
▶	□	tbl_customers
	□	tbl_customers_emailid
	□	tbl_customers_contacts
	□	tbl_customers_address
	□	tbl_orders
	□	tbl_order_item
	□	tbl_Ref_contact_types
	□	tbl_Contacts_ref

Figure 27: Display tables

In Figure 27, column “table_nam”represents the name of the tables present in the database. Users can select tables or tables data that need to migrate.

4. Display of Schema: In Type 4, Normalization to Denormalization and Tables to Domain migration method display schema of tables. This enables user to select only those columns that need to be migrated. This type typically involves combining columns from different tables and migrating columns and their data to a single domain. Figure 28 presents the interface.



Relational Database to Amazon SimpleDB Migration

Enter Domain Name:

Tables

☑ tbl_author_Contact_info
☐ tbl_bookscategories
☐ tbl_books
☐ tbl_customers
☐ tbl_customers_emailid
☐ tbl_customers_contacts
☐ tbl_customers_address
☐ tbl_orders
☐ tbl_order_item
☐ tbl_Ref_contact_types
☐ tbl_Contacts
☐ sysdiagrams
☑ tbl_Author
☑ tbl_author_a

Reference Tables

☑ tbl_Author
☐ tbl_Author

Select Columns

Select	Table_name	Column_name
☐	tbl_author_Contact...	Author_phone_info
☐	tbl_author_Contact...	Author_id
☐	tbl_Author	Author_id
☐	tbl_Author	Author_Fname
☐	tbl_Author	Author_lname
☐	tbl_Author	Author_dateofbirth
☐	tbl_Author	Author_Gender
☐	tbl_author_address...	Author_address
☐	tbl_author_address...	Author_id

Figure 28: Schema display

In Figure 28, section “Tables”represents a list of tables that are present in the database. Section “Reference Tables”displays tables which are referenced by tables in the “Tables”section. Section “Select Columns”displays the schema of selected tables.

5. Display Migration Result : This displays a message to user that “Data is migrated successfully ”.

4.2.5 Schema Mapping

The schema mapping component is used to create mapping rules between the source and the target system. These rules are according to the users specifications. Our system supports two types of mapping:

- Table-Domain Mapping
- Column-Attribute Mapping

The “**Domain**”in Amazon SimpleDB is similar to “**Table**”in relational database. “**Attribute**”in the Amazon SimpleDB is similar to “**Column**”in the relational database. In the Type 1, Type 2 and Type 3 migration methods, columns are implicitly mapped to attributes in Amazon SimpleDB.

In the Type 4 migration method, the columns are explicitly mapped to attributes in Amazon SimpleDB. During the migration process, these columns will be automatically converted into attributes in the target database.

The Table-Domain Mapping maps the tables in the relational database to domains in the Amazon SimpleDB. In the Type 1, Type 2 and Type 4 migration methods, the user needs to input the domain name. The Type 3 migration method supports implicit conversion of the name of tables to the name of domains. Once the migration process starts, it will automatically map tables and columns of the relational database to domains and attributes respectively.

The rule mapping module allows the user to manage the rules created. Before data migration, the user can validate created rules and proceed with or without all the rules. The user interface gives the option to the user to change the mapping before the actual migration process starts.

4.3 Data Migration Objectives

Our approach to data migration serves the following objectives:

- Robust and User-Friendly: A majority of the data migration process is automatically done. DBA and developers require semantic knowledge for interaction. This makes the data migration process user friendly with few inputs from the user. Because most of the process is automatic, this speeds up the data migration from the source to the destination database. In this process of migration, the developers are not required to have knowledge of the underlying object-relational mapping every time.

- Flexibility: Our model presents different ways of migrating data. This gives flexibility to the organizations to migrate their large data sets. The user can select any of the approaches depending on the feasibility, the flexibility, and the size of data to be migrated in the cloud database.

4.4 Limitations

In this thesis, we present a system to automate the data migration from relational databases to cloud databases. The limitations of the model are:

- Stored Procedure: Stored procedures are pre-compiled objects stored in the database. Stored procedure is also a batch which is stored under a name and it is executed as a single unit. Because cloud databases does not provide stored procedure functionality, we cannot migrate existing stored procedures of relational databases to cloud databases.

- User-defined Function: User-defined function is a T-SQL routine that returns a value. Because cloud databases does not provide any User-defined function functionality, we cannot migrate existing functions of the database to the cloud database.

- Triggers: Triggers are a code block that are comprised of set of T-SQL statements that are executed (fired) in response to an event such as Insert, Delete or Update statements on a table. Trigger can also be considered to be special type of a stored procedure that gets fired automatically in an event. Because cloud databases do not provide any triggers functionality, we cannot migrate existing triggers of the databases to the cloud databases.

Chapter 5

Experiments

In order to evaluate our model, we perform experiments to migrate the relational database to the cloud database (SimpleDB). In this chapter, we present the experimental results of our migration model.

We based our experiments on a relational database of the "**online bookstore**"application. The sample database consists of thirteen tables and sample data. The experiments are processed in the following environment:

- Dell Inspirion N5110
- Intel Core i3 processor
- 6GB DDR3 1066mHz RAM
- 10Mbps ADSL2 internet connection

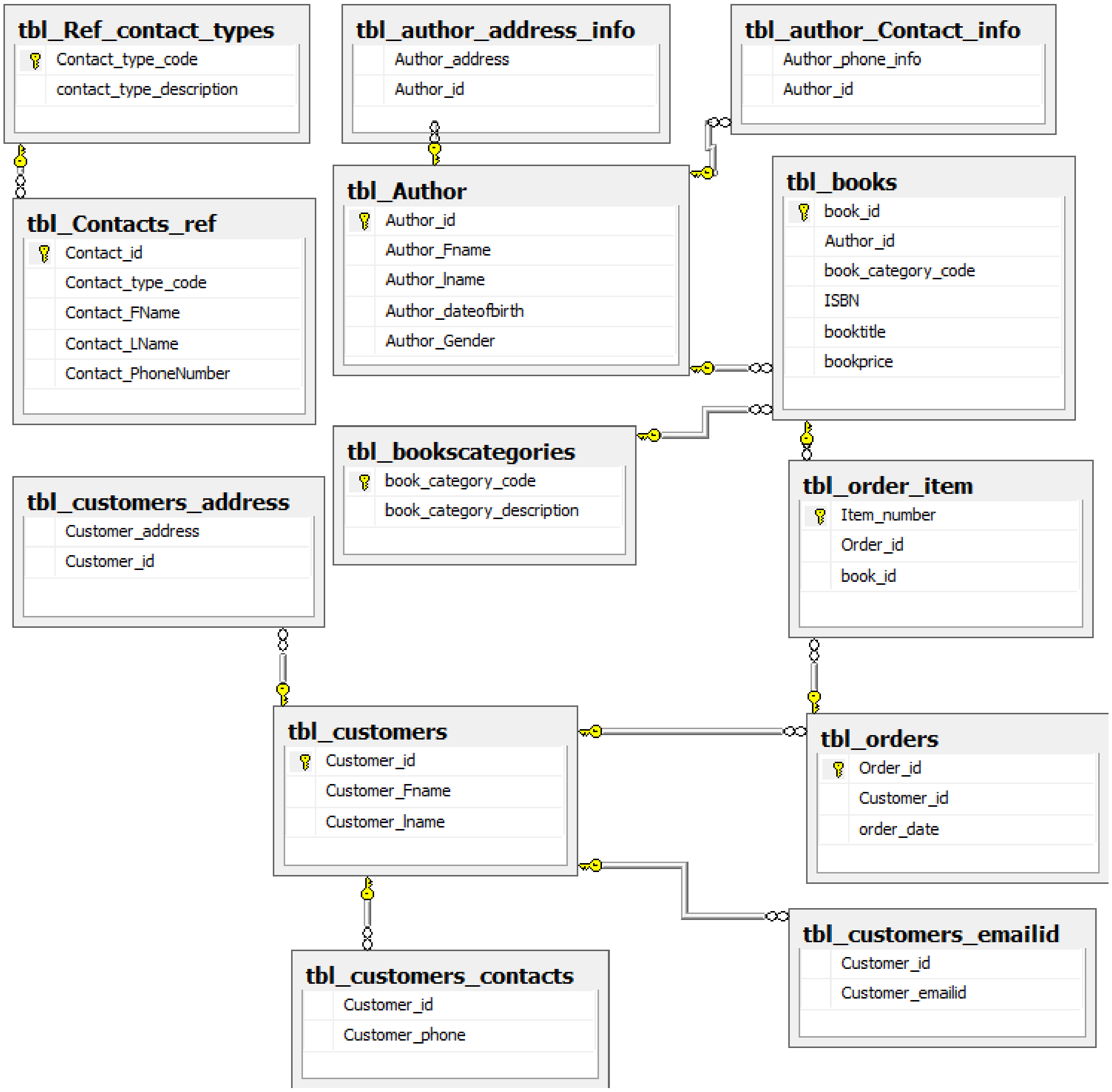


Figure 29: Relational schema used in the experiments

Figure 29 presents the online bookstore schema that was used.

5.1 Type 1 Migration Experiment

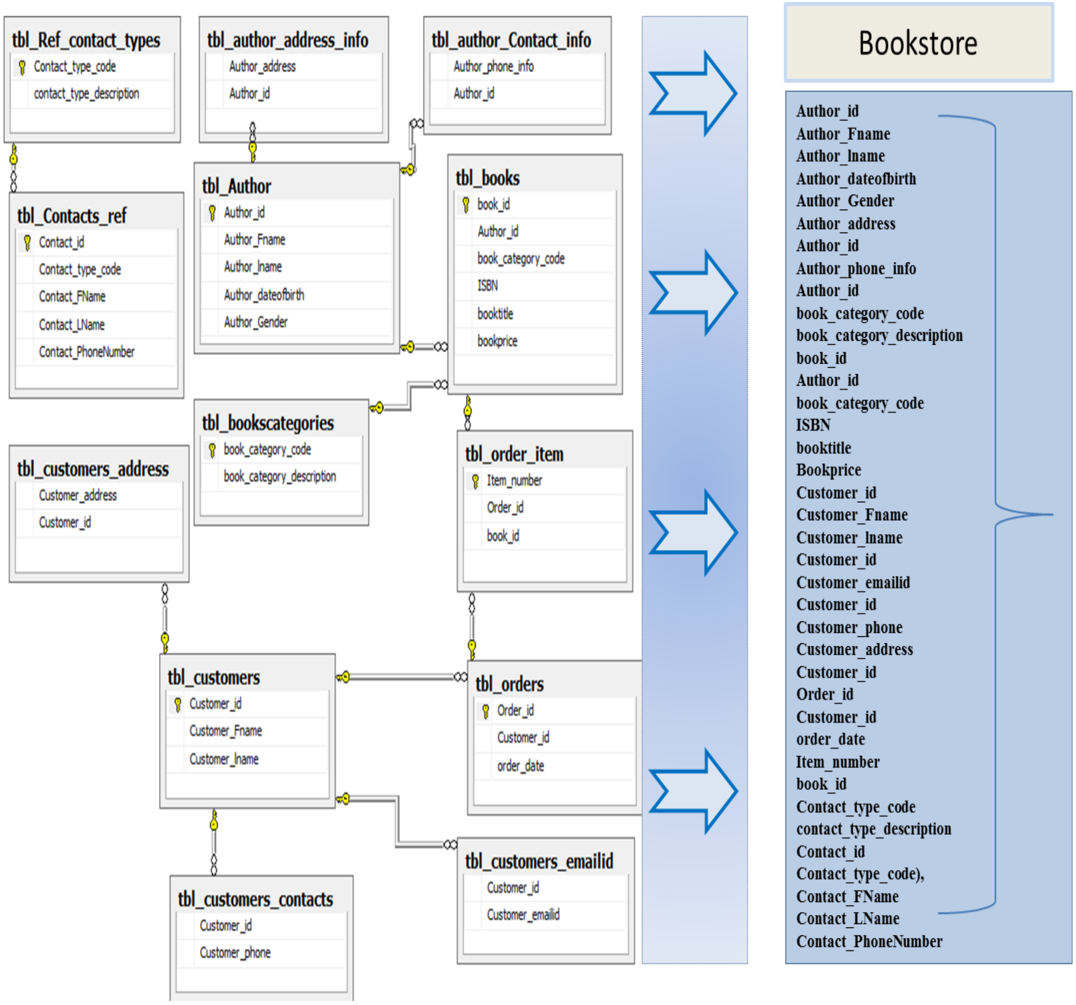
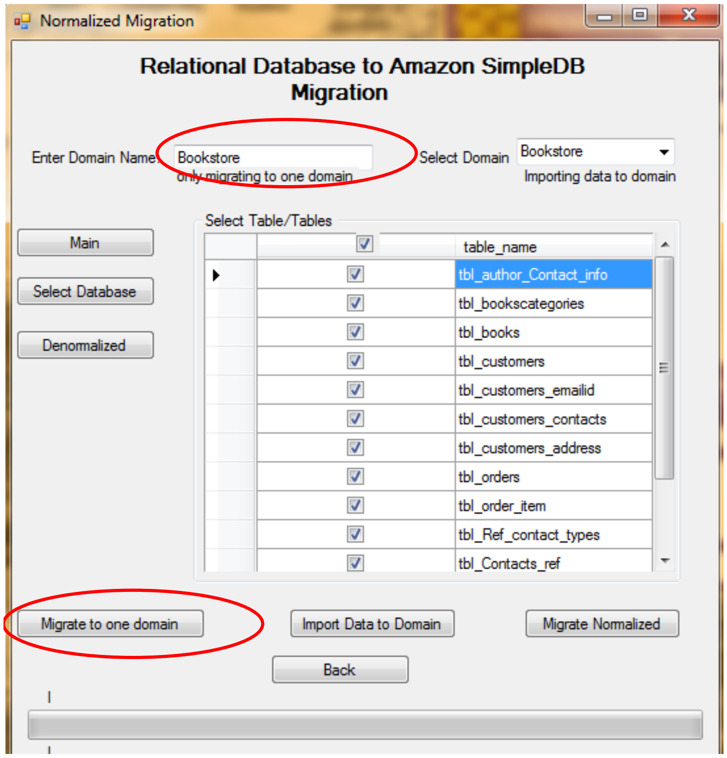


Figure 30: Type 1 conversion

The Type 1 migration process involves migrating the whole relational database to a single domain Amazon SimpleDB (cloud). It refers to migrating entire data from all the tables in a relational database to a single domain of Amazon SimpleDB. Because the old system and the new system have different object models, this results in different database schemata.

Suppose we create a new domain in SimpleDB as "**Bookstore**", all data will be migrated to this domain in the form of key-value pairs. Figure 30 depicts the migrated database in the domain "**Bookstore**". Figure 31 shows the interface in which users can select all tables and enter user defined domain name.



Relational Database to Amazon SimpleDB Migration

Enter Domain Name: Bookstore
only migrating to one domain

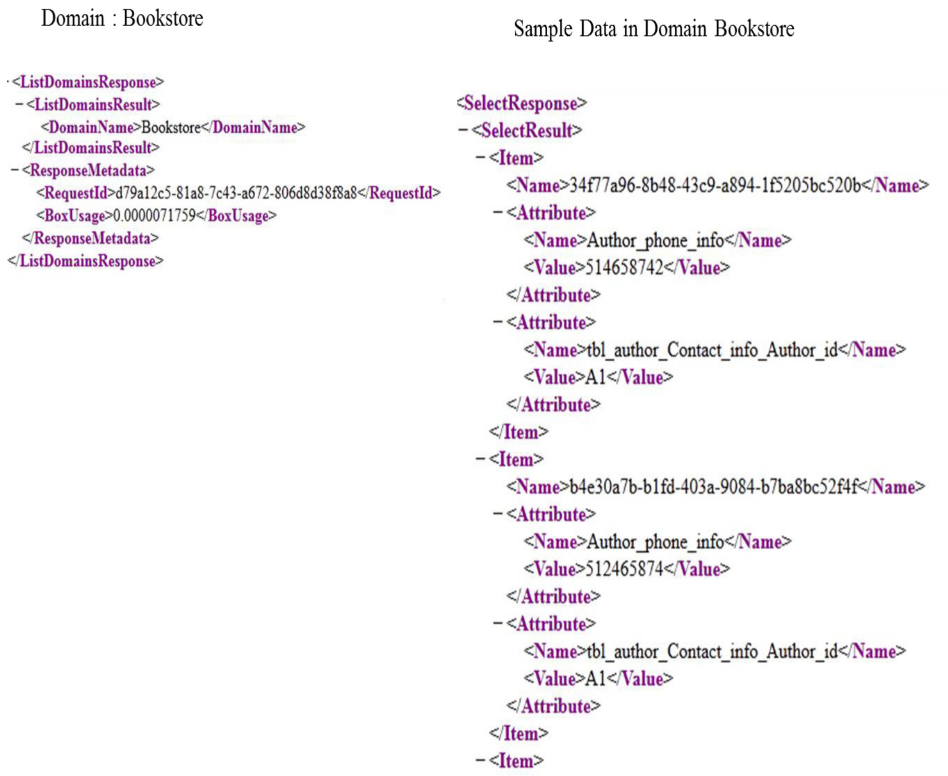
Select Domain: Bookstore
Importing data to domain

Select Table/Tables

☑	table_name
☑	tbl_author_Contact_info
☑	tbl_bookscategories
☑	tbl_books
☑	tbl_customers
☑	tbl_customers_emailid
☑	tbl_customers_contacts
☑	tbl_customers_address
☑	tbl_orders
☑	tbl_order_item
☑	tbl_Ref_contact_types
☑	tbl_Contacts_ref

Figure 31: Type 1 tool interface

The migrated domain and sample data are presented in Figure 32.



```
<ListDomainsResponse>
 - <ListDomainsResult>
     <DomainName>Bookstore</DomainName>
   </ListDomainsResult>
 - <ResponseMetadata>
     <RequestId>d79a12c5-81a8-7c43-a672-806d8d38f8a8</RequestId>
     <BoxUsage>0.0000071759</BoxUsage>
   </ResponseMetadata>
</ListDomainsResponse>
```

```
<SelectResponse>
- <SelectResult>
  - <Item>
      <Name>34f77a96-8b48-43c9-a894-1f5205bc520b</Name>
    - <Attribute>
        <Name>Author_phone_info</Name>
        <Value>514658742</Value>
      </Attribute>
    - <Attribute>
        <Name>tbl_author_Contact_info_Author_id</Name>
        <Value>A1</Value>
      </Attribute>
    </Item>
  - <Item>
      <Name>b4e30a7b-b1fd-403a-9084-b7ba8bc52f4f</Name>
    - <Attribute>
        <Name>Author_phone_info</Name>
        <Value>512465874</Value>
      </Attribute>
    - <Attribute>
        <Name>tbl_author_Contact_info_Author_id</Name>
        <Value>A1</Value>
      </Attribute>
    </Item>
  - <Item>
```

Figure 32: Type 1 SimpleDB interface

5.1.1 Join

SimpleDB does not support the concept of a join. We provide an example to handle a join in a Type 1 migration. Developers can make simultaneous queries to domains and exploit data at the design time.

Example 8 *Suppose we need to fetch data from the tables **tbl_Author** and **tbl_author_Contact_Info** for a particular Author_id. Table 5 compares and handles join functionality in relational database and SimpleDB.*

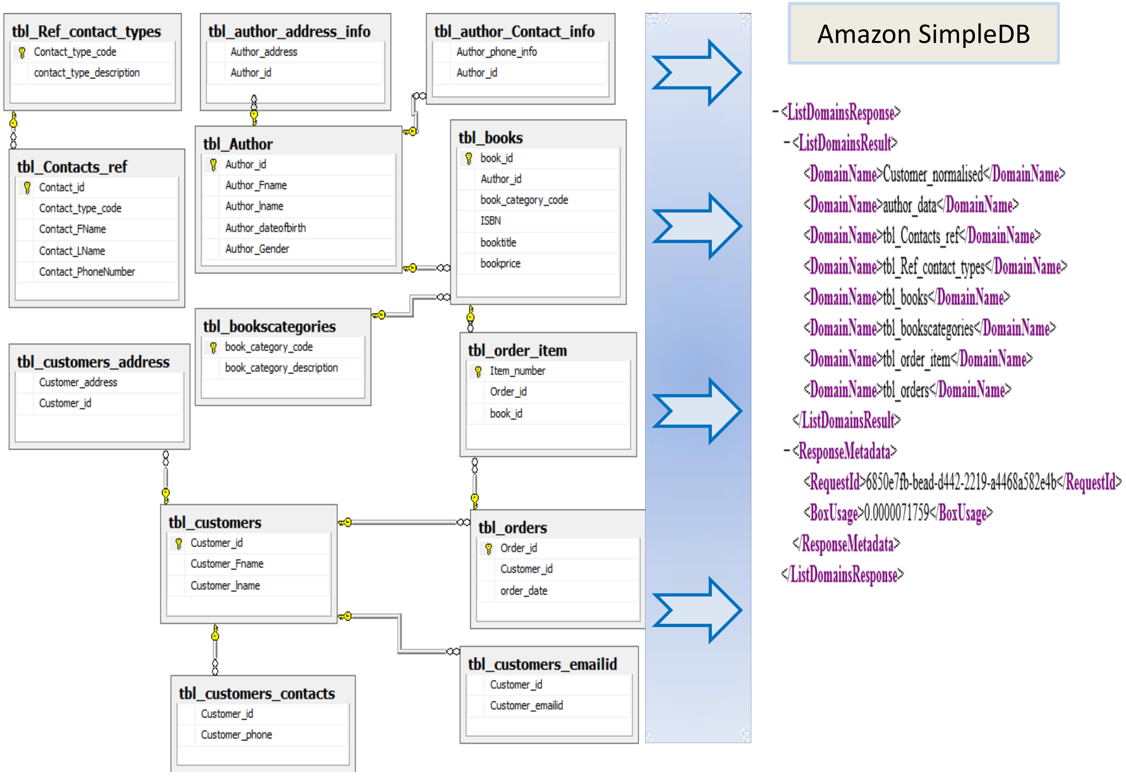
Table 6: Type 1 join

RDBMS	SimpleDB
Select Author_Fname, Author_lname, Author_dateofbirth, Author_Gender, Author_phone_info FROM tbl_Author JOIN tbl_author_Contact_Info ON tbl_Author.Author_id = tbl_author_Contact_Info.Author_id WHERE tbl_author_Contact_Info.Author_id = 'A1'	Select Author_Fname, Author_lname, Author_dateofbirth, Author_Gender from Bookstore where Author_id = 'A1' Select Author_phone_info FROM Bookstore where Author_id = 'A1'

5.2 Type 2 Migration Experiment

In this migration, we migrate data of relevant tables on which users are likely to make joins in one domain in SimpleDB. Most of the tables where the user performs joins have primary-foreign key relationships. The other tables are migrated as “Table-Domain ”and the column as “Column-Attribute”in Amazon SimpleDB.

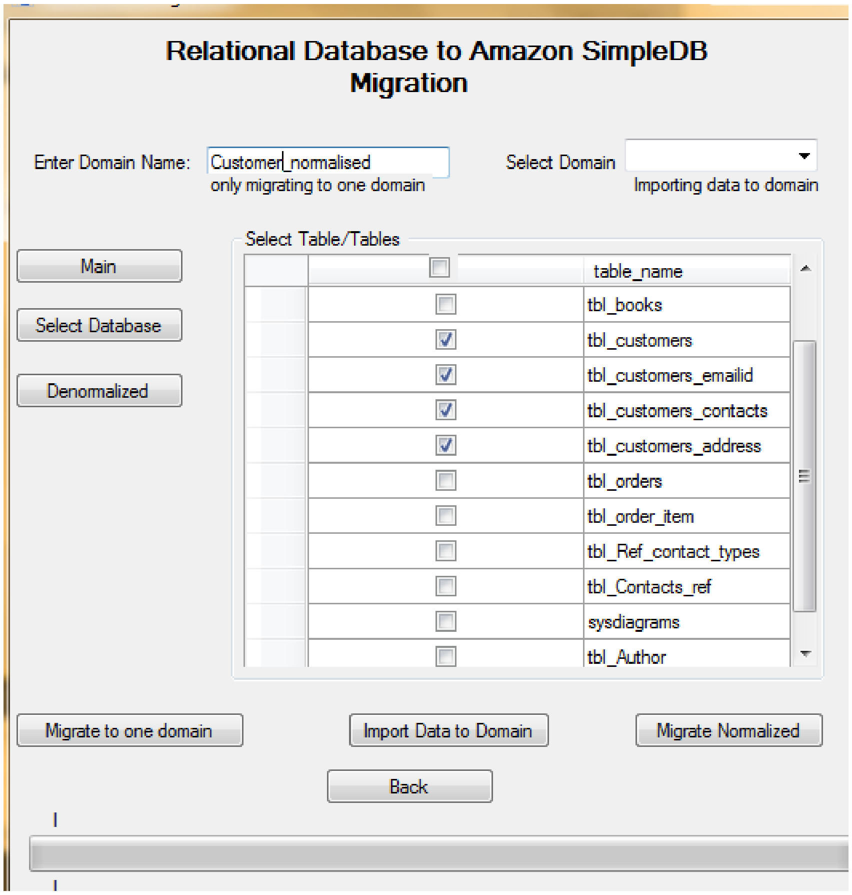
In the online bookstore schema, the tables data of “**tbl_Author**”, “**tbl_author_address_info**”and “**tbl_author_Contact_info**”are migrated to one domain, say, “**author_data**”. Similarly, customer data of tables “**tbl_customers**”, “ **tbl_customers_emailid**”, “**tbl_customers_address**”and “**tbl_customers_address**”are combined to one domain, say, “**Customer_normalised**”. All other tables and their data are migrated as individual domains in SimpleDB. Figure 33 presents a Type 2 migration.



```
- <ListDomainsResponse>
  - <ListDomainsResult>
      <DomainName>Customer_normalised</DomainName>
      <DomainName>author_data</DomainName>
      <DomainName>tbl_Contacts_ref</DomainName>
      <DomainName>tbl_Ref_contact_types</DomainName>
      <DomainName>tbl_books</DomainName>
      <DomainName>tbl_bookscategories</DomainName>
      <DomainName>tbl_order_item</DomainName>
      <DomainName>tbl_orders</DomainName>
    </ListDomainsResult>
  - <ResponseMetadata>
      <RequestId>6850e7fb-bead-d442-2219-a4468a582e4b</RequestId>
      <BoxUsage>0.0000071759</BoxUsage>
    </ResponseMetadata>
  </ListDomainsResponse>
```

Figure 33: Type 2 conversion

Figure 34 shows the interface in which the user can select relevant tables and enter a user defined domain name. Data and schemas are automatically transferred to SimpleDB. The interface shows migration of all customer data in domain “**Customer_normalised**”.



Relational Database to Amazon SimpleDB Migration

Enter Domain Name: Customer_normalised
only migrating to one domain

Select Domain
Importing data to domain

Select Table/Tables

☐	table_name
☐	tbl_books
☑	tbl_customers
☑	tbl_customers_emailid
☑	tbl_customers_contacts
☑	tbl_customers_address
☐	tbl_orders
☐	tbl_order_item
☐	tbl_Ref_contact_types
☐	tbl_Contacts_ref
☐	sysdiagrams
☐	tbl_Author

Figure 34: Type 2 interface

5.2.1 Join

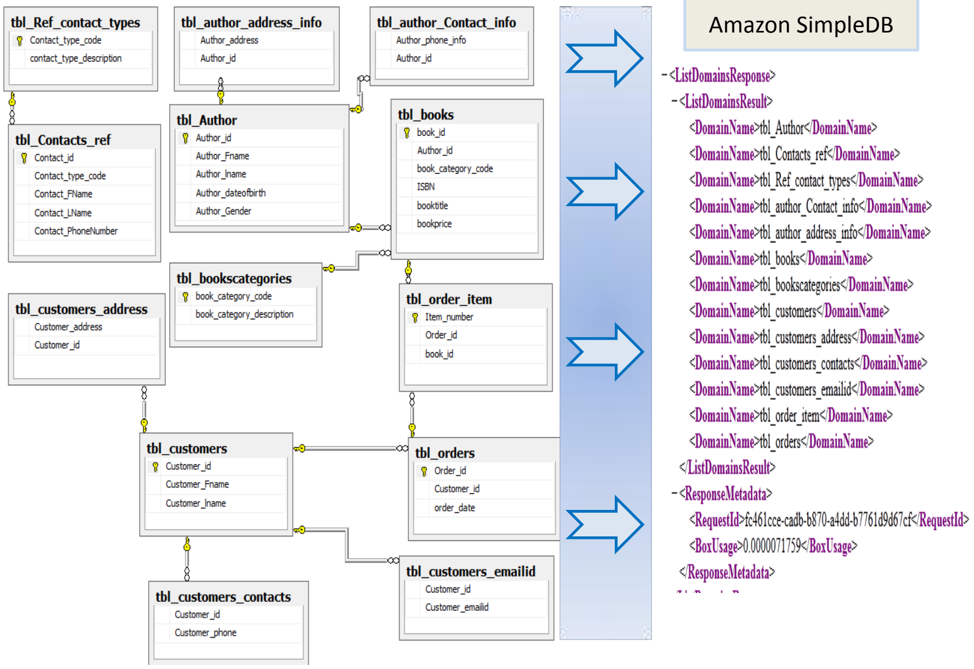
The following example handles join in Type 2 migration. Developers can make simultaneous queries to the domain and exploit data at the design time.

Example 9 *Suppose we need to fetch data from the tables "**tbl_Author**"and "**tbl_author_Contact_Info**"for a particular Author_id. Table 8 compares and handles join functionality in the relational databases and in the SimpleDB.*

Table 7: Type 2 join

RDBMS	SimpleDB
Select Author_Fname, Author_lname, Author_dateofbirth, Author_Gender, Author_phone_info FROM tbl_Author JOIN tbl_author_Contact_Info ON tbl_Author.Author_id = tbl_author_Contact_Info.Author_id WHERE tbl_author_Contact_Info.Author_id = 'A1'	Select Author_Fname, Author_lname, Author_dateofbirth, Author_Gender from author_data where Author_id = 'A1' Select Author_phone_info FROM author_data where Author_id = 'A1'

5.3 Type 3 Migration Experiment



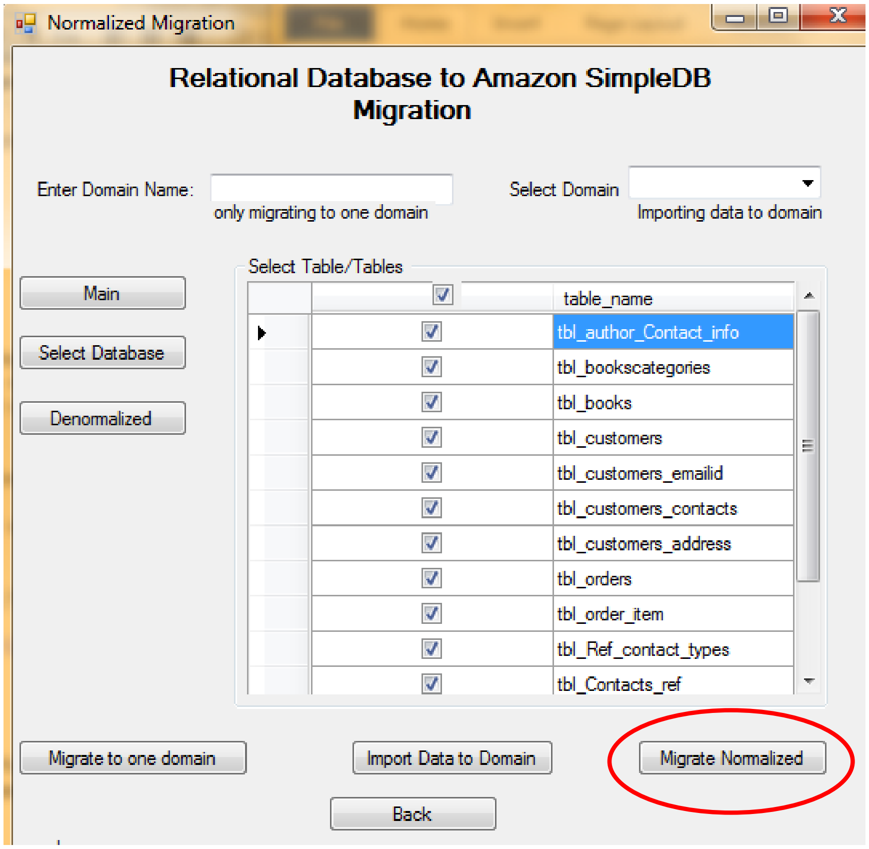
```
-<ListDomainsResponse>
  -<ListDomainsResult>
     <DomainName>tbl_Author</DomainName>
     <DomainName>tbl_Contacts_ref</DomainName>
     <DomainName>tbl_Ref_contact_types</DomainName>
     <DomainName>tbl_author_Contact_info</DomainName>
     <DomainName>tbl_author_address_info</DomainName>
     <DomainName>tbl_books</DomainName>
     <DomainName>tbl_bookscategories</DomainName>
     <DomainName>tbl_customers</DomainName>
     <DomainName>tbl_customers_address</DomainName>
     <DomainName>tbl_customers_contacts</DomainName>
     <DomainName>tbl_customers_emailid</DomainName>
     <DomainName>tbl_order_item</DomainName>
     <DomainName>tbl_orders</DomainName>
   </ListDomainsResult>
  -<ResponseMetadata>
     <RequestId>fc461cce-cadb-b870-a4dd-b7761d9d67cf</RequestId>
     <BoxUsage>0.0000071759</BoxUsage>
   </ResponseMetadata>
```

Figure 35: Type 3 conversion

In this approach, each table is mapped to a single domain of SimpleDB and the column is mapped to an attribute of SimpleDB. Once the migration process starts, the tables and data will automatically map to the cloud database.

In the "**online bookstore**"schema, every single table is mapped to a single domain and the data is migrated into the respective domain. Figure 35 presents the Type 3 migration.

Figure 36 shows an interface in which the user can select all the tables. Data and schemas are automatically transferred to SimpleDB.



Normalized Migration

Relational Database to Amazon SimpleDB Migration

Enter Domain Name:
only migrating to one domain

Select Domain
Importing data to domain

Select Table/Tables

	☑	table_name
▸	☑	tbl_author_Contact_info
	☑	tbl_bookscategories
	☑	tbl_books
	☑	tbl_customers
	☑	tbl_customers_emailid
	☑	tbl_customers_contacts
	☑	tbl_customers_address
	☑	tbl_orders
	☑	tbl_order_item
	☑	tbl_Ref_contact_types
	☑	tbl_Contacts_ref

Figure 36: Type 3 interface

5.3.1 Join

The following example handles a join in Type 3 migration. The developer can make simultaneous queries to different domains and exploit data at the design time.

Example 10 *Suppose we need to fetch data from the tables “* ***tbl_Author****”and “* ***tbl_author_Contact_Info****”for a particular Author_id. Table 7 compares and handles join functionality in relational database and SimpleDB.*

Table 8: Type 3 join

RDBMS	SimpleDB
Select Author_Fname, Author_lname, Author_dateofbirth, Author_Gender, Author_phone_info FROM tbl_Author JOIN tbl_author_Contact_Info ON tbl_Author.Author_id = tbl_author_Contact_Info.Author_id WHERE tbl_author_Contact_Info.Author_id = 'A1'	Select Author_Fname, Author_lname, Author_dateofbirth, Author_Gender from tbl_Author where Author_id = 'A1' Select Author_phone_info FROM tbl_author_Contact_Info where Author_id = 'A1'

5.4 Type 4 Migration Experiment

First, the migration denormalizes the relational database based on the semantics. Once semantics are provided by the user, it will automatically denormalize the database and migrate the data to cloud database. The denormalization process is important and this will utilize the multi-value attribute property of Amazon SimpleDB and migrate data in the form of arrays.

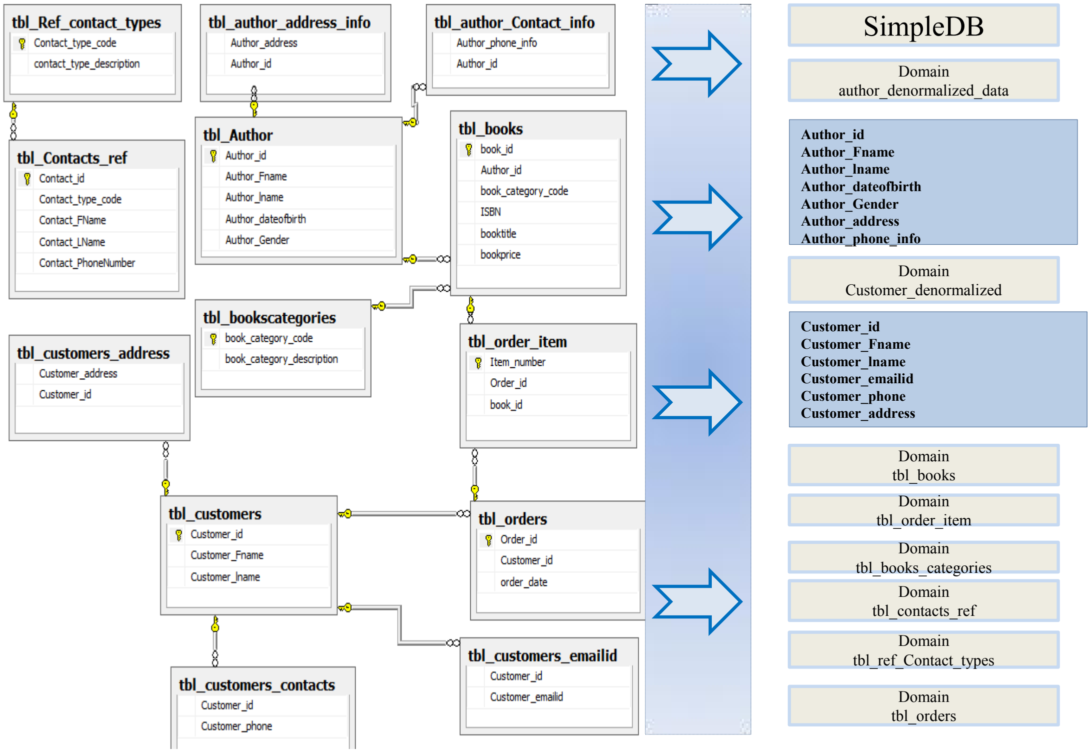
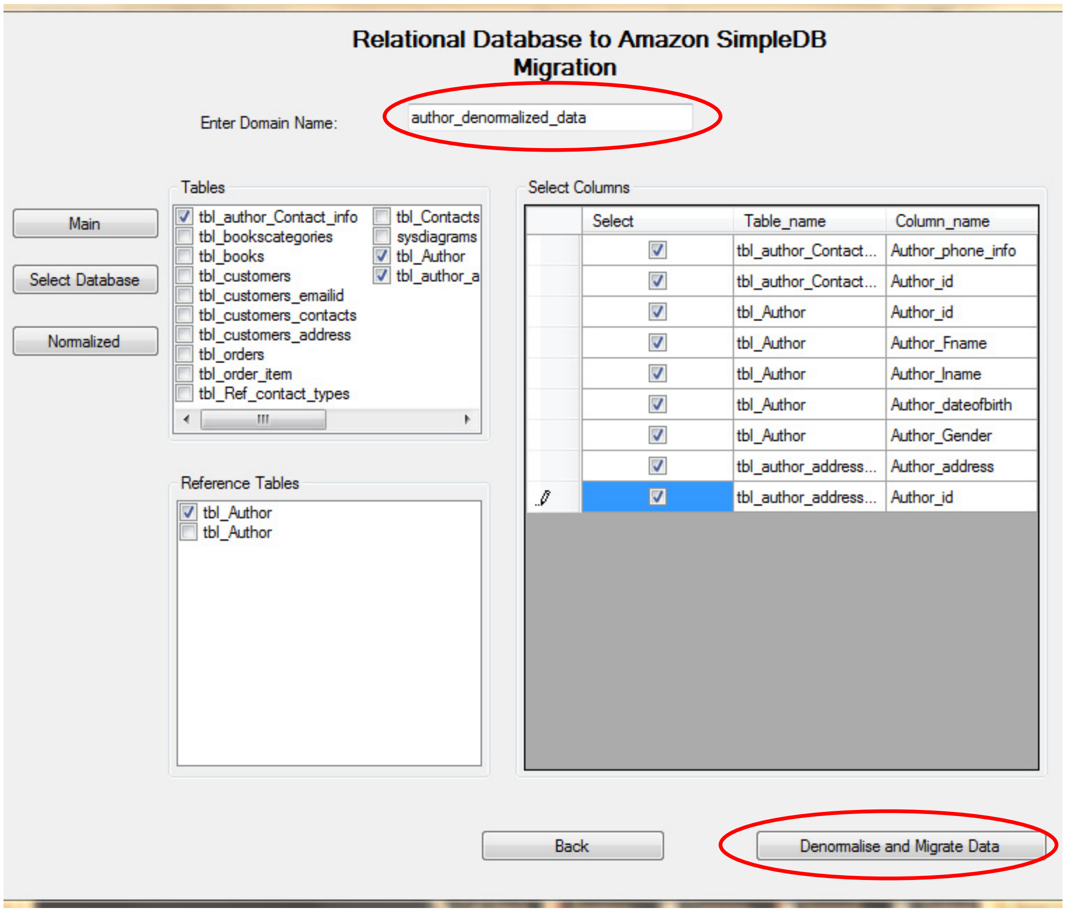


Figure 37: Type 4 migrated schema

In "**Bookstore**"schema, table columns of "**tbl_author_address_info**"and "**tbl_author_Contact_info**"are combined to "**tbl_author** "and form a denormalized domain, say "**author_denormalized_data** "with column "**Author_phone_info**" and "**tbl_author_address_info**"forming multi-value attributes in SimpleDB.

Similarly, table columns of "**tbl_customers_emailid**", "**tbl_customers_address**"and "**tbl_customers_address**"are combined to "**tbl_customers**"to form a denormalized domain, say "**Customer_denormalized**". All other tables and their data are migrated as individual domains in SimpleDB. Figure 37 depicts the migrated database schema.

Figure 38 presents an interface in which user can select tables and columns based on semantics. Data and schemas will automatically transfer to SimpleDB.



Relational Database to Amazon SimpleDB Migration

Enter Domain Name: author_denormalized_data

Tables

- ☑ tbl_author_Contact_info
- ☐ tbl_bookscategories
- ☐ tbl_books
- ☐ tbl_customers
- ☐ tbl_customers_emailid
- ☐ tbl_customers_contacts
- ☐ tbl_customers_address
- ☐ tbl_orders
- ☐ tbl_order_item
- ☐ tbl_Ref_contact_types
- ☐ tbl_Contacts
- ☐ sysdiagrams
- ☑ tbl_Author
- ☑ tbl_author_a

Reference Tables

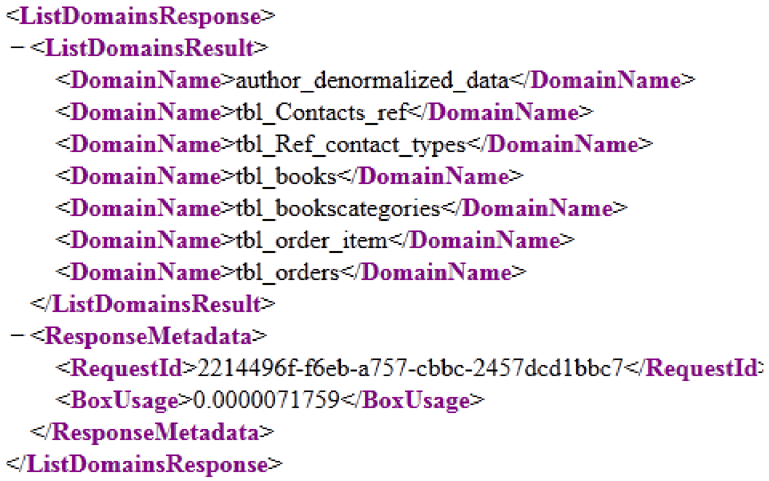
- ☑ tbl_Author
- ☐ tbl_Author

Select Columns

Select	Table_name	Column_name
☑	tbl_author_Contact...	Author_phone_info
☑	tbl_author_Contact...	Author_id
☑	tbl_Author	Author_id
☑	tbl_Author	Author_Fname
☑	tbl_Author	Author_lname
☑	tbl_Author	Author_dateofbirth
☑	tbl_Author	Author_Gender
☑	tbl_author_address...	Author_address
☑	tbl_author_address...	Author_id

Figure 38: Type 4 interface

The migrated domain is presented in Figure 39.



```
<ListDomainsResponse>
 - <ListDomainsResult>
     <DomainName>author_denormalized_data</DomainName>
     <DomainName>tbl_Contacts_ref</DomainName>
     <DomainName>tbl_Ref_contact_types</DomainName>
     <DomainName>tbl_books</DomainName>
     <DomainName>tbl_bookscategories</DomainName>
     <DomainName>tbl_order_item</DomainName>
     <DomainName>tbl_orders</DomainName>
   </ListDomainsResult>
 - <ResponseMetadata>
     <RequestId>2214496f-f6eb-a757-cbbc-2457dcd1bbc7</RequestId
     <BoxUsage>0.0000071759</BoxUsage>
   </ResponseMetadata>
</ListDomainsResponse>
```

Figure 39: Type 4 SimpleDB interface

5.4.1 Join

The following example handles join in Type 4 migration.

Example 11 *Suppose we need to fetch data from the tables ”**tbl_Author**“and ”**tbl_author_Contact_Info**“for a particular Author_id. Table 8 compares and handles a join functionality in the relational database and SimpleDB.*

Table 9: Type 4 join

RDBMS	SimpleDB
Select Author_Fname, Author_lname, Author_dateofbirth, Author_Gender, Author_phone_info FROM tbl_Author JOIN tbl_author_Contact_Info ON tbl_Author.Author_id = tbl_author_Contact_Info.Author_id WHERE tbl_author_Contact_Info.Author_id = ’A1’	SELECT * FROM author_denormalized_data WHERE Author_id = ’A1’

5.5 Code Generation

We propose an interface which will assist the developer to generate code automatically. This includes the basic usage of Select, Insert, Delete and Update queries. The interface requires following inputs:

- Domain Name: The user needs to select a domain against which user will make queries.

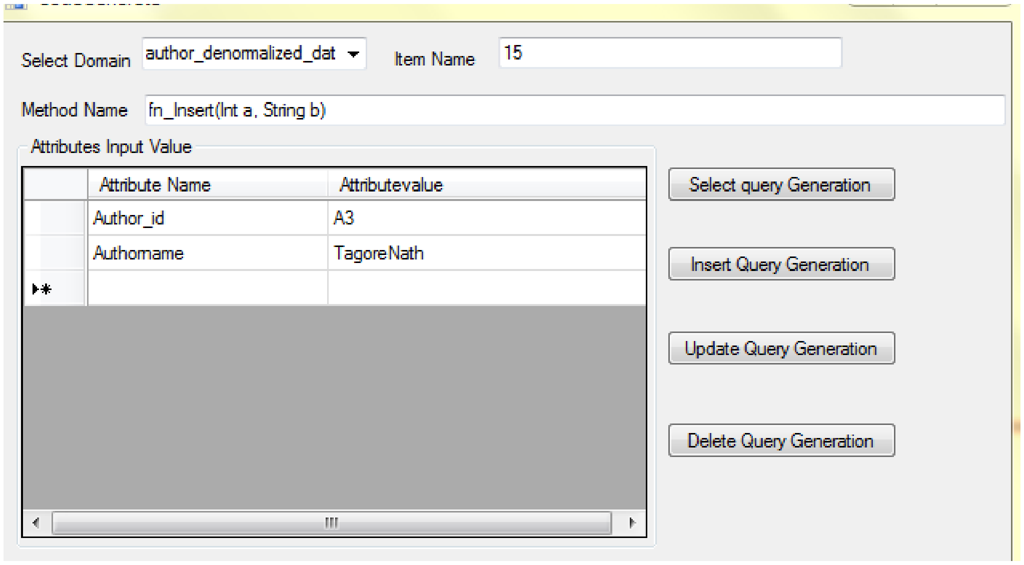
- Item Name: The user needs to input an item name in case of Delete, Insert and Update queries.

- Method Name: The user needs to input a method name and if required, the user can provide parameters to the methods as well.

- Attribute and Attribute Value: The attribute name is similar to the column name. As SimpleDB does not have any schema, the user needs to input attributes with its value.

Once the user provides the inputs and clicks on the designated button, code will be generated using C# API of Amazon SimpleDB. This will help the developer by directly using the code in their migrated application. Moreover, it will assist a developer to understand and query against the domains in SimpleDB.

Figure 40 present a code generation interface.



Select Domain: author_denormalized_dat

Item Name: 15

Method Name: fn_Insert(Int a, String b)

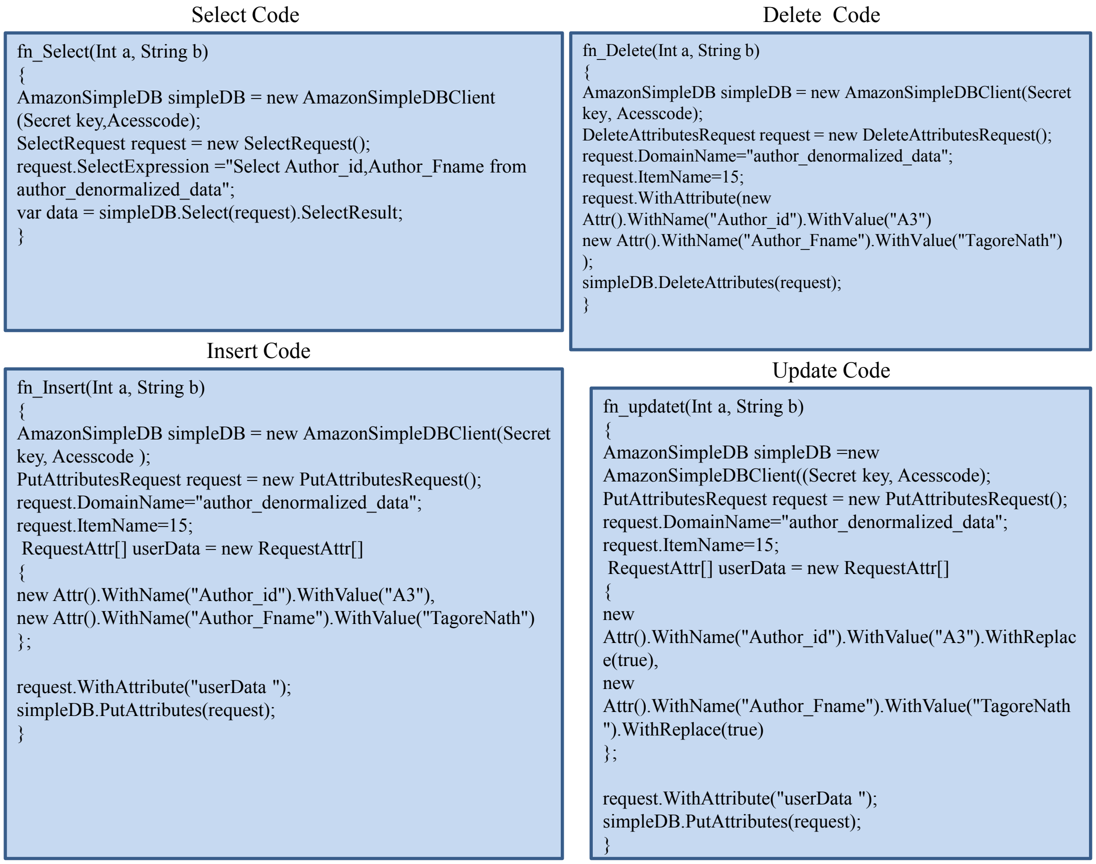
Attributes Input Value

Attribute Name	Attributevalue
Author_id	A3
Authorname	TagoreNath

Figure 40: Code generation interface

Suppose firstly, we need to fetch Author_id, Author_Fname from "**author_denormalized_data**"domain presented in the Type 4 Migration approach. Secondly we need to perform Delete, Update and Insert operations on the domain "**author_denormalized_data**"having attributes Author_id, Author_Fname. Once the user clicks on the desired button, the respective code is generated. The user can directly use this code in the "Data Access Layer"of its application. This code will greatly assist the user while migrating application code from the relational database to the Amazon SimpleDB.

Figure 41 presents a sample code for Select, Insert, Delete and Update operations.



Select Code

```
fn_Select(Int a, String b)
{
AmazonSimpleDB simpleDB = new AmazonSimpleDBClient
(Secret key,Acesscode);
SelectRequest request = new SelectRequest();
request.SelectExpression ="Select Author_id,Author_Fname from
author_denormalized_data";
var data = simpleDB.Select(request).SelectResult;
}
```

Insert Code

```
fn_Insert(Int a, String b)
{
AmazonSimpleDB simpleDB = new AmazonSimpleDBClient(Secret
key, Acesscode );
PutAttributesRequest request = new PutAttributesRequest();
request.DomainName="author_denormalized_data";
request.ItemName=15;
 RequestAttr[] userData = new RequestAttr[]
{
new Attr().WithName("Author_id").WithValue("A3"),
new Attr().WithName("Author_Fname").WithValue("TagoreNath")
};

request.WithAttribute("userData ");
simpleDB.PutAttributes(request);
}
```

Delete Code

```
fn_Delete(Int a, String b)
{
AmazonSimpleDB simpleDB = new AmazonSimpleDBClient(Secret
key, Acesscode);
DeleteAttributesRequest request = new DeleteAttributesRequest();
request.DomainName="author_denormalized_data";
request.ItemName=15;
request.WithAttribute(new
Attr().WithName("Author_id").WithValue("A3")
new Attr().WithName("Author_Fname").WithValue("TagoreNath")
);
simpleDB.DeleteAttributes(request);
}
```

Update Code

```
fn_updatet(Int a, String b)
{
AmazonSimpleDB simpleDB =new
AmazonSimpleDBClient((Secret key, Acesscode);
PutAttributesRequest request = new PutAttributesRequest();
request.DomainName="author_denormalized_data";
request.ItemName=15;
 RequestAttr[] userData = new RequestAttr[]
{
new
Attr().WithName("Author_id").WithValue("A3").WithReplac
e(true),
new
Attr().WithName("Author_Fname").WithValue("TagoreNath
").WithReplace(true)
};

request.WithAttribute("userData ");
simpleDB.PutAttributes(request);
}
```

Figure 41: Select, Insert, Delete and Update code generation.

Chapter 6

Performance Evaluation

In this chapter, we empirically analyze the computation cost of different migration strategies. We propose a “**performance model**”to evaluate different migration methods. The “**performance model**”consists of computational time, and storage costs.

6.1 Experiment Setup

In order to evaluate the performance of our protocol, we use a laptop equipped with an Intel i3 processor at 2.20GHz and 6GB of RAM running on windows 7 Home Premium. The upload speed is at an average 22Mbps and the download speed is at an average 19Mbps. We used a virtual machine (VM) running SQL Server on Amazon Elastic Cloud Compute (EC2) as the cloud with a micro instance. The Amazon

SimpleDB charges based on the amount of machine capacity used to complete the particular request (e.g., SELECT, GET, PUT), normalized to the hourly capacity of a circa 2007 1.7GHz Xeon processor. All the experiments are done under the same conditions.

6.2 Performance Model

The performance model consists of computational time and storage cost. Figure 42 presents our "**performance model**".

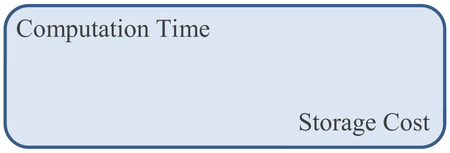


Figure 42: Performance model

6.2.1 Computational Time

We evaluate the average computation time of fetching 25 records, 50 records and 100 records in different migration methods from Amazon SimpleDB. We fetch the same attribute-value pairs in all the different approaches to evaluate computational time in each migration method. We execute the queries four times to fetch the same number of the records. Figure 43 presents the result of fetching the different number of records.

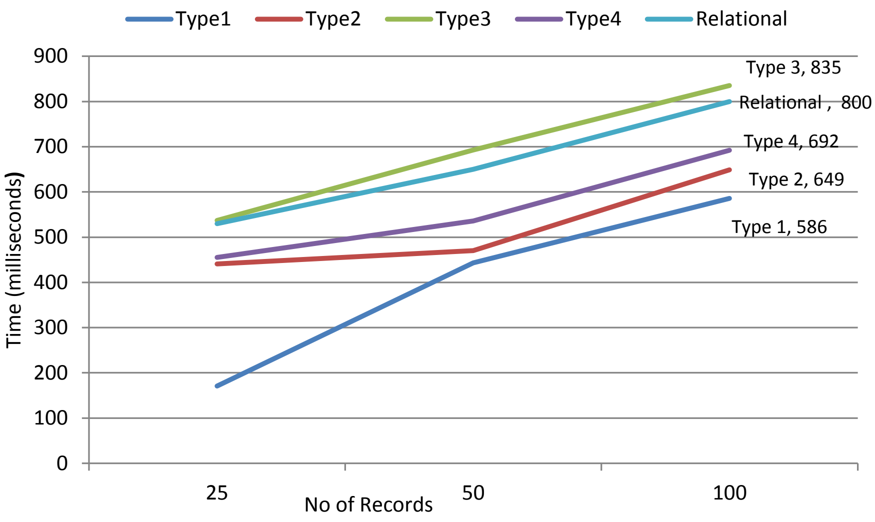


Figure 43: Computational time of fetching records

We use the *online Bookstore schema* as presented in Figure 13 to fetch these records. Upon a request from an authorized user, the Amazon SimpleDB returns back the result. As shown in Figure 43, the average computation time varies from 171 to 735 ms upon requesting data in the Type 1 Migration Method when the number of records changes from 25 to 100. In the Type 2 Migration Method, the average computation time varies from 441 ms to 649 ms when the number of records changes from 25 to 100. The average computation time in the Type 3 Migration Method varies from 537 ms to 835 ms. In the Type 4 Migration Method, the average computation time varies from 455 to 692 ms on fetching 25 to 100 records. The Type 1 Migration Method took the least amount of time while the Type 3 Migration Method took the maximum time. This happens because in the Type 1 Migration Method, data is fetched from only one domain and in the Type 3 Migration Method, data is fetched from multiple domains. The Type 3 Migration Method queries the records from different domains at the same time. The Type 2 Migration Method and Type 4 Migration Method have nearly the same time because in both types, relevant tables are combined to form one domain based on semantics. The records are fetched from one domain in which multiple tables are combined and from other domains as well. This restricts the query to only one domain and increases the performance and decreases the query time. We also fetch the same number of records from the Amazon RDS of SQL Server to compare the results with our migration methods. Figure 43 shows that the average Computation time of fetching records in Type1, Type2 and Type4 approaches is less as compared to fetching the records in the relational database. The average computation time of fetching records in Type3 method and in a relational database is nearly same.

Table 10: Average Computation time (milliseconds) in different approaches

Number of Records	Type 1	Type 2	Type 3	Type 4
25	171	441	537	455
50	443	470	693	536
100	586	649	835	692

6.2.2 Storage Cost

Every request made to SimpleDB returns a BoxUsage value as a part of the response message. This response message includes the usage of system resources by specific operations. BoxUsage is a measure of machine hour usage by making requests to

SimpleDB. The charges are applied to the hourly capacity of a 2007 1.7 GHz Xeon processor.

The cost of renting storage space in SimpleDB is divided into three categories.

- Structure Data: Structure data consists of a number of items, the average number of attributes per item and the total size of attribute values stored by a domain.

- Machine Utilization: This consists of the number of batchputs, the average number of items per batchput, the number of gets and the number of simple selects made to the domain.

- Data Transfer: Data transfer consists of data transfer out and data transfer in from the domains.

First, we calculate renting 10 GB of space in each migration method of migrating relational database "**online Bookstore schema**"as presented in figure 29. Each table has an average of 5 columns.

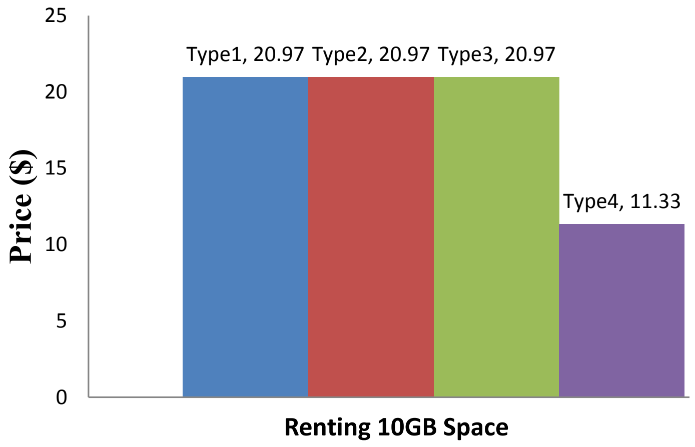


Figure 44: Storage cost of 10GB data

Figure 44 presents the cost of renting 10GB of data in SimpleDB. The Type 1 Migration Method supports storing data up to 10GB because one domain can only extend up to 10GB. The Type 1, Type 2 and Type 3 data migration methods cost the same amount of $20.97, while the Type 4 Migration Method costs $11.33. The Type 4 Migration costs less because the machine utilization in terms of number of batchputs, average number of items per batchput decreases. Also, storing of structural data in terms of numbers of items decreases. Both machine utilization and storing structured data reduce the overall cost.

As the data increases beyond 10GB, Type 1 migration does not provide support. Figure 45 presents the result of renting storage of 25GB data.

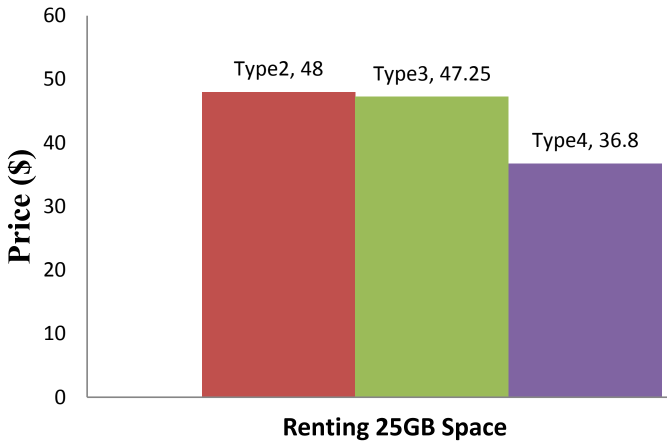


Figure 45: Storage cost of 25GB data

The Type 2 Migration Method and the Type 3 Migration Method cost the nearly same as $48 and the Type 4 Migration Method costs $36.8. Our results shows that storing data saves costs in the Type 4 Migration Method as compare to other data migration methods. The cost of renting Amazon RDS of SQL Server is costly as compare to Amazon SimpleDB.

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

Over the last decade, cloud has been a successful paradigm for web applications. The popularity of cloud computing is increasing and emerging as a billion-dollar industry. DBMSs store and serve data for an application, hence data becomes critical and central to a web application.

The overarching goal of this thesis is to propose a model, methods and paradigms to develop a system which migrates relational databases to cloud databases. This thesis provides techniques and a model which will help software industries to migrate their existing relational databases and data models to the cloud. We propose four diverse methods to migrate relational databases to cloud databases:

- Type 1: complete relational database to one domain.
- Type 2: multiple tables to one domain.
- Type 3: A table to one domain.
- Type 4: normalization to denormalization and tables to domain

Each method is independent of the other. Joins in relational databases combine records from two or more tables but cloud databases lacks joins. This thesis also provides a way of handling joins in each approach. Finally, we propose an interface which generates code with respect to cloud API and helps in code re-factoring during application migration to the cloud. The interface will assist the developer to generate code which includes the usage of basic Select, Insert, Delete and Update queries. This code will assist greatly while migrating application code from relational databases to Amazon SimpleDB.

7.2 Future Direction

Apart from the advantages of this model, it also has limitations. Firstly, stored procedures are not supported by cloud databases (SimpleDB), so we cannot migrate existing stored procedures of the databases to the cloud database. Our future research will focus on migrating the stored procedures from relational databases, if any support is provided by cloud databases. Secondly, the cloud database does not provide any support for user-defined functions and triggers, so we cannot migrate existing functions and triggers of relational databases to cloud databases. Our future research will focus on migrating the user-defined functions and triggers from relational databases, if any support is provided by cloud databases.

Bibliography

[1] M. Abdelsalam, A. Akhtar, and N. Rossiter. A framework for relational database migration. `http://computing.unn.ac.uk/staff/cgma2/papers/RDBM.pdf`. Retrieved July 23, 2013.

[2] Amazon. Amazon simpledb. `http://aws.amazon.com/simpledb/`. Retrieved June 28, 2013.

[3] B. Bordbar, D. Draheim, M. Horn, I. Schulz, and G. Weber. Integrated model-based software development, data access, and data migration. In *Model Driven Engineering Languages and Systems*, pages 382–396. Springer, 2005.

[4] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

[5] Andre Calil and Ronaldo dos Santos Mello. Simplesql: a relational layer for simpledb. In *Advances in Databases and Information Systems*, pages 99–110.

Springer, 2012.

[6] R. Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[7] P. Chaganti and R. Helms. *Amazon SimpleDB Developer Guide*. Packt Publishing Ltd, 2010.

[8] S. Chandrasekaran and R. Bamford. Shared cache-the future of parallel databases. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, volume 1063, pages 17–00, 2003.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th conference on usenix symposium on operating systems design and implementation - volume 7*, pages 205–218, 2006.

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[11] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.

[12] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings VLDB Endow.*, 1(2):1277–1288, August 2008.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[15] C. Dede. The evolution of information technology: Implications for curriculum. *Educational Leadership*, 7(1):23–26, 1989.

[16] D. J. DeWitt, R. H. Gerber, G. Graefe, M. Heytens, K. Kumar, and M. Muralikrishna. *A High Performance Dataflow Database Machine*. Computer Science Department, University of Wisconsin, 1986.

[17] C. Drumm, M. Schmitt, H. Do, and E. Rahm. Quickmig: automatic schema

matching for data migration projects. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 107–116. ACM, 2007.

[18] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *ACM SIGOPS Operating Systems Review*, volume 31, pages 78–91. ACM, 1997.

[19] S. Fushimi, M. Kitsuregawa, and H. Tanaka. An overview of the system software of a parallel relational database machine grace. In *Proc. Intl. Conf. on Very Large Databases*, 1986.

[20] K. Haller. Data migration project management and standard software–experiences in avaloq implementation projects. In *Data Warehousing Conference-DW2008: Synergien durch Integration und Informationslogistik*, pages 391–406, 2008.

[21] K. Haller. Towards the industrialization of data migration: Concepts and patterns for standard software implementation projects. In *Advanced Information Systems Engineering*, pages 63–78. Springer, 2009.

[22] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.

[23] J. Henrard, M. Hick, P. Thiran, and J. Hainaut. Strategies for data reengineering. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, pages 211–220. IEEE, 2002.

[24] A. R. Hickey. Cloud computing market hot, but how hot? estimates are all over the map. `http://www.forbes.com/sites/joemckendrick/2012/02/13/cloud-computing-market-hot-but-how-hot-estimates-are-all-over-the-map/`. Retrieved July 23, 2013.

[25] P. Howard and C. Potter. Bloor research: Data migration in the global 2000 - research, forecasts and survey results. `http://www.techrepublic.com/whitepapers/bloor-research-data-migration-in-the-global-2000-research-forecasts-and-survey-results/322625`.Retrieved March, 2013.

[26] Andrei Idu. Data migration for data intensive software products. 2012.

[27] S. Islam. Data migration: Connecting databases in the cloud. *The Second International Conference on Communications and Information Technology*, 2012.

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

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

[30] J. W. Josten, C. Mohan, I. Narang, and J. Z. Teng. Db2's use of the coupling facility for data sharing. *IBM Systems Journal*, 36(2):327–351, 1997.

[31] D. Kargerand, E. Leighton, T. Panigrahy, R. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[32] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 579–590, 2010.

[33] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[34] B. G. Lindsay, L. M. Haas, C. Mohan, F. P. Wilms, and A. R. Yost. Computation and communication in r*: A distributed database manager. *ACM Transactions on Computer Systems (TOCS)*, 2(1):24–38, 1984.

[35] D. B. Lomet, R. Anderson, T. K. Rengarajan, and P. Spiro. *How the Rdb/VMS data sharing system became fast.* Citeseer, 1992.

[36] A. Maatuk, A. Ali, and N. Rossiter. Relational database migration: A perspective. In *Database and Expert Systems Applications*, pages 676–683. Springer, 2008.

[37] F. Matthes and C. Schulz. Towards an integrated data migration process model. *Software Engineering for Business Information Systems (sebis)*, 2012.

[38] R. Rawson and J. Gray. Hbase at hadoop world nyc. `http://bit.ly/HBase_HWNYC09`. Retrieved November 19, 2012.

[39] J. Rothnie, B. James, P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, A. T. Landers, C. Reeve, D. W. Shipman, and E. Wong. Introduction to a system for distributed databases (sdd-1). *ACM Transactions on Database Systems (TODS)*, 5(1):1–17, 1980.

[40] P. Russom. Best practices in data migration. *Renton/USA*, 2006.

[41] H. G. Sockut and P. R. Goldberg. Database reorganization—principles and practice. *ACM Computing Surveys (CSUR)*, 11(4):371–395, 1979.

[42] 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 430–434, New York, NY, USA, 2010. ACM.

[43] W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.

[44] Wikipedia. Data_migration. `http://www.en.wikipedia.org/wiki/Data_migration`. Retrieved June 26, 2013.

[45] Wikipedia. Denormalization. `http://www.en.wikipedia.org/wiki/Denormalization`. Retrieved August 29, 2013.