# Data Sharing in Data-Centric Multi-Tenant SaaS Application


**Submitted By**

Usman Aslam

2010-NUST-MS PhD-IT-22

**Thesis Supervisor**

Dr. Hamid Mukhtar

**Committee Members**

Dr. Sharifullah Khan

Dr. Zahid Anwar

Bilal Ali

**Department of Computing (DoC)**

**School of Electrical Engineering & Computer Science (SEECS)**

**National University of Sciences & Technology (NUST),**

**Islamabad, Pakistan**

# Approval

This thesis has been submitted in partial fulfillment of requirements for the Master of Information Technology at National University of Sciences & Technology.

It is certified that the contents and form of the thesis entitled "**Data Sharing in Data-Centric Multi-Tenant SaaS Application**" submitted by **Usman Aslam** have been found satisfactory for the requirement of the degree.

**Advisor:** **Dr. Hamid Mukhtar**

**Signature:** _____

**Date:** _____

**Committee Member 1:** **Dr. Sharifullah Khan**

**Signature:** _____

**Date:** _____

**Committee Member 2:** **Dr. Zahid Anwar**

**Signature:** _____

**Date:** _____

**Committee Member 3:** **Bilal Ali**

**Signature:** _____

**Date:** _____

# Dedication

To Dr. Hamid Mukhtar

# Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at National University of Sciences & Technology (NUST) School of Electrical Engineering & Computer Science (SEECS) or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Usman Aslam

**Signature:** _____

# Acknowledgement

This thesis would not have been possible without the continuous support of my supervisor **Dr. Hamid Mukhtar**.

Thanks to my brother **Muhammad Tariq** for proof reading of this dissertation for any grammatical mistakes.

Also thanks to my committee members, **Dr. Sharifullah Khan**, **Dr. Zahid Anwar**, and **Mr. Bilal Ali**, for their guidance and support.

And finally, thanks to my family, and numerous friends who endured this long process with me, always offering support and love.

# Table of Contents

# List of Abbreviations

| **Abbreviation** | **Stands for** |
|---|---|
| SaaS | Software as a Service |
| DAO | Data Access Object |
| DAL | Data Access Layer |
| VM | Virtual Machine |
| DBMS | Database Management System |
| RDBMS | Relational Database Management System |
| JDK | Java Development Kit |
| RAM | Random Access Memory |
| MB | Megabyte |
| GB | Gigabyte |

# List of Tables

# List of Figures

# List of Graphs

## Abstract

SaaS applications are deployed on a shared environment that can be accessed by the users from client-end software by using the Internet. Organizations using SaaS applications do not have control over the infrastructure. SaaS applications are built using multi-tenant system architecture. Multi-tenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client organizations (tenants). Multi-tenant applications provide a common UI for all the organizations and data of multiple tenants is saved in a single database.

The problem in existing standards of multi-tenant SaaS applications is that data have to be migrated from one tenant to another for many reasons like mergers, joint marketing campaign, moving from pilot to production SaaS instance. Data migration requires a very skilled and time consuming human effort and results in data duplication.

The suggested solution will share data between organizations rather than making a copy of data for each organization, thus reducing human effort in data migration and eliminating data duplication.

The contribution of this thesis are several folds: 1) middleware for data sharing between different organizations; 2) a design pattern and algorithm for the implementation of the middleware; 3) extension to current multi-tenant data-centric models for SaaS applications; 4) identification of constraints; and 5) suitability of proposed solution for existing SaaS applications.

Results show that data can be shared between different tenants of a SaaS efficiently and accurately without making any to presentation and business layer in existing architecture of SaaS.

The suggested solution shares all the data of one organization with another organization. Restricted data sharing between organizations can be an extension of the suggested solution and basis for future work.

# 1. Introduction

A Cloud service is a collection of interconnected computing resources that are provisioned dynamically and presented as unified computing resource based on agreement between the provider and consumer. Clouds mostly have multi-tenant system architecture. In this architecture different applications of organizations using cloud services are organized in a single environment that is partitioned logically.

Cloud computing delivers computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a metered service over a network.

Cloud computing is based on the decades of research in networking, virtualization, distributed computing, load balancing, and utility computing. It's an offspring of service-oriented architecture, have reduced information technology knowledge and management overhead for the end-user, provides great flexibility in terms availability of computing resources, total cost of ownership of computing resources is much lesser than on-premise computing model, and is based on on-demand services model as well. Cloud computing provides several general service models like Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) as shown in Figure 1-1.

Most of the SaaS applications are data-centric and revolve around managing business data. Therefore databases have a vital role. In SaaS, data of multiple organizations is saved in a single database and is virtually partitioned from each other [1]. Configurability maturity level of SaaS is determined by the ability provided by SaaS to configure its data, user interface, and logic. There are four levels of SaaS maturity. Data, logic, and user interface is configurable and customizable at second, third and, fourth level [2] and end users of SaaS can add virtual or physical columns depending on model. This kind of SaaS customization makes data structure of each tenant heterogeneous even the data saved in a single database.

## 1.1 Limitation of Current Standards

Some SaaS applications provide data import and export facility to move data from one tenant to another and most of the SaaS like Gmail don't. For example if a person has two email addresses

on two different domains registered at Gmail, there is no way to see emails from both email addresses from single view. He has to create a forwarding rule in one email address to see emails of both email addresses from one email address. Email forwarding makes his life easier to view all the emails without navigating back and forth between the email addresses. This email forwarding duplicates data as there are two copies of the same email in each email address.

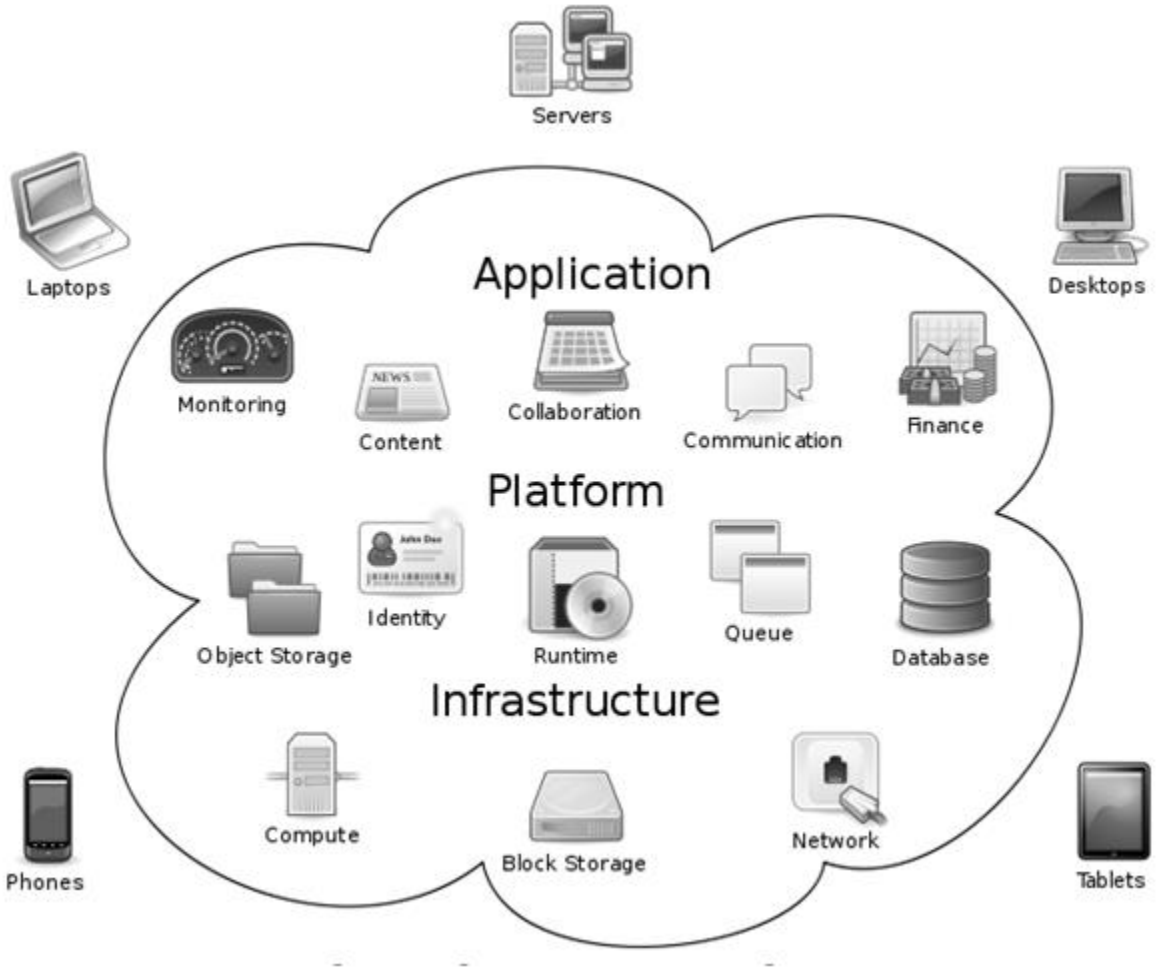



Figure 1-1: Cloud Computing [3]

Similarly, if two different companies were using same SaaS application and now one of them is now acquired by the other. What they should do now? Import data to other tenant or re-create it manually or login in each tenant to complete work or there should be a facility in SaaS application to share data between multiple tenants without duplicating it?

## 1.2 Significance of Problem

In case of mergers, joint marketing campaigns, migrating customizations from DEV tenant to QA tenant [4], upgrading from pilot SaaS instance to production SaaS instance, we have to migrate data from one tenant to another. Data migration requires a very skilled and time consuming human effort and it results in data duplication. If DBMS supports data-sharing as in [5], the application still needs to be re-configured to share data between different tenants.

## 1.3 Objectives

The objective of this research is to provide a valid/practical proof of concept of all the multi-tenancy models while taking into consideration that this data-sharing / extensions of existing models will not affect the performance / efficiency of the SaaS application.

This research will also identify the constraints for data-sharing in SaaS environment and suitability of proposed models for existing SaaS applications.

The result of this thesis will be multi-tenancy model supporting data sharing and this model will be implemented as a middleware between application controller and data access layers.

## 1.4 Motivation

Motivation of this research is to find an optimal solution at application level that allows sharing data between tenants. Hence we can eliminate the need for data migration between the tenants of the same application, thus avoiding data duplication and saving storage resources.

In this thesis an algorithm has been suggested and a middleware that will allow an application to share data of one tenant with another tenant without affecting the overall performance of the application. To incorporate the suggested middleware in the existing application, SaaS doesn’t have to make significant change in the architecture of the application. Our results shows that data retrieved from different sources can be shared accurately without affecting overall performance of SaaS.

## 1.5 Organization

In the following sub-sections, briefly explain the basic concepts related to this thesis. Chapter 2 elaborates the problem in data-centric multi-tenant architectures. After that, work done or research already been carried out on multi-tenant architectures and data sharing in multi-tenant environment is discussed. Chapter 4 presents a formal problem statement and challenges associated with that problem statement that needs to be addressed in data sharing while Chapter 5 provides details about the suggested solution. In Chapter 6 discusses the evaluation environment and impact of data sharing on the performance of application in a shared, distributed, and heterogeneous environment. In the last chapter concludes this thesis and provides future direction for data sharing in a multi-tenant SaaS.

## 1.6 Background

At first it is necessary to get a basic overview of the evolution of distributed computing, its first encounter and how it developed. It is also certain to clearly explain what is cloud computing, what are the concepts behind it and how it different from all the other concepts.

### 1.6.1 Utility Computing

Utility Computing is a business model in which computing resources like physical storage media and computational resources are bundled as metered services like electricity and telephone. For utility computing monitoring and accounting services are applied to computing infrastructure like Grids which can be utilized privately and publically [6].

The idea of providing computational resources as a service came to into real existence, utility computing got realized in cloud computing.

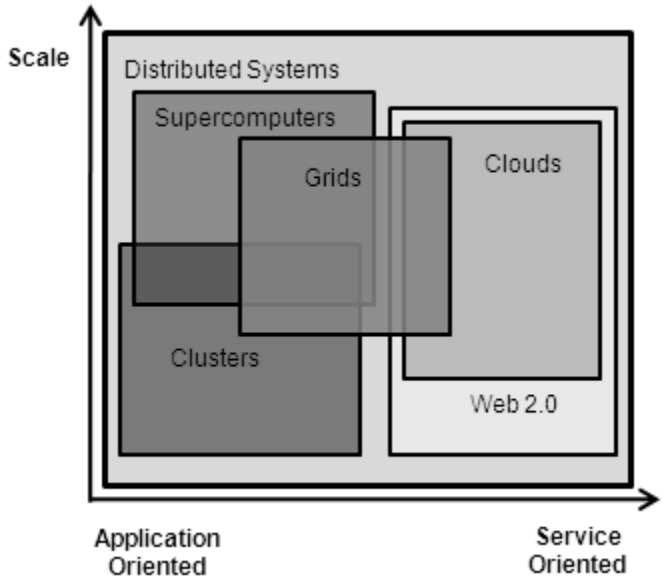



Figure 1-2: Grid Computing [6]

### 1.6.2 **Distributed Computing**

Distributed computing is a virtual computing infrastructure having heterogeneous computing devices networked together agreed to share their resources to process a common job or task [7]. Computing devices can be of same or different types, located across the globe or in a single building. Supercomputers, clusters, clouds, web 2.0 and grids are subsets of distributed systems as shown in Figure 1-2 [6].

Distributed is used when a single computer cannot perform a particular computation in a reasonable amount of time.

#### 1.6.2.1 Clusters

Clusters are made up of linked computers having similar kind of hardware and software and are normally located in a single building. Clusters are designed for specialized purpose

#### 1.6.2.2 **Supercomputers**

Supercomputers follow the concept of clusters except the processing units are merged in single box. The architecture of supercomputers is implemented using highly-tuned computer clusters with thousands of commodity processors intercommunicating with custom interconnects [8].

#### 1.6.2.3 Grids

Grids involve different type of computing devices that are connected to each other and are dispersed across the globe. The hardware and software on computing devices can also be different. Grids can be used for variety of purposes as compared to cluster which are designed for specialized purpose.

#### 1.6.2.4 Clouds

A Cloud is a collection of interconnected computing resources that are provisioned dynamically and presented as unified computing resource based on agreement between the provider and consumer.

**Public Cloud:** Public cloud is a cloud that is made available to general public on pay-as-per-use basis [9].

**Private Cloud:** Private cloud is opposite of public cloud. It refers to datacenters that are not available to general public and are used by the organization that owns these datacenters [9].

**Hybrid Cloud:** Hybrid cloud is comprised of both public and private cloud. Hybrid cloud use public clouds when private cloud is not capable to performing the computation due some limitation or workload [9].

### 1.6.3 **Supporting Technologies**

Multi-tenancy, Virtualization and Load Balancing are three technologies that helped tremendously in realization of cloud computing.

#### 1.6.3.1 **Multi-tenancy**

Authors of [10] explain that multi-tenancy is an architecture in which different applications of the organizations using cloud services are organized in a single environment that is partitioned logically to achieve economies of scale and optimization in terms of maintenance, data security, high availability, disaster management, and speed.

#### 1.6.3.2 **Virtualization**

Virtualization is completely software based architecture that provides the illusion of a real machine to all software running above it. Virtualization is the result of the idea that adds a layer between the user of a computing environment and underlying hardware to provide flexibility. Virtualization improves scalability and overall resource utilization and provides facility of administration of virtual environment [11]. Virtualization has a great contribution in the field of pay-as-per-use Computing [12].

#### 1.6.3.3 **Load Balancing**

Load balancing has a great importance in cloud-related deployment models [13]. It is a process that evenly distributes the tasks among different nodes of the distributed system in order to improve response time of the task and fair usage of the computational resources. The process of load balancing is also responsible to avoid situations where a node is used for major portion of the tasks while others nodes are either working on small portion of the tasks or idle [14].

### 1.6.4 **Cloud Computing**

When comparing the cloud idea to the existing clusters or supercomputers, it is obvious that clouds are located at dispersed location and is made up of different and unknown networks [6] as compared to current standards of clusters and supercomputers.

According to authors of [6] cloud and grid computing are the same theoretically, which is decreasing the computing costs and increasing the scalability, reliability, and availability of services at the same time.

Due to huge investments in virtualization by large companies such as Salesforce.com, Facebook, and Oracle, there are “large commercial systems containing thousands of computers” [6]. In other words, cloud computing has taken distributed computing to another level.

Essence of cloud computing lies in multi-tenancy that is using a single infrastructure for multiple organizations to gain economy of scale, thus reducing cost for adopting clouds.

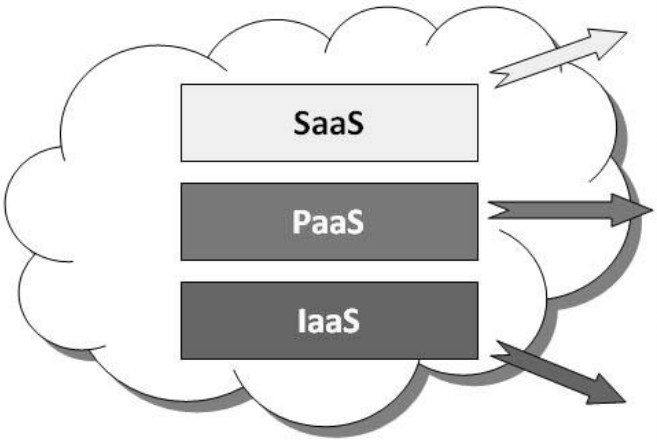
### 1.6.4.1 **Cloud Service Models**

Cloud computing provides several general service models like Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) as shown in Figure 1-3.

**Infrastructure as a Service (IaaS):** It is the service provided to organizations in the shape of virtual machines, operating systems, message queues, network, storage, memory, processing units, and backup services on demand and rental basis so the organizations can deploy their applications and services on the cloud instead of purchasing these computing infrastructures resources. Amazon is the typical example of this kind of cloud model.

**Platform as a Service (PaaS):** This service model of cloud provides facility for application development, integration, deployment, testing and operation. The whole software environment is hosted on the cloud.

Google AppEngine and Force.com are the examples of PaaS. Developers can make multi-tenant applications easily and these application run at datacenters of provider. Security, backup, and maintenance are also the responsibility of the provider.




| Who Uses It | What Services are available | Why use it? |
|---|---|---|
| Business Users | EMail, Office Automation, CRM, Website Testing, Wiki, Blog, Virtual Desktop ... | To complete business tasks |
| Developers and Deployers | Service and application test, development, integration and deployment | Create or deploy applications and services for users |
| System Managers | Virtual machines, operating systems, message queues, networks, storage, CPU, memory, backup services | Create platforms for service and application test, development, integration and deployment |

Figure 1-3: Cloud Computing Service Models [15]

**Software as a Service (SaaS):** Business user can use software like Email, CRM, and Virtual Desktop etc that made available online to complete business tasks.

Google Apps is an alternative to on-premise office suite and is a typical example of SaaS.

### 1.6.5 SaaS Maturity Models

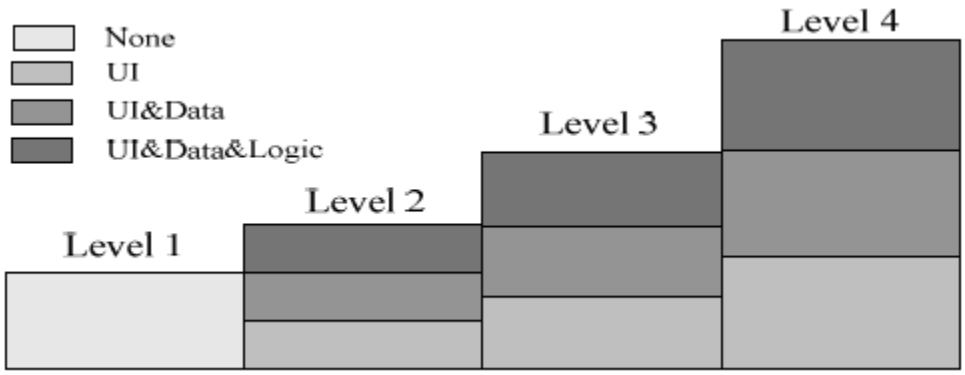



Figure 1-4: SaaS Maturity Model [2]

There are four maturity models of SaaS. At Level 1, SaaS doesn't provide customization at all. While data, logic, and user interface is configurable and customizable at second, third and, fourth level. At Level 4, the ability to customize and configure data, logic, and user interface is more than Level 3, and Level 2. SaaS Maturity Model is shown in Figure 1-4.

# 2. Research Problem and Its Significance

In this chapter the problem is explained by its context, and real world examples. After the explanation of the problem, challenges involved in finding the solution for the problem are discussed.

## 2.1 Context of the Problem

Some SaaS applications provide data import/export facility and most of the SaaS application lie Gmail don't.

### 2.1.1 Applications with Data Import/Export Facility

As we know that resources are always limited. Import/export of data has following problems;

- Data duplication; export data from one tenant of that application and import it to another tenant.
- Effort; import/export of data is not straight forward; we have to take care of relationship between data. For example, if we will import data in a tenant, the imported data will have new IDs for each record. So, we have to take care of each child record to have new ID of the parent record.

### 2.1.2 Applications without Data Import/Export Facility

In this case the end user will have to manually re-create the data in the target tenant/organization or development team will do this at the back-end using xls/csv/xml files or the end user will have to login in multiple tenants to do their work.

If there is a facility of data sharing in a SaaS we can overcome the limitations/problems stated above.

## 2.2 Real World Examples

Below are few practical examples in which we have to move data from one tenant to another.

### 2.2.1 Mergers

Two different companies were using same SaaS application and now one of them is now acquired by the other. What they should do now? Import data to other tenant or re-create it manually or login in each tenant to complete work or there should be facility in SaaS application to share data in multiple tenants without duplicating it?

### 2.2.2 Joint Marketing Campaign

For instance, two companies have agreed to use each other client data to run a marketing campaign. For this marketing campaign, the same questions arise as in 3.2.1.

### 2.2.3 Moving from Staging to Production SaaS instance

Staging instance is a tenant who has real data and the organization provides access to few users because of limited licenses. On the other hand, production instance has unlimited number of licenses for each organization. Now an organization has decided to upgrade its instance in a SaaS application. In this scenario it will be quite beneficial for organization to share its data in pilot tenant with a new tenant of the same SaaS application.

### 2.2.4 Moving from Developer to QA SaaS instance

In Salesfroce.com – a well known cloud-based CRM – whenever an organization buys its licenses, the organization is given a stack of (DEV, QA, Production etc) tenants of the application [4]. Developers do their work in DEV tenant and then deploy customizations to QA. QA department tests it in QA tenant and approve it for deployment to Production server.

For developer testing, data is created in DEV tenant, then QA department same sort of data to test the customizations. By this way, two tenants (DEV, QA) have the same data and effort is involved in re-creating the same data in QA tenant. If we can share data in these two instances of the same organization, we can eliminate the efforts required for re-creation of data.

## 2.3 Research Challenges

Following are few challenges that are associated to the problem and its solution.

### 2.3.1 Distributed Environment

In a distributed environment, the application and database servers will or can be at different locations and data of different tenants will or can be at different DBMSs. In such an environment, application needs to be changed to retrieve data from different DBMSs efficiently to share data of one tenant with another.

### 2.3.2 Heterogeneous Environment

Application and database servers can be of different types. For example, one tenant's data is residing at a structured database server like Oracle and other tenant's data is at unstructured database like SimpleDB. In such an environment, sharing data retrieved from different types of DBMSs will not be straight forward.

### 2.3.3 Heterogeneous Data Structure

[16, 17, 18, 19] have explained different multi-tenant database architectures for SaaS that allows end user to customize the SaaS depending on their needs. Due to these end user customizations, table names, column names, and columns data type can be different between tenants. In such a scenario data needs to be transformed and merged before any further processing.

### 2.3.4 Data Accuracy and Integrity

After data transformation, there is a strong chance that data lose its accuracy and integrity. The solution must be robust enough to share data without losing its integrity and accuracy.

# 3. Related Work

This chapter discusses existing multi-tenant models and data sharing techniques for data-centric applications. While data integrity and accuracy techniques for outsourced data is discussed in the last part of this chapter.

## 3.1 Data-Centric Multi-Tenant Models

There are several multi-tenant models for data-centric application that virtually partition data of tenants in the same database. In the following sub-sections these models are discussed.

### 3.1.1 Metadata Driven Model

[16] is based on multi-tenant architecture that provides one application for different set of users. Developers can also customize the existing service or develop their own services using same platform. All custom applications have same platform so these can be integrated to manage data efficiently. This multi-tenant application platform ensures that data will be reliable, easily customizable, upgradable, and secure. It is very difficult to achieve when applications are statically executed so it needs to execute in dynamic in nature.

For this purpose a runtime engine is designed for multi-tenant applications that use metadata for application components generation. Metadata is based on an architecture in which runtime engine is isolated from metadata and application data. So it uses this metadata driven architecture, it provides more customizability to user and provide scalable and high performance multi-tenant applications which secures data of one tenant from the other tenants. It provides complete accessibility to application components and metadata to read and customize it according to the requirements. All the metadata is stored in universal data dictionary. In this architecture SaaS components definition is stored in metadata rather than in database and runtime engine creates virtual components of application during runtime, so anyone can customize its application. It stores its recently used metadata in a special cache which increases processing and accessibility to data without compilation of application.

This data is saved with an index on it and search services remain active during data manipulation. Due to this index based searching it gives most accurate results to the users. It has

runtime application generator that generates application at runtime and internal execution is done effectively and efficiently by query optimizer.

Metadata is stored in virtual tables and so for keeping all records for users, virtual table uses pivot table, and data is stored in normalize form in pivot tables. All data stores are physically partitioned on the basis of tenant IDs which means that all metadata is store in logically smaller groups which increase the accessibility, availability, and customizability. For this reason search is very efficient and target data can easily be found because it will not search the entire metadata table.

### 3.1.2 **Sparse Table Based Approach**

In this business model all the data is belong to the same instance but referenced by different tenants. As each tenant have different schema and columns, to solve this problem one big table called the sparse table is used. Sparse table is a comprehensive table which contain data for all the tenants. For example there are two tenants “A” and “B”, “A” have 20 columns and “B” with 30 columns so the sparse table will consist of 50 columns. For tenant “A” all the 30 columns of tenant “B” will be mark as null in sparse table and vice versa.

The sparse table approach is similar as of traditional sparse data processing. In both the large amount of nulls causes wastage of space and also effect the performance of query execution. The techniques of traditional sparse data processing such as interpreted attribute storage and vertical schema can also be applied in multi-tenant data storage with multi-tenant characteristics.

There are many analogous points in traditional sparse data and multi-tenant sparse data. For example in both cases the size of sparse table is very large. In case of multi-tenant the number of tenants and their need for are unknown so large number of columns are required to entertained different tenants. The table in both situations is always sparse with respect to their requirements. Table is continuously altered due to undefined and frequent changes in schema. More over outsized number of nulls result in space wastage and query performance degradation.

Though there are similarities in traditional sparse data and multi-tenant sparse data but some characteristics differences also exist due to different historical background. The tenants cause the sparsity in sparse table but there is no concept of tenant in traditional approach. In multi-tenant

sparse table there are two type of null, first one is schema null which indicate that the tenant does not customize the table's columns and used most of the time, the other one is called value null which indicate that the tenant customize the table. As most of the nulls are schema nulls and tenant consume the columns of the sparse data from left to right so it make the table left intensive or right sparse.

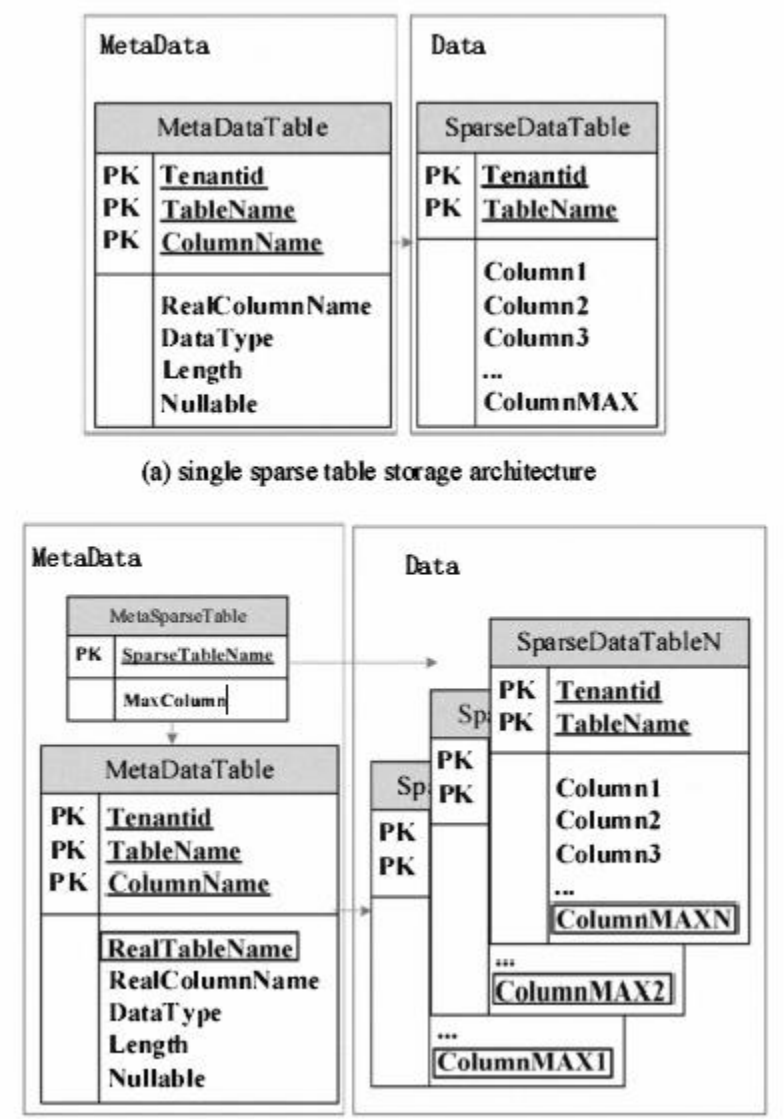



Figure 3-1: Sparse Table Based Approach [17]

Authors of [17] have proposed a multi-tenancy model for SaaS that extends single sparse table technique. In their proposed approach, each table has inclining number of columns and metadata

is stored in another sparse table. This approach helps eliminating null values in each record hence improves storage and query processing.

### 3.1.3 **XML Based Approach**

pureXML by IBM was designed to process XML with relational data. A multi-tenancy model using pureXML is suggested by the authors of [18]. Base columns of each table are shared by each tenant. A special column is there in each table to store data of virtual columns created by the end user. The structure of XML stored in this column varies for each tenant. XML representation is kept as compact as possible to improve performance.

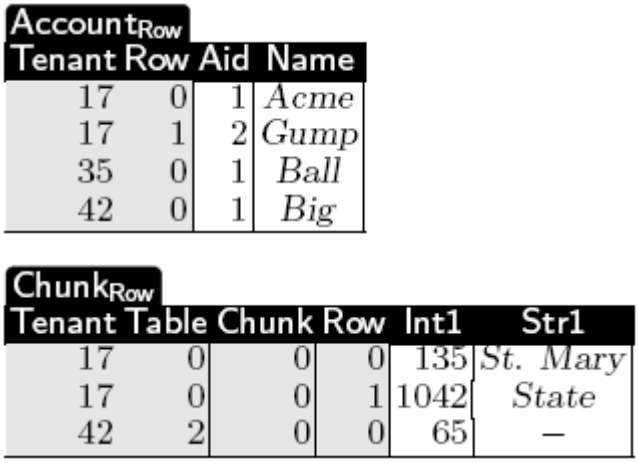
Account

| Tenant | Aid | Name | Ext_XML |
|---|---|---|---|
| 17 | 1 | *Acme* | <ext><hospital>St. Mary</hospital><br><beds>135</beds></ext> |
| 17 | 2 | *Gump* | <ext><hospital>State</hospital><br><beds>1042</beds></ext> |
| 35 | 1 | *Ball* | |
| 42 | 1 | *Big* | <ext><dealers>65</dealers></ext> |

Figure 3-2: XML Based Approach [18]

### 3.1.4 **Partitioned Table Approach**

Aulbach et al. [19] have discussed six different models to implement multi-tenant designs for SaaS databases. They have suggested a new technique for multi-tenancy as well. This new technique is based on creating columnar partitions of the logical schema, and then maps it to the physical schema of the database. The partitions are stored together in physical tables in a multi-tenant database and are joined to process the user request. Performance is enhanced by mapping most utilized portions of the logical schema with physical schema. Figure 3-3 shows the Partitioned Table Approach.



AccountRow

| Tenant | Row | Aid | Name |
|---|---|---|---|
| 17 | 0 | 1 | *Acme* |
| 17 | 1 | 2 | *Gump* |
| 35 | 0 | 1 | *Ball* |
| 42 | 0 | 1 | *Big* |

ChunkRow

| Tenant | Table | Chunk | Row | Int1 | Str1 |
|---|---|---|---|---|---|
| 17 | 0 | 0 | 0 | 135 | *St. Mary* |
| 17 | 0 | 0 | 1 | 1042 | *State* |
| 42 | 2 | 0 | 0 | 65 | – |

Figure 3-3: Partitioned Table Approach [19]

### 3.1.5 **Summary**

In all the data-centric multi-tenant architecture data is partitioned physically or logically based on the tenant ID and provide facility to add virtual or physical columns in a database for SaaS customizations. As data of multiple tenants is saved in single database, the addition of virtual or physical columns by a tenant leads to different data structure for that tenant from other tenants.

## 3.2 Data Sharing Approaches

Following are few data sharing techniques that allow data to be shared between different tenants.

### 3.2.1 **FLEXSCHEME**

In virtually partitioned environment makes it difficult for enterprise applications to develop a lot of important features. The very first feature is to provide support for the master data that needs not to be duplicated but shared in all the virtual organizations to help us reduce the cost. For example, insensitive information about the business bodies like DUNS and supplier performance. Data may be shared in the form of hierarchical mechanism or among the subsidiaries. Whichever the ways is concerned, the organizations must have the ability to modify the data according to their needs and DBMS must keep those changes separate for each of the organization.

Second road block that one may face is modification of application. That is applicable for the data as well as schema. This functionality is mandatory to let the application cop up with needs of every individual business and it may vary according to their demographic characteristics. Extension can have both possibilities separate or shared for virtual organizations but if it is shared, then it needs to be developed by some vendor and purchased as an add-on to the actual application. While basic customizations can be utilized in configurations and wizards. It strictly requires changing schema for the more complex applications.

Third issue is to evolve schema and master data. It requires letting the extensions be upgraded. To upgrade the extensions in such a way that the changes remain separate for each organization, the upgrades should occur only if there is raised any demand. The point also needs to be taken under considerations that to obtain the operation cost, the upgrades should be made so easy so that individual body can do themselves without hiring any consultancy. It is encouraged that the upgrades should be postponed to some convenient time in future. Also this is desirable that both (old and upgraded) systems run parallel in case of any roll back due to some critical issue during up gradation.

The extensions mentioned above contain a hierarchy that can share both master data and configurations. These are composed of the instance objects, schema, and common data. Data sharing help us decrease storage media and easy upgrades. For a virtually partitioned environment, any kind or upgrade or modification must not affect the other organization that's why write access should be kept private. Share data doesn't only interfere with schema but the data itself. This issue can be resolved by introducing versioning in schema and data itself. It also enables organization to stay on some specific version.

All these characteristics do not apply to the traditional old DBMS and today if they are being applied on somewhere that is application layer only. For example Force.com maintains multiple tenants to keep the data separate from the deployment world. But it make the development process more complex as a lot of advanced features or DBMS are used like query optimizations etc that requires to re create the source code rather than using the existing one.

The author of [5] describes FLEXSCHE that is also known as integrated model for virtually partitioned environment. This is extensible evolved and provides sublime data sharing. A

prototype that we have created has optimized query plan to obtain FLEXSCHEME capability. We have efficiently implemented the versioning for the shared data using XOR Delta, made it optimized and effective for the SaaS environment and popularity of latest data is at peak.

Next generation database management systems need to be optimized in hardware usage. Their work shows that DBMSs consumes less energy while using main memory than that of disk memory. And it is a fact main memory is an expensive piece of hardware that needs to be used in optimized way that is why hardware optimization is of vital importance in next generation DBMSs. And this can be achieved by vacating the main memory straight way after the data has been used. That may not sound a big deal but in terms of virtually partitioned systems it matters a lot to identify which instance is being used by which virtual system so that it can be flushed after the purpose of the organization is achieved.

The limitation of model in [5] is that the main-memory DBMSs are not suitable for a large SaaS. Even if DBMS supports data-sharing, the application needs to be re-configured to accommodate data from different tenants.

### 3.2.2 **Multiple Copies Approach**

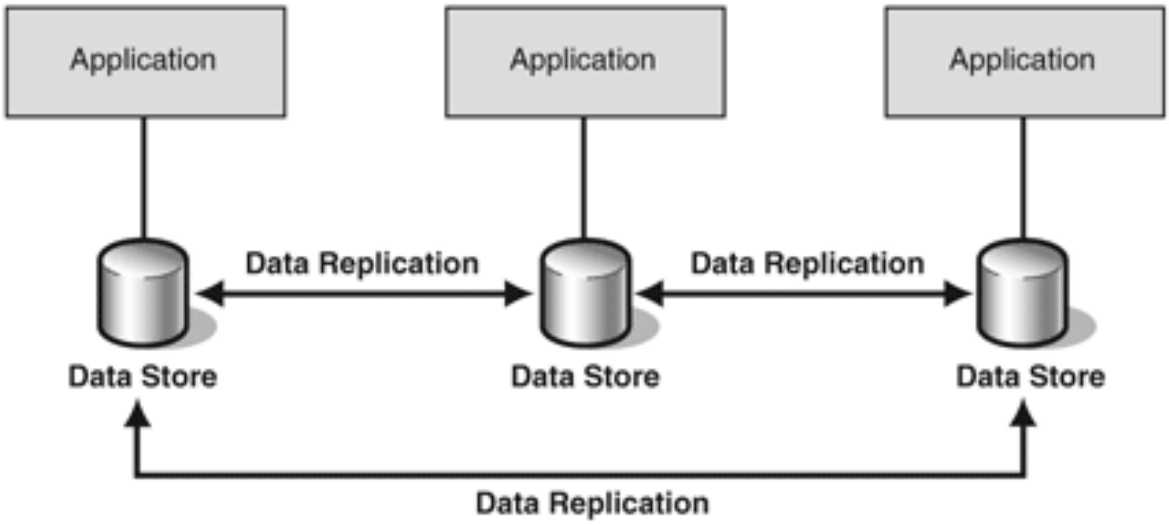



Figure 3-4: Multiple Copies Approach [20]

An approach suggested in [20] creates multiple copies of the database and tenant has its own dedicated store. Data is copied from one store to another to keep copies synchronized. In this

approach, every organization has a dedicated storage space but there is more than one copy of data and overhead of handling data synchronization issues is also involved.

### 3.2.3 Two Way Synchronization Approach

In [21] data sharing is done through two way synchronization between tenants using web services. In this approach, there is copy of data on each tenant and is synchronized between the tenants whenever the data is changed. Data duplication and overhead of data synchronization issues are inherent in this approach.

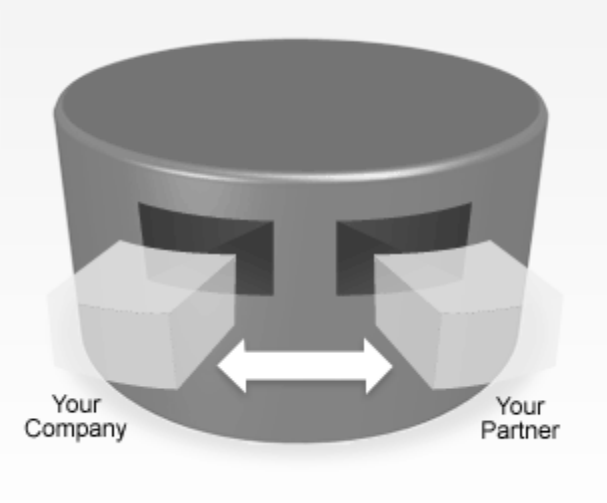



Figure 3-5: Two Way Synchronization Approach [21]

### 3.2.4 Summary

All the data sharing approaches discussed above are either maintaining multiple copies of the data or not suitable for enterprise applications.

## 3.3 Data Integrity and Accuracy

These day's companies are more focused on their core competencies, and because of reduced costs of telecommunication and Infrastructure they are keen to outsource their IT units. The trend of outsourcing is increasing by the rate of 79% these days. But whenever there is talk about outsourcing of data processing, a big question mark arrives on data security. Every outsourcing party requires it data processing to be secured on infrastructure level in order to gain confidence. For most of the cases, security of data is considered when clients perform their queries in

encrypted manner and also when data is stored in encrypted form. But in all these scenarios the Integrity of data is very important, and which cannot be ensured without introducing some checking mechanisms, like changes in DBMS kernel or by setting some subset of data on client side. The given two mechanisms can ensure integrity of data but their implementation is considered difficult, because it is hard to implement kernel again, and when you are using mobile clients then it is very difficult to store subset of data.

Author of [22] has presented the randomize approach for ensuring the integrity of the data. This will be ensured by generating fake tuples randomly and on the basis of these tuples; client queries the server and ensures the integrity of the data which is received. Over here client which needs data already have some prior information about certain tuples that must be present in the incoming data and if that information is present, the data is considered correct if it is not present then data is considered incorrect. Therefore client in this case will be storing all the fake tuples which might not be very good.

In order to overcome the drawback of randomize approach the author has presented the deterministic approach of data validation. In this approach instead of generating tuples randomly, they have used deterministic function to generate those tuples and have divided the result space into a discrete grid and used this grid to validate the results returned by the server. This concept is very similar to randomize approach explained above; the only difference is that instead of storing all fake tuples the client has used gird to validate the results by finding out how many are fully covered and how many are partially covered.

But when we talk about the discrete grid, it might be impractical when we have high dimensional feature space, because of two reasons. First is there might be cell in gird which are empty and not queried second is the impracticality in those cases where clients do not support highly dimensional gird. To handle this situation the author has shown histogram based approach to generate tuples. In this approach we distribute our queries based on the region defined on the grid, in this way we can overcome the problem explained previously.

### 3.3.1 **Summary**

These approaches will be used to check accuracy and integrity of transformed data that is going to be shared with other tenant.

# 4. Research Methodology

In this chapter a formal problem statement is presented along with research approach adopted for this thesis.

## 4.1 Problem Statement

How can data be shared between tenants of a multi-tenant SaaS application in a heterogeneous and distributed environment to reduce human effort, eliminate data duplication, and assuring integrity/accuracy of the shared data?

Research Approach

### 4.1.1 Scope

All data-centric multi-tenant application/models come under the scope of this research.

### 4.1.2 Methodology

Table 4-1 outline the questions inherent in the problem statement and method adopted to answer those questions.

| **Question** | **Method** |
|---|---|
| How data migration problems can be solved in the context of multi-tenant application? | Proof of Concept |
| How data sharing can be incorporated in data-centric multi-tenant architectures? | Proof of Concept |
| What will be the constraints for data sharing in data-centric multi-tenant architectures? | Proof of Concept |
| What will be the performance issues due to data sharing in data-centric multi-tenant architectures? | Proof of Concept |
| What are the benefits of data sharing over data migration? | Proof of Concept |
| How data can be shared without losing its integrity and accuracy? | Proof of Concept |

Table 4-1: Research Methodology

# 5. Middleware for Data Sharing

The proposed solution to share data between tenants includes: 1) extension to current data-centric multi-tenant models for SaaS applications; 2) middleware for data sharing between different organizations; 2) a design pattern and algorithm for the implementation of the middleware.

## 5.1 Multi-tenant Databases Extension

There are several different implementations of a database that support multi-tenancy as discussed in [16, 17, 18, 19]. Databases in all the architectures are logically partitioned using tenant or organization identifier. To support data sharing at application level we have introduced a new table (Tenant_Heirarchy) to store any relationship between tenants. This table is a child of the table that stores information about the tenants (usually Tenant, Company, or Organization table). The structure of the new table is shown in Table 5-1. Primary tenant is the tenant of currently logged in user and secondary tenant is the one whose data we are going to share with the primary tenant. Require mappings flag (requires_mapping) indicates that the secondary tenant data is on a different type of DBMS or structure of the tables in the database of secondary tenant is different.

| **Tenant_Heirarchy** | | | |
|---|---|---|---|
| ***tenant_heirarchy_id*** | ***primary_tenant*** | ***secondary_tenant*** | ***requires_mapping*** |
| 1 | 1 | 4 | TRUE |
| 2 | 1 | 6 | FALSE |

Table 5-1: Structure of Tenant_Hierarchy Table

## 5.2 Data Sharing Middleware

Our solution proposes a middleware that will work above data access layer and below access control and business logic layer of the application to address the challenges explained in section 2.3. It will only be used if there are any secondary tenants attached with the primary tenant. Architecture of the middleware is generic and can be implemented using any development platform. Figure 5-1 shows simplified flow of the application after introducing the data sharing middleware. Figure 5-2 is showing detailed architecture of data-centric multi-tenant application.

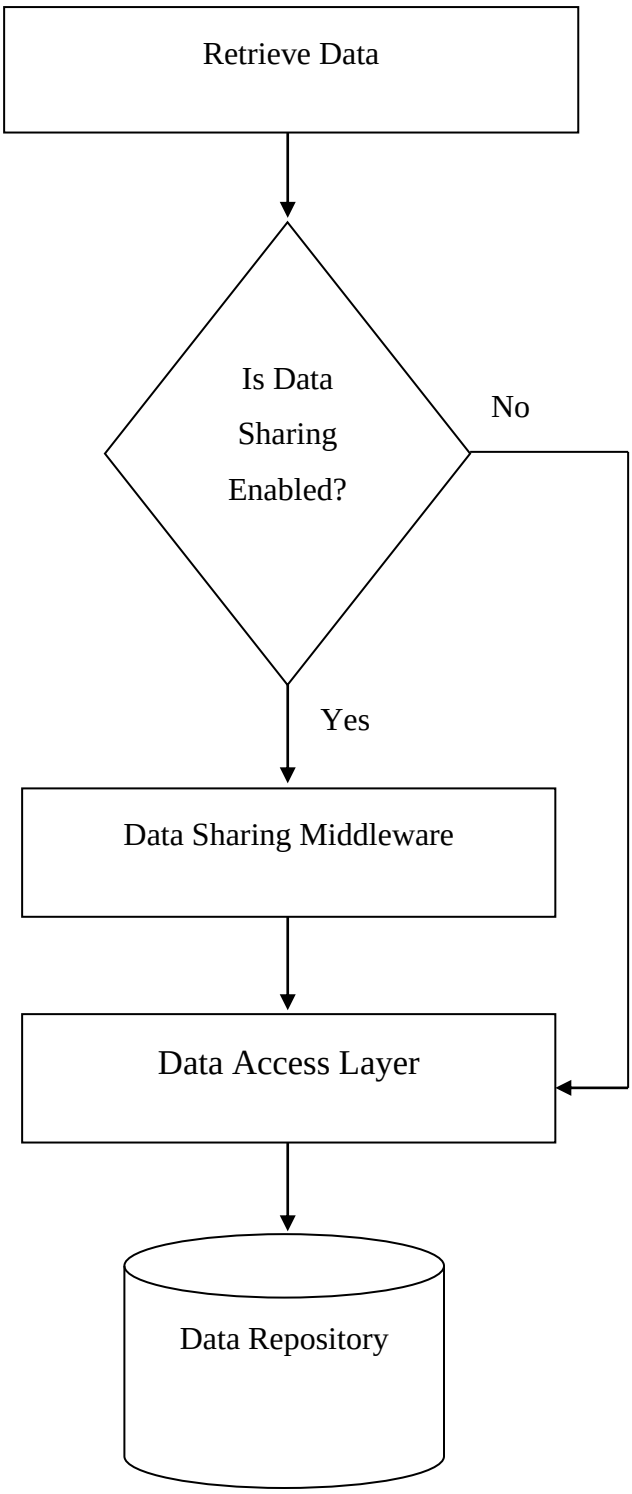



Figure 5-1: Simplified Flow of SaaS after introducing Data Sharing Middleware

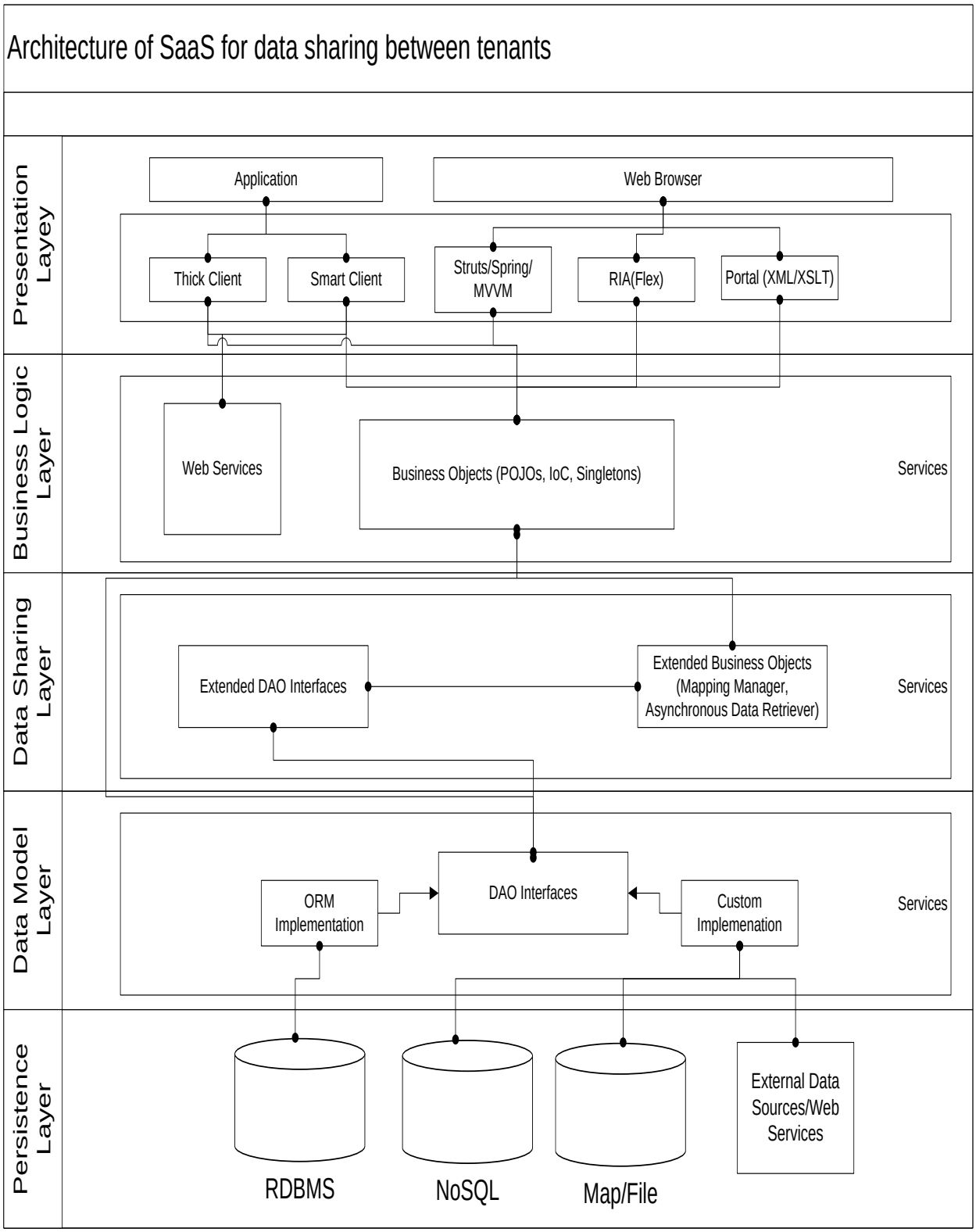



Figure 5-2: Detailed Architecture of Data-Centric Multi-Tenant Application

## 5.3 Data Sharing Middleware

Mapping Manager, Asynchronous Data Retrieval, and Extended DAO Interfaces are three modules of our proposed middleware for data sharing. Higher level architecture diagram of the middleware is shown in the Figure 5-3.

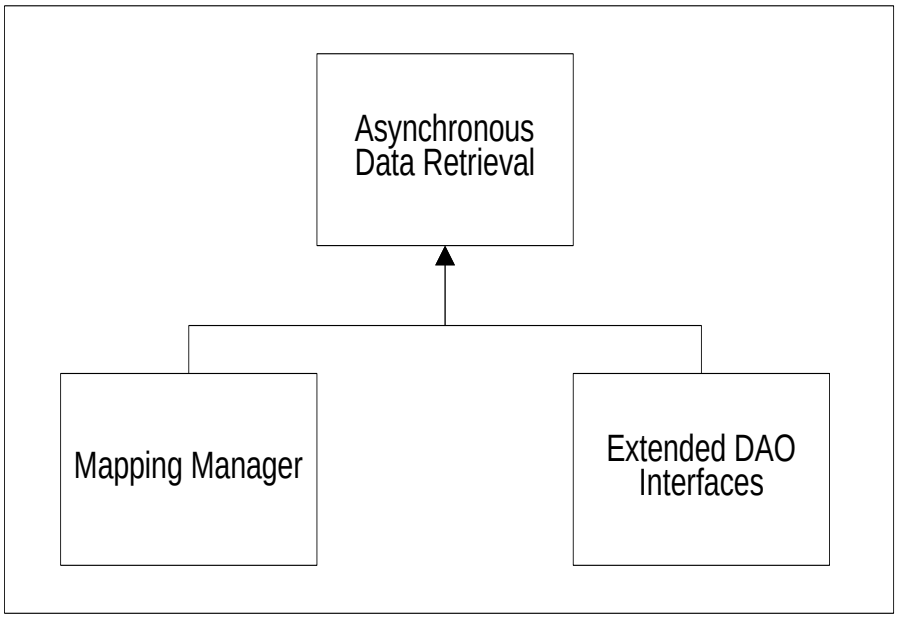



Figure 5-3: Higher Level Architecture Diagram of the Middleware

Asynchronous Data Retrieval modules get mapping details from Mapping Manager and transform data in Extended DAO Interfaces module using mapping details and sends data back to the requester.

### 5.3.1 Mapping Manager

This module is responsible for creating and manipulating mappings between the tables of primary and secondary tenant. This module is responsible for handling A and B challenge described above. Mappings can be either in the form of XML or stored in a table in database. We recommend storing mappings in database table because by using this way we don't have to write a custom XML parser to create, edit, and read mappings. The mapping will have the information about how data from a table in a database of a secondary tenant will relate to table in a database of the primary tenant and how data will be transformed from one structure to another.

| **Mapping** | | | |
|---|---|---|---|
| ***mapping_Id*** | ***tenant_heirarchy_id*** | ***primary_table*** | ***secondary_table*** |
| 1 | 1 | Account | Accounts |
| 2 | 1 | Quote | Order |

Table 5-1: Structure of Mapping Table

| **Mapping_Detail** | | | | | | | |
|---|---|---|---|---|---|---|---|
| ***det_id*** | ***mapp_id*** | ***pri_col*** | ***pri_typ*** | ***pri_fmt*** | ***sec_col*** | ***sec_type*** | ***sec_fmt*** |
| 1 | 1 | name | varchar | | title | varchar | |
| 2 | 1 | c_date | date | dd-mm-yy | c_on | datetime | |

Table 5-2: Structure of Mapping_Detail Table

Structure of Mapping and Mapping_Detail tables are shown in Table 5-1 and 5-1 required to store mappings. Mapping table is child of Tenant_Heirarchy table while Mapping_Detail is child of Mapping table.

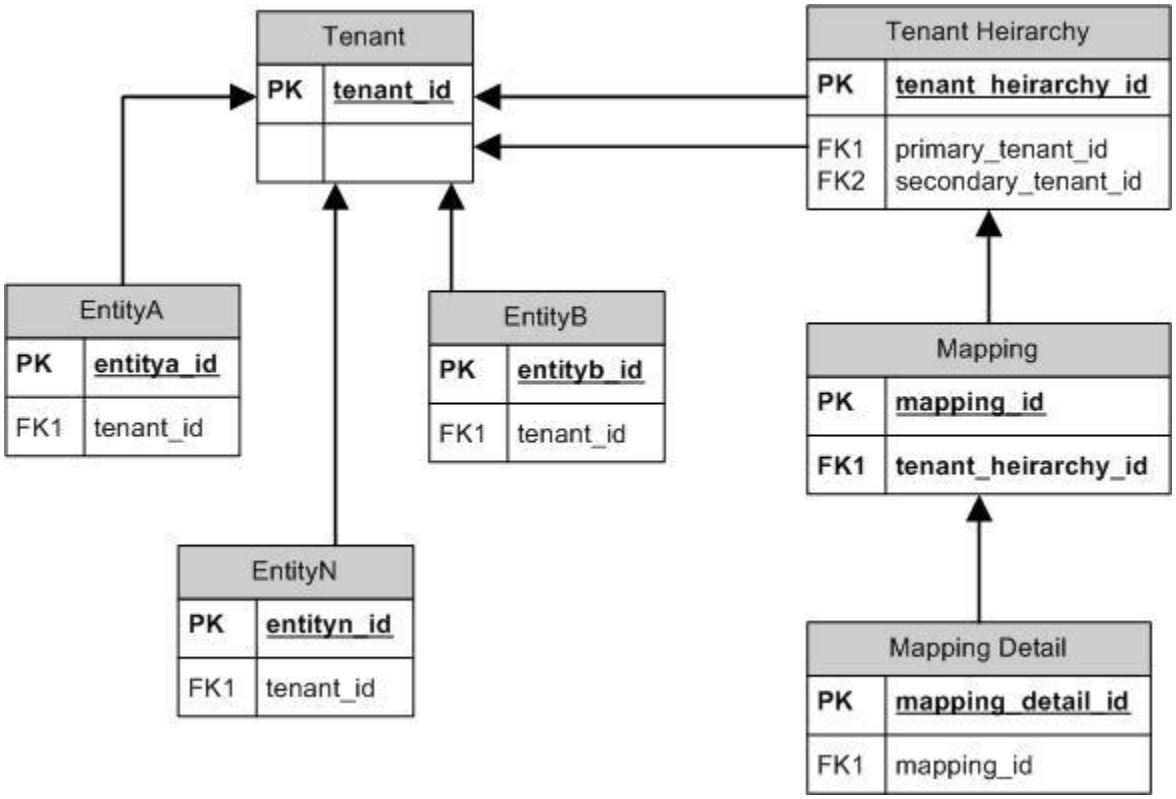



Figure 5-4: Data-Centric Multi-Tenant Architecture with Data Sharing

After introducing the three tables in data-centric multi-tenant architecture, the ERD of the architecture will be something like Figure 5-4.

### 5.3.2 **Asynchronous Data Retrieval**

This module is recommended if the application's programming language support multi-threading or multi-core programming. This module will retrieve, and transform data from different sources asynchronously and in parallel fashion.

To retrieve asynchronously we have suggested a design pattern as shown in Figure 5-5.

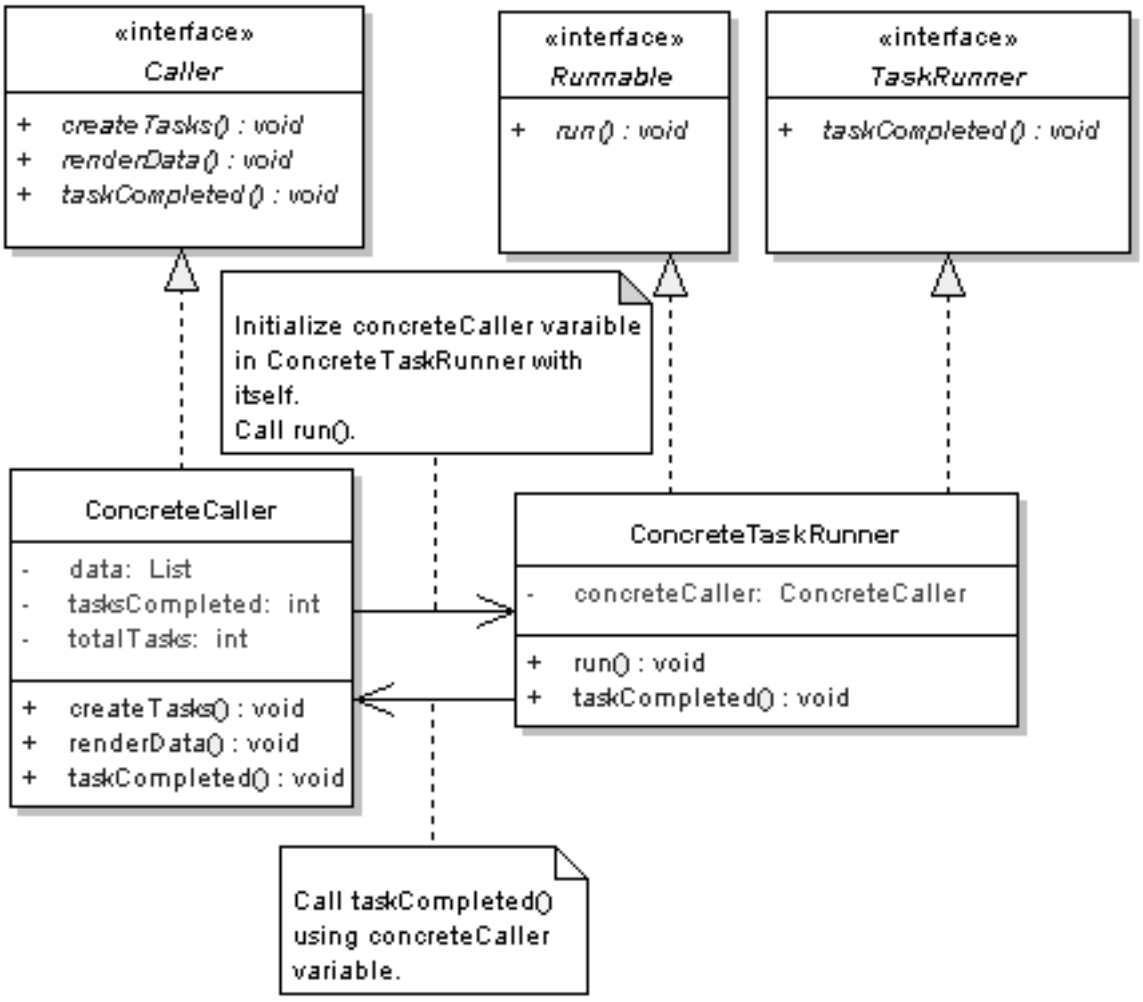



Figure 5-5: Asynchronous Data Retrieval Design Pattern Class Diagram

This design pattern is not language specific and can be implemented if the application's platform supports multi-thread programming. This module will help us to retrieve data from different

servers efficiently without affecting the overall performance of the application. Sequence diagram of this design pattern is in shown in Figure 5-6.

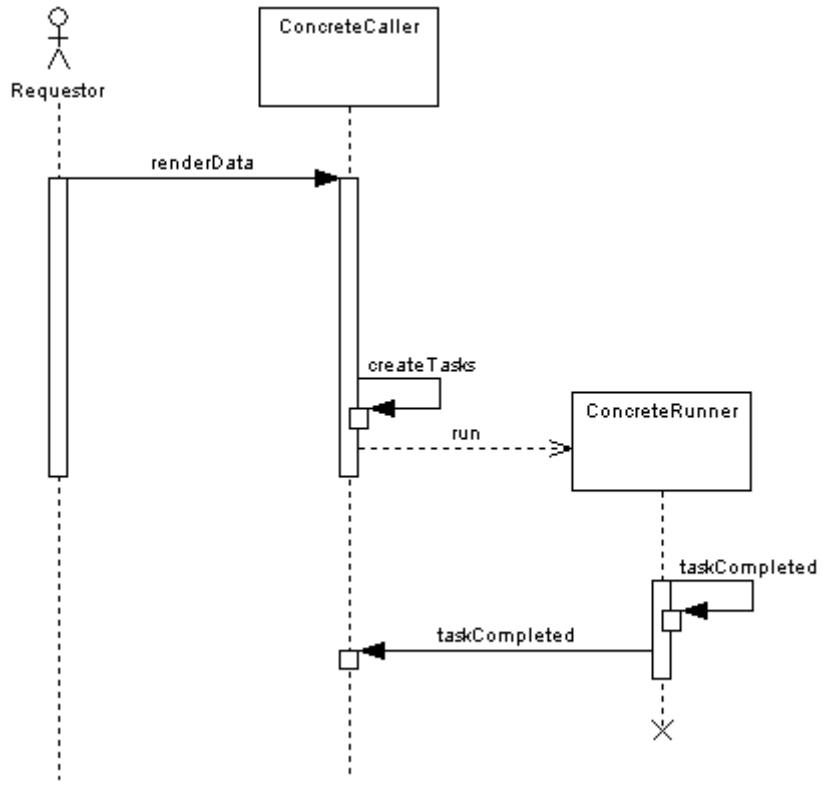



Figure 5-6: Asynchronous Data Retrieval Design Pattern Sequence Diagram

### 5.3.3 **Extended DAO Interfaces**

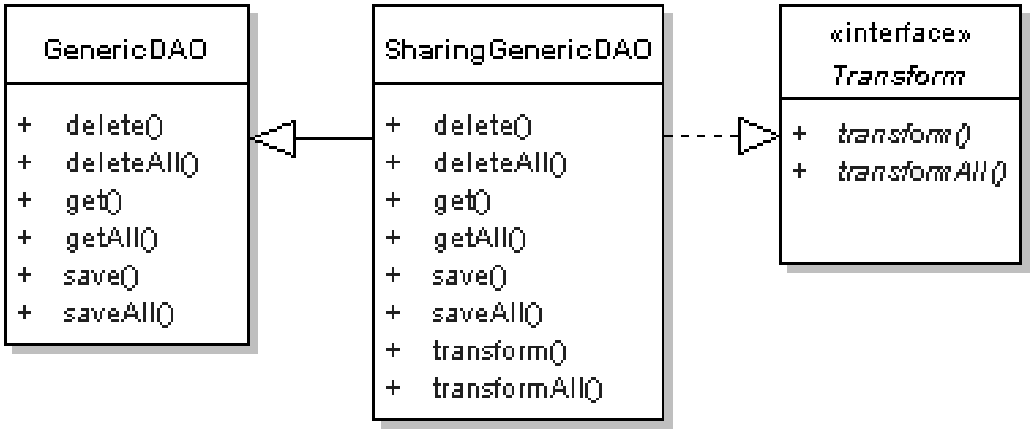



Figure 5-7: Extended DAO Interfaces Class Diagram

If data is to be retrieved from different types of sources and each source has different structure, we have to transform the data of each secondary tenant. DAOs are responsible for persistence and retrieval of data to and from DBMS. To transform data, we suggest extending the data access object (DAO) of data access layer to complete the job of data transformation from one form to another. Figure 5-7 shows the class diagram for Extended DAO Interfaces. The Extended DAO Interfaces will utilize most of the functionality of existing DAOs and will only be responsible for data transformation.

## 5.4 Algorithm

Steps that need to be performed to share data between tenants are outlined in the Table 5-3.

| | **Algorithm: Data Sharing in Data-Centric Multi-Tenant SaaS** |
|---|---|
| | ***Input:*** Primary Tenant ID<br>***Output:*** Data from primary and all the secondary tenants attached with primary tenant |
| ***Step 1*** | *1.1* Check if any secondary tenants attached with the primary tenant.<br>*1.1.1* If there are no secondary tenants, go to step *4.2*<br>*1.2* Get all secondary tenants IDs attached with the primary tenant |
| ***Step 2*** | *2.1* Check if the data of primary and secondary tenant will come from different DBMS or the structure of secondary tenant's database is different<br>*2.1.1* If yes, get mapping between table of primary tenant and table of secondary tenant.<br>*2.1.1.1* If mappings are not available, go to step 4.2 |
| ***Step 3*** | *3.1* Get data from secondary tenant's DBMS<br>*3.2* If mappings are involved<br>*3.2.1* Transform data from secondary source using mappings |
| ***Step 4*** | *4.1* Repeat step 1.1 to 3.2 until data of all secondary tenants is retrieved<br>*4.2* Get data for primary tenant<br>*4.3* If there is any data from secondary tenants, merge it with primary tenant |

Table 5-3: Detailed Architecture of Data-Centric Multi-Tenant Application

Figure 5-8 graphically shows the activities performed in the algorithm.

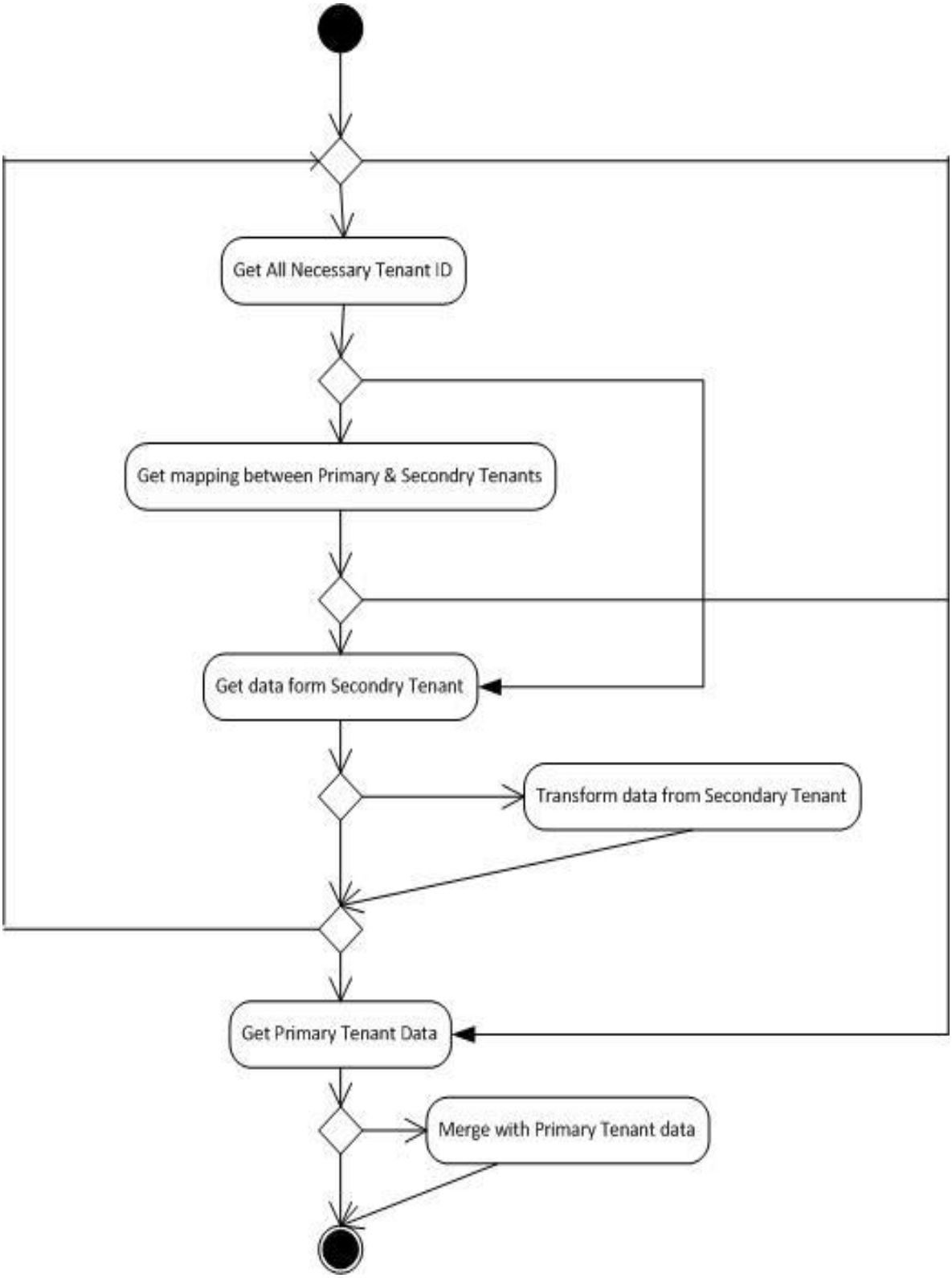



Figure 5-8: Activity Diagram of Algorithm

## 5.5 Constraints

Asynchronous Data Retrieval is recommended if the application's programming language support multi-threading or multi-core programming. This module will retrieve, and transform data from different sources asynchronously and in parallel fashion.

There are four configurability maturity levels of SaaS and are determined by the ability provided by SaaS to configure its data (D), user interface (UI), and logic (L) [2]. Table 5-4 shows that which modules of our proposed middleware are required by SaaS for data sharing if it is at a particular level of configurability maturity.

| **Middleware Modules** | **SaaS Configurability Maturity Level** | |
|---|---|---|
| | **Level 1** | **Level 2, 3, 4** |
| | None | UI & L & D |
| **Mapping Manager** | No | Yes |
| **Asynchronous Data Retrieval** | Yes | Yes |
| **DAL Extension** | No | Yes |

Table 5-4: Middleware Module required for SaaS Configurability Maturity Level

Few data sharing challenges in data-centric multi-tenant SaaS are described in section 2.3. If a SaaS has any or all of the challenges identified by us, it can be met by one of the module of our proposed middleware. Modules required for a challenge are shown in Table 5-5. After data transformation, there is a strong chance that data lose its accuracy and integrity. Data Integrity and Accuracy issues only arise if is to be shared from a remote location having different architecture from the host location.

| **Middleware Modules** | **Challenges** | | |
|---|---|---|---|
| | **Distributed Environment** | **Heterogeneous Environment** | **Heterogeneous Structure** |
| **Mapping Manager** | No | Yes | Yes |
| **Asynchronous Data Retrieval** | Yes | Yes | Yes |
| **DAL Extension** | No | Yes | Yes |

Table 5-5: Middleware Module required for Data Sharing Challenge

## 5.6 Advantages

The result of this research will eliminate the human effort involved in data migration that took place between the different tenants of a SaaS, avoids data duplication, and efficient use of data storage facility. Eventually saves time and financial resources of both SaaS providers and users.

## 5.7 Areas of Application

This research is directly applicable to all data-centric multi-tenant applications, where the application has virtually partitioned data and configuration for each tenant. The result of this research can be used by any enterprise providing SaaS/cloud-based data management solutions.

# 6. Results

In this chapter results related to data retrieval efficiency, integrity, and accuracy are discussed.

## 6.1 Efficiency Results

In this section results related to efficient retrieval of data are discussed.

### 6.1.1 Evaluation Environment

The environment to evaluate middleware includes two VM instances with MySQL 5.5 and MongoDB 2.2 DBMS in each instance. We have installed DBMSs in VMs to emulate distributed environment. MySQL is a structured DBMS while MongoDB is unstructured or NoSQL DBMS.

| **MongoDB** | | | **MySQL** | |
|---|---|---|---|---|
| **Accounts** | | | **Account** | |
| ***Column*** | ***Type*** | | ***Column*** | ***Type*** |
| Name | string | → | name | varchar |
| acc_code | string | → | acc_id | bigint |
| pri_contact | string | → | contact_id | bigint |
| bill_phone | string | → | phone | varchar |
| phone | string | | | |
| bill_fax | string | → | fax | varchar |
| Fax | string | | | |
| web_site | string | → | website | varchar |
| email | string | → | pri_email | varchar |
| createddate | string | → | create_date | datetime |
| updateddate | string | → | updated_date | datetime |
| org_id | string | → | t_id | bigint |
| | | | updated_by | bigint |
| | | | created_by | bigint |

Table-6-1: Structure of tables used and mapping of columns

Database in MySQL have a table named Account with three million records, one million records for each tenant. Similarly database in MongoDB have Accounts table with four million records, one million records for each tenant. Table 6-1 shows structure of table in each database and mapping between the tables. The arrowhead (→) between tables shows mapping of each column. Both tables have different names, column names, and DBMS to emulate the challenge 2.3.1 and

2.3.2 explained in section 2.3. Code is written using JDK 1.7 and executed on 2.2 Core i5 64-Bit machine with 4GB RAM to share data between one tenant having data in MySQL running in a VM instance with a tenant with data in MongoDB running in other VM instance.

### 6.1.2 **Results**

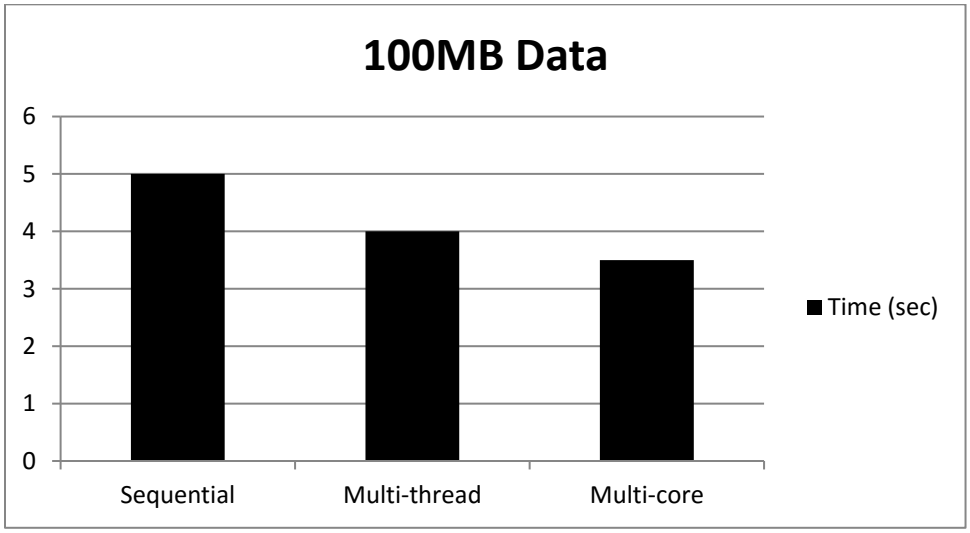
Average time to retrieve and transform one million records from a DBMS records in this evaluation environment is 3.45 seconds approximately.

The data is retrieved and transformed by using three different modes; Sequential, Multi-thread, and Multi-core.

**Sequential:** In sequential mode, data is retrieved from one DBMS and then from second DBMS in a single invocation.

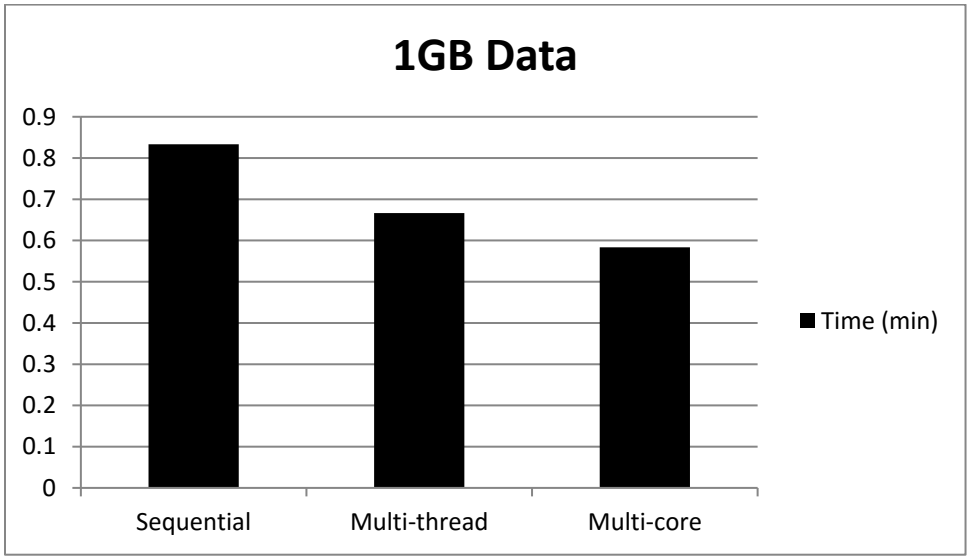
**Multi-thread:** In this technique two threads are created programmatically to retrieve data from each DBMS asynchronously.

**Multi-core:** While in multi-core mode, task of asynchronous data retrieval from each node is assigned to different cores.

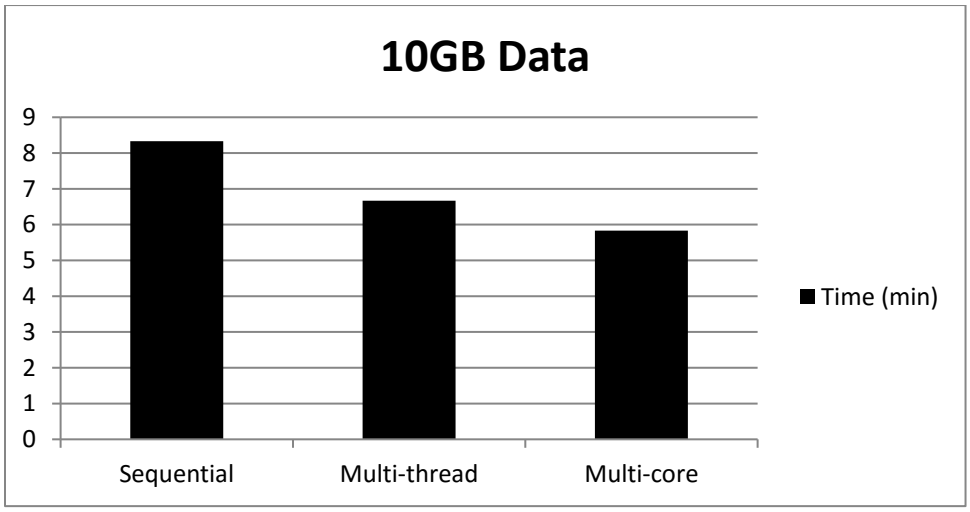



Graph 6-1: Average Time for Retrieval, Transformation, and Merge - 100MB

In all these techniques all the VMs and host machine is restarted before evaluating each mode so the data in RAM or cache should not affect our results. Graph 6-1, 6-2, and 6-3 are showing the average time for retrieving, transforming, and merging different size of data.




Graph 6-2: Average Time for Retrieval, Transformation, and Merge - 1GB




Graph 6-3: Average Time for Retrieval, Transformation, and Merge - 10GB

This average time is calculated after executing each technique three times and includes time required for retrieval and transformation of data from secondary tenant. Our experiment shows that data can be shared in shared, distributed, and heterogeneous environment.

If the underlying platform supports multi-thread or multi-core programming, the retrieval and transformation average time for two million records from two different DBMS is quite close to average time to retrieve and transform one million records from a DBMS. This shows that data sharing will not affect the overall performance of SaaS.

## 6.2 Accuracy and Integrity Results

### 6.2.1 Evaluation Environment

Evaluation environment for accuracy and integrity results is similar to the environment explained above in section 6.1.1 except the number of records is ten thousand per tenant while fake records are one thousand to test the accuracy and integrity. Fake records or values of deterministic function are stored in a database at host machine. Mappings in Table 6-1 above are used to transform data from the remote DBMS.

### 6.2.2 Results

It is inefficient to check all the records returned by the query/retrieval call for accuracy and integrity. That's why following three techniques are used for this purpose presented in [22].

**Randomize Approach:** In this approach accuracy will be ensured by generating fake tuples randomly and on the basis of these tuples; client queries the server and ensures the integrity of the data which is received. Over here client which needs data already have some prior information about certain tuples that must be present in the incoming data and if that information is present, the data is considered correct if it is not present then data is considered incorrect.

**Deterministic Approach:** In this approach instead of generating tuples randomly, a deterministic function is used to generate a value for the tuples and have divided the result space into a discrete grid and used this grid to validate the results returned by the server. This concept is very similar to randomize approach explained above; the only difference is that instead of storing all fake tuples, use gird to validate the results.

**Distributed Approach:** In this approach fake tuples are equally distributed across all the data and then finding fake tuples in results returned by the query/retrieval call. Figure 6-2 shows the fake tuples used in our test.

| account_id | name | account_code | primary_contact | phone | fax | website_identifier | support_email |
|---|---|---|---|---|---|---|---|
| 1 | Test Name 10 | C10 | Contact 1 | 908510 | 908710 | http:www.10.com | ac10.com |
| 2 | Test Name 20 | C20 | Contact 1 | 908520 | 908720 | http:www.20.com | ac20.com |
| 3 | Test Name 30 | C30 | Contact 1 | 908530 | 908730 | http:www.30.com | ac30.com |
| 4 | Test Name 40 | C40 | Contact 1 | 908540 | 908740 | http:www.40.com | ac40.com |
| 5 | Test Name 50 | C50 | Contact 1 | 908550 | 908750 | http:www.50.com | ac50.com |
| 6 | Test Name 60 | C60 | Contact 1 | 908560 | 908760 | http:www.60.com | ac60.com |
| 7 | Test Name 70 | C70 | Contact 1 | 908570 | 908770 | http:www.70.com | ac70.com |
| 8 | Test Name 80 | C80 | Contact 1 | 908580 | 908780 | http:www.80.com | ac80.com |
| 9 | Test Name 90 | C90 | Contact 1 | 908590 | 908790 | http:www.90.com | ac90.com |
| 10 | Test Name 100 | C100 | Contact 1 | 9085100 | 9087100 | http:www.100.com | ac100.com |

select * from account_distributed LIMIT 0, 1000

Figure 6-1: Fake tuples used in Distributed Approach

Valid mappings results are shown in Figure 6-2 while Figure 6-3 is showing invalid mappings result.

```
Console
<terminated> ProofOfConcept [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Dec 31, 2012 12:06:32 PM)
Database connection established
SELECT * FROM  tenant_heirarchy where primary_tenant_id = 4 and secondary_tenant_id = 1
Database connection established
Distributed Test
totoal retrieved records: 10000
retrieved records should have 1000 fake records
total fake records found: 1000
```

Figure 6-2: Valid Mappings Result

```
Console
<terminated> ProofOfConcept [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Dec 31, 2012 12:13:21 PM)
null
null
Database connection established
Distributed Test
totoal retrieved records: 10000
retrieved records should have 1000 fake records
total fake records found: 0
```

Figure 6-2: Invalid Mappings Result

After applying these techniques the results show that accuracy and integrity is directly proportional to mapping. If mappings are correct, data can be transformed and shared accurately without losing its integrity.

# 7. Conclusion

In data-centric multi-tenant SaaS applications there are situations like mergers and joint marketing campaigns; data needs to be shared between tenants. Currently the data is shared by migrating data from tenant to another. Data migration requires a very skilled and time consuming human effort and it results in data duplication. A feasible, comprehensive, and implementable solution for data sharing in data-centric multi-tenant SaaS is not proposed by anyone yet. Hence the idea proposed in this thesis is of great importance and will take SaaS economy of scale to another level. A middleware for existing applications to share data between different tenants of the SaaS in shared, heterogeneous, and distributed environment without changing existing DAL is presented in this research. The suggested middleware is based on three modules which are loosely coupled with each other and can be added or removed depending on the complexity of the SaaS. Feasibility of the middleware is explained using an example.

If we have data access layer for another SaaS, this middleware can be extended to share data between different SaaS applications.

## 7.1 Future Work

The suggested solution shares all the data of one organization with another organization. Organizations usually create security policies to restrict data access for individual users. To further enhance this solution, security policies of primary and secondary tenants related to data being retrieved and shared should be considered. In other words, restricted data sharing between organizations can be an extension of the suggested solution and basis for future work.

## 8. References


[1] C. Bezemer and A. Zaidman, "Multi-tenant SaaS applications: maintenance dream or nightmare?," Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), New York, 2010, pp. 88-89.

[2] Z. Zhu, L. Chen, J. Song and G. Liu, "Applying SaaS Architecture to Large Enterprises for Enterprise Application Integration," 2nd International Conference on Information Science and Engineering (ICISE), 2010, pp. 1888-1889.

[3] "Cloud Computing," Internet: http://en.wikipedia.org/wiki/Cloud_computing, [Mar. 03, 2012].

[4] "On-Demand Release Management," Online: http://salesforce.vo.llnwd.net/o1/us/community/ppt/ADM023_Farwell.ppt, Jul. 23, 2009 [Mar. 03, 2012], pp. 15-18.

[5] S. Aulbach, M. Seibold, D. Jacobs and A. Kemper, "Extensibility and Data Sharing in Evolving Multi-Tenant Databases," IEEE 27th International Conference on Data Engineering (ICDE), Germany, 2011, pp. 99-101.

[6] I. Foster, Y. Zhao, I. Raicu and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," Grid Computing Environments Workshop, Chicago, USA, 2008, pp. 1-2.

[7] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," International Journal of Software: Practice and Experience (SPE), Wiley Press, New York, USA, May 2002, pp. 135-164.

[8] "Supercomputer," Internet: http://en.wikipedia.org/wiki/Supercomputer, [Mar. 03, 2012].

[9] Armbrust, M., Fox, A., Griffith, R. et al, "A View of Cloud Computing," Communications of the ACM, Volume 53 Issue 4, 2010, pp 50-58.

[10] C. Bezemer, A. Zaidman, "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare," Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), New York, USA, 2010, pp 88-92.

[11] "Virtualization," Internet: http://en.wikipedia.org/wiki/Virtualization, [Mar. 03, 2012].

[12] N. Kiyanclar, "A Survey of Virtualization Techniques Focusing on Secure On-Demand Cluster Computing," University of Illinois at Urbana-Champaign, 2005.

[13] "The role of a Load Balancer in a Platform-as-a-Service," Internet: http://blog.afkham.org/2011/09/role-of-load-balancer-in-platform-as.html, [Mar. 03, 2012].

[14] M. Alakeel, "A Guide to Dynamic Load Balancing in Distributed Computer Systems," IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.6, 2010.

[15] “What are services?,” Internet: http://info.apps.gov/content/what-are-services, [Mar. 03, 2012].

[16] C. D. Weissman and S. Bobrowski, “The Design of the Force.com Multitenant Internet Application Development Platform,” ACM SIGMOD International Conference on Management of Data, New York, 2009, pp. 889-892.

[17] C. Weiliang, Z. Shidongt and K. Lanju, “A Multiple Sparse Tables Approach for Multi-tenant Data Storage in SaaS,” 2nd International Conference on Industrial and Information Systems (IIS), 2010, pp. 414-415.

[18] S. Aulbach, D. Jacobs, A. Kemper and M. Seibold, “A Comparison of Flexible Schemas for Software as a Service,” ACM SIGMOD International Conference on Management of Data, 2009, pp. 883-884.

[19] S. Aulbach, T. Grust, D. Jacobs, A. Kemper and J. Rittinger, “Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques,” ACM SIGMOD International Conference on Management of Data, New York, 2008, pp. 1196-1199.

[20] “Microsoft Data Integration Patterns,” Internet: http://msdn.microsoft.com/en-us/library/ff647273.aspx, [Mar. 03, 2012].

[21] “Salesforce to Salesforce,” Internet: http://www.salesforce.com/platform/cloud-infrastructure/salesforce-to-salesforce.jsp, [Mar. 03, 2012].

[22] M. Xie, H. Wang, J. Yin and X, Meng, "Integrity auditing of outsourced data," Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007.