Islamic University of Gaza
Deanery of Higher Studies
Faculty of Information Technology

An Efficient Approach for Supporting Multi-Tenancy Schema Inheritance in RDBMS for SaaS

***By*:**

Samir A. Hillis

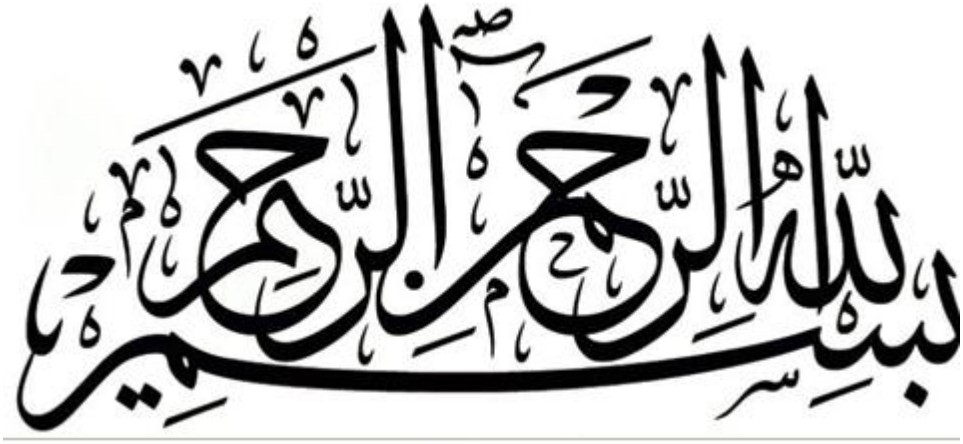
120100708

Supervisor:

Dr. Tawfiq SM Barhoom

A Thesis Submitted as Partial Fulfillment of the Requirements for the Degree of Master in Information Technology

2015, 1436H



Abstract

Multi-tenant data management is a major application of Software as a Service (SaaS) , whereby a third party service provider hosts databases as a service and provides its customers with needed services. SaaS applications are deployed on a shared environment that can be accessed by the users from client-end software by using the Internet. 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 user interface (UI) for all the organizations and data of multiple tenants are saved in a single database to reduce total cost of ownership. Common practice is to map multiple single-tenant logical schemas in the application to one Multi-tenant physical schema in the database. Such mappings are challenging to create .This is due to the flexibility of a base scheme to be extended by enterprise application tenants which provides different dynamically modified versions of the application. The fundamental limitation on scalability of this approach is the number of tables of database can handle. Shared Tables Shared Instances (STSI) is a state-of-the-art approach to design the schema. However, they suffer from performance time and high space overhead.

In this research, we are going to introduce an efficient approach for supporting Multi-tenancy schema inheritance based on STSI, that allows sharing core application schema between tenants while enabling schema extensions per tenant, Schema inheritance allows deriving a schema from another schema. Thereby, a derived schema inherits the objects that are defined in the parent schema. The idea is based on the changes that occur at runtime in the meta data and the data . Also exploitation some situations of data needs to be shared between tenants.

Several experiments were conducted to trade-off STSI and our approach. Different sizes of small, medium, large and very large databases, starting from 10 GB up to 300 GB. Experimental results show that our method achieves good scalability and high performance with low space requirement, and outperforms STSI methods at different rates depending on data manipulation language (DML) operations. It is ranged 50% in selection processes, and have been in the range of 20% and 40% in the update and insert. Also, our approach achieved less storage space compared with STSI about 50% .

Keywords: *cloud computing, Database as a Service (DBaaS),Multi-tenant database , schema-mapping technique, Benchmarking .*

عنوان البحث باللغة العربية

منهج كفؤ لدعم وراثة مخطط متعدد المستأجرين لأنظمة إدارة قواعد البيانات العلائقية في البرمجيات كخدمة

الملخص باللغة العربية

إدارة البيانات متعددة المستأجرين هي تطبيق هام ضمن تطبيقات البرمجيات كخدمة (SaaS)، حيث يستضيف مزود خارجي "طرف ثالث" قاعدة البيانات ويوفرها كخدمة للعملاء مع باقي الخدمات اللازمة . يتم نشر تطبيقات البرمجيات كخدمة على بيئة مشتركة مما يتيح لمستخدمين متعددين الوصول إليها عبر الإنترنت .
تشير معمارية تعدد المستأجرين إلى وجود نسخة واحدة من البرنامج تعمل على الخادم ، لكنها تقدم الخدمة لمنظمات متعددة يسمى كل منها مستأجر . تعمل تطبيقات تعدد المستأجرين على توفير واجهة بيانية لجميع المستأجرين . ويتم حفظ بيانات أولئك المستأجرين على خادم قاعدة بيانات واحد بدلا من امتلاك كل مستأجر لها بهدف الحد من التكاليف الإجمالية. أحد الأساليب الشائعة هي مطابقة العديد من قواعد البيانات لمستأجرين متعددين مع مخطط منطقي واحد في قاعدة البيانات . هذه المطابقة تشكل تحديا عند إنشائها . ويتحقق هذا بمرونة مخطط قاعدة البيانات الذي يمكن تمديده من قبل مستأجرين وتعديله وقت التشغيل. أحد القيود على هذا النهج هو عدد جداول قاعدة البيانات . يعتبر منهج الجدول المشترك – النسخة المشتركة من أهم منهجيات تصميم المخطط . ومع ذلك فإنها تعاني من انخفاض الأداء وارتفاع حاجتها للتخزين .

في بحثنا نقدم منهج فعال لدعم تعدد الإيجار يعتمد الوراثة ، إنه يسمح بمشاركة المخطط الأساسي للتطبيق بين مختلف المستأجرين مع السماح لكل مستأجر تمديد المخطط . علاوة على أنه يسمح باشتقاق مخطط من مخطط آخر، وبالتالي فهو يرث كافة الكائنات سابقة التعريف بدلا من إعادة تعريفها وتستند فكرته على الاستفادة من التغييرات التي تحدث في وقت التشغيل على البيانات الوصفية والبيانات نفسها ، كما أنه يستفيد من بعض الحالات التي تكون فيها البيانات مشتركة بين المستأجرين .
قمنا بالعديد من التجارب للمفاضلة بين STSI ونهجنا على أحجام بيانات مختلفة (صغيرة ومتوسطة وكبيرة وكبيرة جدا بدءا 10 GB إلى 300 GB ، وقد أظهرت النتائج أن منهجنا يحقق تدرجية جيدة وأداء عالي ضمن مساحة تخزين منخفضة مما يجعله يتفوق على STSI بمعدلات مختلفة تختلف حسب طبيعة عملية لغة معالجة البيانات DML . حيث تراوحت في حدود 50% في عمليات الاختيار ، بينما كانت في حدود 20% إلى 40% في عمليات التحديث والإدراج . أيضا حقق نهجنا انخفاض في مساحة التخزين المطلوبة مقارنة مع الجدول المشترك بحدود 50% .

إهداء

إلى من كلله ربي بالهيبة والوقار ... إلى من علمني العطاء دون انتظار

إلى من أحمل اسمه بكل افتخار ...

والدي العزيز ... رحمه الله وجعل قبره روضة من رياض الجنة

إلى من كان دعاءها سر نجاحي وحنانها بلسم جراحي

إلى أمي أطال الله في عمرها وألبسها ثوب العافية

إلى رفيقة دربي زوجتي وإلى أولادي الذين تحملوا معي مشقة الدراسة

إلى أهلي وأساتذتي وأصدقائي وكل ما ساندني

Acknowledgment

First and foremost, I thank **Allah Almighty** for helping me to complete this research successfully, and I am asking Allah to guide me to what he loves and pleases.

I would like to express my deep and sincere gratitude to my supervisor, **Dr. Tawfiq S.M. Barhoom**, Dean of Faculty of Information Technology, in Islamic University-Gaza, for his guidance and for giving me the opportunity to accomplish this research. He advised me during my work to present the research as clearly as possible.
I would like to extend my thanks to many people, for their advices, especially my **academic staff** in Islamic university, for teaching me, and helping me with my courses in my master's degree.

Also, I would like to thank my family: mother, brothers, sisters and a special thank for my wife, for her patience during my study.
I cannot forget my colleagues at work, my friends, and all who supported me.

Samir A. Hillis

Table of contents

Chapter 4

Chapter 5

Chapter 6

List of Figures

List of Tables

List of Abbreviations

CSP	Cloud Service Provider.
IaaS	Infrastructure as a Service.
PaaS	Platform as a Service.
SaaS	Software as a Service.
NIST	National Institute of Standards and Technology.
Ms	Milliseconds
IT	Information Technology
DaaS	Data as a service
DBaaS	Database as a service
RDBMS	Relational Database Management System
SNIA	Storage Networking Industry Association
STSI	Shared Tables and Shared Database Instances
CLOB	Character Large Object
BLOB	Binary Large Object
UI	User Interface
OLTP	Online Transaction Processing
OLAP	Online Analytical Processing
SQL	Structured Query Language
TPC	Transaction Processing Performance Council
TPC-VMS	TPC-Virtual Measurement System
SUT	System under test
SF	Scale factor

Chapter 1

Introduction

It is a clear trend that cloud data outsourcing is becoming a pervasive service. Along with the widespread enthusiasm on cloud computing, In addition to cloud infrastructure and platform providers, such as Amazon, Google, IBM, Microsoft and SalseForce, more and more cloud application providers are emerging which are dedicated to offering more accessible and user friendly data storage services to cloud customers. Cloud computing becomes a natural and ideal choice for organizations and customers. It provides IT-related services over the network on-demand anytime .

Usually the objectives and characteristics of a cloud are to be highly available, scalable, flexible, secure, and efficient . The most important characteristic is scalability. This means applications would scale to meet the demands of the workload automatically. It's important to note that the cloud should not just scale up, but also down in times where the demands are lower. Availability is another critical characteristic of a cloud. An application deployed in a cloud is up and running 24/7/365, basically every minute of every day. Reliable of the cloud refer to an Applications cannot fail or lose data when the system down, and users should not notice any degradation in service. a cloud must be flexible to be compatible with the most efficient means to deploy and extension of an application. An important aspect of a cloud, is that it must be serviceable. Serviceable implies that in the event it is necessary to modify any of the underlying cloud architecture, the application is not disrupted during this time period . Now the software industry is adopting the Software-as-a-Service (SaaS) deployment model in many application domains. A special kind of SaaS offering is a Multi-tenant software application [17]. It serves multiple tenants (e.g., companies or non-profit groups) from a single application instance. A special kind of SaaS offering is a Multi-tenant software application [2,6]. it runs from the same code base, and can thus be maintained centrally [6] .

With more usage of Cloud computing, demand for provisioning of database, services has raised. Provisioning of Cloud databases is known as Database-as-a-Service in Cloud terminology . In the implementation of hosted business services, multiple tenants are often consolidated into the same database to reduce total cost of ownership. Despite of that, the current Schema-Mapping Techniques are still immature to allow tenants to extend the database schema. One problem is that Multi-tenancy makes it harder to support application extensibility, since shared structures are harder to individually be modified and these techniques offer only limited support for schema evolution DDL

commands over existing data, if they are supported at all, consume considerable resources and negatively impact performance.

Database as a service (DaaS) attempts to move the operational burden of provisioning, access control, configuration, scaling, performance tuning, backup, and privacy away from database users to the service provider. DaaS is so appealing because it promises to offer scalability as well as being an economical solution. It will allow for users to take advantage of the lack of correlation between workloads of different applications, the service can also be run using fewer machines than if each workload was individually provisioned for its peak [23].

The final concept to understanding cloud computing is the different infrastructure models, which consist of public, private, and hybrid clouds. Generally, third party vendors develop public clouds. In the majority of public clouds, database applications from multiple different customers are mingled together on the cloud's servers, storage system and networks. It's called Multi-tenant databases. The benefits of a public cloud is that it can be much larger than a private cloud. It has the ability thus to offer scaling up or down on demand.

In this research, we introduce a novel approach to support Multi-tenant schema inheritance in RDBMS for SaaS. Due to the Multi-tenant database should be configurable and extendable at runtime. Our approach is fulfill the expectations of various tenants by design different parts of the application, and automatically adjust and configure its behavior during the application's runtime execution without redeploy the application .

Objects and their fields are mapped to metadata tables. The database schema integrates Multi-tenant relational tables and virtual relational tables and makes them operate virtually as a single database schema for each tenant and make it a suitable for Multi-tenant database environment that can execute any business domain database.

1.1 Statement of the Problem

When hosting data in the Shared Table in the cloud systems , all tenants will share both physical database and schema, the problem is the inability to customize and enable extension dynamically by Schema-Mapping Techniques when the system is on-line without affect the logical schemas of other tenants.

1.2 Objectives

Achieving our objectives of the research depends on a deep study of the Multi-tenant databases . In this section, we present main objective and specific objectives of the research work.

1.2.1 Main Objective

The main objective of this research is to generate a new virtual schema that inherit both shared data and metadata from shared schema, Thereby, it allows extending tables and creating objects according parent schema of a Multi-tenant database system based on the standard RDBMS.

1.2.2 Specific objectives

There are real specific objectives extracted from the main objective such as:

1. Using database generator to generate and configure database schema to be more suitable for Multi-tenant database.
2. Fulfill the expectations of various tenants by allow each tenant to create custom extensions to standard data objects and entirely new custom data objects.
3. Selecting an engine that generates application components from metadata.
4. Integrating physical Multi-tenant relational tables and virtual relational tables and makes them operate virtually as a single database schema and make it a suitable for Multi-tenant database environment.
5. Design the proposed model that also includes supporting extension schema .
6. Implement the proposed model on virtual machine environment.
7. Testing and evaluating the performance of proposed model compared with the results of previous models.

1.3 Importance of the Thesis

Incremental advances in a cloud technology create major paradigm shifts in the way software applications are designed, and delivered to end users. The concept of Multi-tenancy has appeared, despite of its importance, it brings about several issues on security, implementation challenges, customization, configurability, scalability, and extensibility .

Therefore, the main importance of this research rises from:

1. Proposes a flexible way to creating database schemas for multiple tenants.
2. Offer a new technology to share data in a Multi-tenant database to avoiding storing redundant data .
3. Improves the Multi-tenant database performance and achieve high scalability .
4. Enhance TPC-H benchmark to suit cloud computing.
5. Allowing integration with Multi-tenant relational database .

1.4 Scope And Limitations of the Thesis

1. **Database consolidation:** We have not set a strategy for database consolidation and inject the old and the new data on the cloud server.
2. **Single server:** All our experiments were conducted on a single cloud server. Measure the performance of different servers outside the scope of our study
3. **STSI focus:** We focus on shared table shared instance , while shared machine outside the scope of our work.
4. **Structured data :** Our study was limited to structured data.
5. **Database migration**: The Migration from cloud provider to another cloud make the ensuring of user's data very hard, when CSP want to move the user's data files from one server to a new one, it may appears integrity breach.
6. **Same site :** Although we use a real cloud database , but tenant's access was from the same site.
7. **Schema mapping :** The focus is mainly on schema mapping techniques.
8. **BigData:** We have not performed any experience on BigData, due to it is out of the scope of this thesis.

1.5 Thesis Structure

This thesis consists of six chapters: introduction, theory background, related works, methodology and implementation, experiments and evaluation and finally, conclusions and future work.

The main points discussed in the chapters are listed below:

i. **Chapter 1** : **Introduction.**

ii. **Chapter 2: Theory Background**, presents an overview about cloud computing, and schema requirements for Multi-tenant Databases systems.

iii. **Chapter 3: Related Works.** it classified into three categories: companies, Multi-tenancy, schema-mapping technique.

iv. **Chapter 4: Methodology and Implementation,** methodology, model architecture and implementation.

v. **Chapter 5: Experiments and Evaluation*:*** the experimental works, starting from generate database , evaluating and analyzing the experimental results.

vi. **Chapter 6: Conclusions and Future Work*:*** presents conclusions and possible future works.

Chapter 2

Theory Background

In this chapter an overview about cloud computing will be presented, its definition, essential characteristics, deployments model and Cloud architecture key components. We also presented cloud computing benefits, Multi-tenancy , Multi-tenant data storage systems.

2.1 Cloud Computing Overview

"Cloud Computing is a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model . Cloud computing is composed of five essential characteristics, three service models, and four deployment models." [1]

2.1.1 Essential Characteristics

The cloud model is composed of five essential characteristics identified by NIST [1]:

1. **On-demand self-service.** A customer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

2. **Broad network access.** Network access is available over the network and controlled through standard mechanisms that promote access by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

3. **Resource pooling.** The provider's computing resources are pooled to serve multiple customers using a multiple-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to customer demand.

4. **Rapid elasticity.** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the customer, the capabilities available for provisioning often appear to be unlimited and can be requested in any quantity and at any time.

5. **Measured service.** Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts).

2.1.2 Deployment Models

There are four different deployment models as shown in figure 2-1 for implementing a cloud based solution.

1. **Private cloud**. The cloud infrastructure is maintained by the organization itself and is used exclusively by a single organization.
2. **Public cloud**. The cloud infrastructure is open for use by the general public .
3. **Community cloud**. The cloud infrastructure is maintained by one organization for a set of organizations (the community) and used by all of them.
4. **Hybrid cloud**. The cloud infrastructure is a composition of two or more distinct cloud infrastructures.

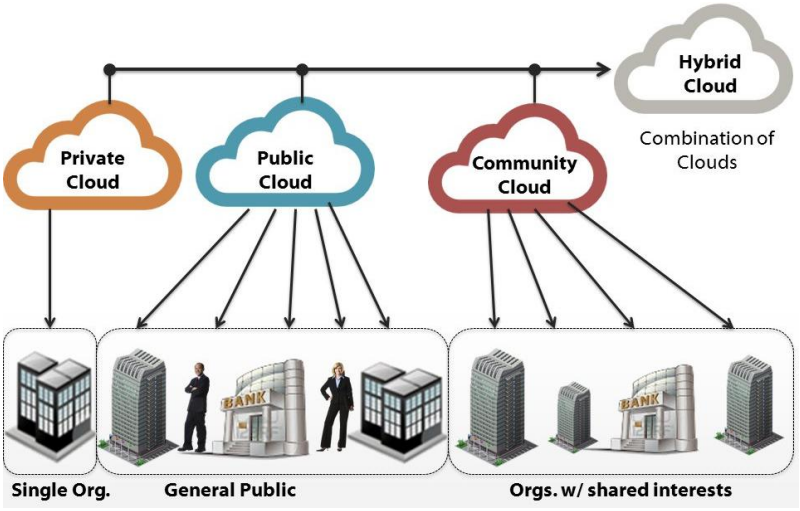


Figure 2-1: Deployment models [26]

2.1.3 Service models:

Cloud computing has three fundamental models of services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Figure 2-2 displays these services.

1. **Infrastructure as a Service (IaaS)** allows consumers to use hardware through commonly available interfaces such as Secure Shell (SSH) or a web browser. This services are provided by data centers to allow their tenants to deploy and run their operating systems and other applications on top of virtualized software.
   The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).)

2. **Platform as a Service (PaaS)** provides customers with a platform for executing and deploying services through a specific interface via a web browser. PaaS enables collaboration, so multiple users can work on the same application without any need for install or download the platform that is provided by the vendor , thus increasing productivity and reduces the cost on the tenants.

3. **Software as a Service (SaaS)** enables users to access the provider's applications running on a cloud infrastructure through a simple client interface, such as a web browser. SaaS applications are installed on remote machines, so that clients do not have to install them on every machine. SaaS allows tenants to subscribe to a paid software service instead of paying for software licenses.

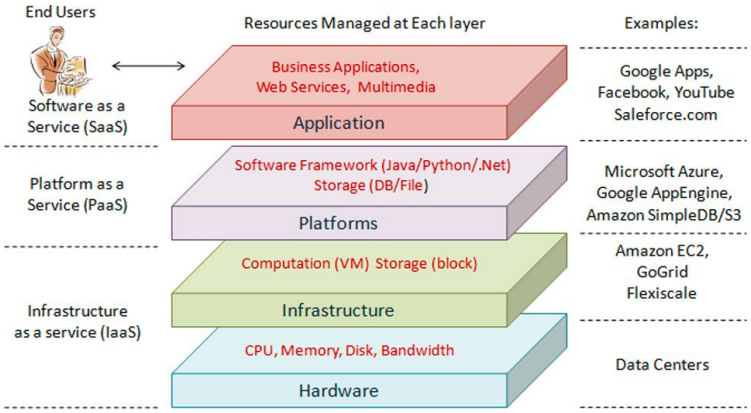


Figure 2-2 : Cloud Computing Architecture [27]

Software as a Service (SaaS) is the major focus in our thesis.

2.1.4 Cloud architecture key components

In the cloud architecture, key components are Cloud Service Provider, User (Data Owner) and Third Party Verifier.

1- Cloud Service Provider (CSP): CSP has significant resources.
2- User: User may be a person or an organization who has data to be stored in the cloud and rely on the cloud for data computation.
3- Third Party Verifier (TPV): TPV has expertise and capabilities that users may not have.

2.2 Cloud computing benefits

Many of the benefits when using Cloud Computing are the lower costs associated [29]. The following are some of the possible benefits for those who offer cloud computing-based services and applications [10]:

2.2.1 Cost Savings.

Companies can reduce their capital expenditures and use operational expenditures for increasing their computing capabilities. The provider is responsible for software deployment and maintenance with his own infrastructure. The user only pays for technical support [29].

2.2.2 Reliability.

Services using multiple redundant sites can support business continuity and disaster recovery.

2.2.3 Scalability/Flexibility.

Companies can start with a small deployment and grow to a large deployment fairly rapidly, and then scale back if necessary. Also, the flexibility of cloud computing allows companies to use extra resources at peak times, enabling them to satisfy consumer demands. Resources within the cloud can be treated as an `unlimited' medium [29].

2.2.4 Maintenance.

Cloud service providers do the system maintenance, and access is through APIs that do not require application installations onto PCs, thus reducing maintenance is required.

2.2.5 Mobile Accessible.

Mobile workers have increased productivity due to systems accessible in an infrastructure available from anywhere.

2.3 Cloud Storage Systems

Cloud Storage, Database as a service (DBaaS) and Data as a service (DaaS) are refers to using cloud service for data storing and management in the cloud. They differ on how data is managed and stored. Cloud storage is a new business model for delivering virtualized storage to customers on demand. The formal term proposed by the Storage Networking Industry Association (SNIA) for cloud storage is Data Storage as a Service (DaaS) – as

> *"Delivery over a network of appropriately configured virtual storage and related data services, based on a request for a given service level." [1]*

It is used mainly for backup purposes and data management. Google drive , Dropbox, iCloud etc. are popular cloud storage services [39].

Database as a service (DBaaS) offers complete database functionality , users can access and store their database on the cloud anytime from any place through Internet. Google's BigTable , Amazon's SimpleDB and Microsoft's SQL Azure are common examples for DBaaS. on the other hand, when installing a traditional database such as oracle DB , SQL Server and MySQL on cloud server, it can serve as a cloud database.

2.3.1 Multi-Tenancy

Cloud Service Providers (CSP) provide many services such as storage, platform and applications. The main benefit of Multi-tenancy is to reduce the operating costs of running software from the provider's perspective. Multi-tenancy is the main property of SaaS [7], it allows vendors to provide multiple requests and configurations through a single instance of the application. In this context, a customer is known as a "tenant". In the same way, a single database is shared amongst customers to store all tenants' data: this is known as "Multi-tenant database". This reduces operational and maintenance costs; offers more reliability. Multi-tenancy is a reference to the mode of operation of software where multiple independent instances of one or multiple applications operate in a shared environment. The instances (tenants are logically isolated, but physically integrated. The degree of logical isolation must be complete, but the degree of physical integration will vary. The more physical integration, the harder it is to preserve the logical isolation. The tenants (application instances) can be representations of organizations that obtained access to the Multi-tenant . The tenants may also be multiple applications competing for shared underlying resources . All this is achieved without changes of the application code to support each customer's individual needs. In order to achieve this, individual meta data for each client has to be stored and has to have impact on the way the system behaves.
Moreover, it can be applied in four software layers, including application, middleware, virtual machine, and operating system [38].

2.3.2 Multi-Tenant Database

Multi-tenant data management is a form SaaS , whereby a third party service provider hosts databases as a service and provides its customers with mechanisms to create, store and access their databases at the host site . In other words Multi-tenant databases is a feature that allows a single instance of an application to handle several end-users at the same time , this idea has been explored previously without any explicit connection with Multi-tenancy [12] .

D. Jacobs et al [12], Bezemer et al [43], are Summarizing the difference between traditional RDBMS and Multi-tenant database in four aspects:

1. Isolating tenant data to ensure that each tenant can access only his own data .
2. Ensuring that each tenant's data is secured .
3. Building robust Multi-tenant database structure.
4. Optimizing the performance of each tenant database.

2.3.3 Multi-Tenant data storage systems

The concept Multi-tenancy is not supported on the traditional DBMS, It is appeared after the spread of cloud computing. however, despite the importance of Multi-tenancy as we mentioned in the previous section, it brings about several issues on security, implementation challenges, customization, configurability, scalability, and extensibility which can be seen only upon the deployment on a data center [14] . A well-designed SaaS application should be optimized to support Multi-tenancy, scalability and configurability [16] . This leads to the implementation and adoption of an additional layer for the real data management. Application developers experience additional problems with Multi-tenant database architectures. not knowing the semantics and the relationships between data. Thus, they can no longer be used for optimization and consistency management. Scalability here refers to the ability of an application to support an increasing number of users without noticing a significant performance overhead [5]. Customization is concerned with the support of specific features of users or meeting service level agreement by the means of configurations. Due to the distributed and shared nature of Multi-tenant applications appropriate security policies should be devised to prevent unauthorized users from accessing other users' private data. Multi-tenant data architecture has two kinds : shared data and isolated data, there are three Approaches to Managing Multi-tenant databases as shown in figure 2-3 : shared machine, shared process and shared table processes [7] , these techniques also called Separate Databases , Shared Database - Separate Schemas and Shared Database- Shared . The most interesting technique is the last one which aims at creating only once the application schema and mapping all tenants directly to this schema by making use of one of the available schema mapping techniques. compared different approaches for the implementation of Multi-tenant databases can be summarized as follows [4,7] :

(a) **Separate Database:** In this approach, a separate database is assigned to each tenant for data storage. This is the simplest approach to data isolation Each database contains some metadata used to redirect each tenant to the correct database. This approach is considered expensive in both implementation and maintenance. Multiple customers share the same machine, therefore it called shared machine.

(b) **Independent Tables and Shared Database Instances (IDII):** In this approach all tenants share the same physical database, however, the schema different for each tenant. This approach is relatively simple to implement.

(c) **Shared Tables and Shared Database Instances(STSI):** In this approach all tenants will share both the physical database and the schema. Tables are shared by all tenants. Customers' information is separated using primary keys which are

specified in the database design. This approach is relatively economic because it supports a large number of tenants per database server .

Chosen the appropriate approach depends on different criteria such as the number of tenants for the data storage and the efficiency and the cost considerations of SaaS implementation . For example, the separate database approach is the appropriate solution for large organizations tenants who need to store large amounts of data. The same approach is also suitable if security and legal requirements are of high concern. On the other hand, the shared database – shared schema is the appropriate solution for individual tenants who have low amounts of data to store. Also, the same approach is the optimum solution in case of frequent changed applications [15].

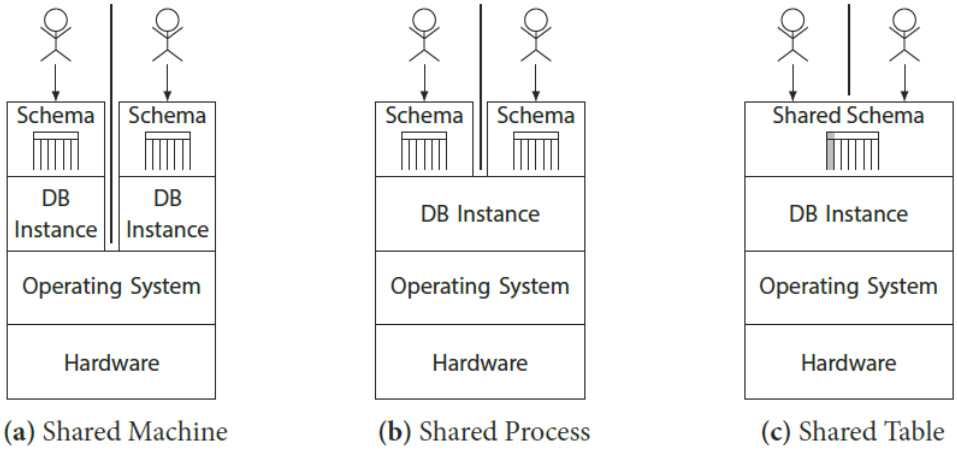


Figure 2-3. Types of Multi-tenant data storage systems [25]

2.3.4 Schema requirements for Multi-tenant Databases

Standard relational DBMSs have only very limited support for online schema evolution. For complex application updates there has to be a significant service downtime and even small schema changes, like the ones individual tenants initiate, have a severe performance impact, as stated in [9]. In turn a Multi-tenant DBMS needs to provide Schema Evolution capabilities. On the one hand, tenants need the ability to tailor the SaaS application to their needs without affecting other tenants. This may require schema modifications of already existing relations. On the other hand, SaaS applications are evolving constantly, as service providers are forced to integrate new features. These new features may require changes to the database schema. Consider, for example, a situation where the service provider needs to deploy a new feature of the base application which requires changes to the schema of existing relations. These changes could be performed online, as long as they do not require changes in the application code, e.g., adding attributes or enlarging the value range of an attribute.

Scalability, namely the ability to serve an increasing number of tenants without too much query performance degradation. One way to achieve high scalability is to offer a single instance of the software which serves multiple clients/organizations Multi-tenancy. By consolidating multiple customers onto the same infrastructure, resources can be economized and used more efficiently [7] .

Costs for third-party software licenses are, therefore, drastically reduced, allowing the saved money to be invested in bigger capacities of the existing infrastructure (e.g. more disk space, memory, etc.). Moreover, management processes can be enhanced while providing a uniform framework for system administration. In a Multi-tenant situation we cannot assume that the number of tenant will remain the same or that the tenant does not require more than one application and database server . The scalability implies that resources can be scaled-up or scaled-down dynamically without causing any interruption in the service[20]. It puts challenges on developers to develop databases in such a way that they can support and handle unlimited number of concurrent users and data growth. A Multi-tenant system should be able to support scale-up (consolidating multiple customers onto the same server) and scale-out (e.g. moving customers from an old data center to a new one) . Multi-tenant software should be able to deal with high complexity and rapid growth of customers, allowing them to have seamless migrations to servers which can meet the customers' SLAs.

2.4 Benchmarking database

Benchmarking a database is the process of performing well defined tests on that particular database for the purpose of evaluating its performance [31]. It also facilitate means for cross platform comparisons of various database. The Response time and the throughput are the two main criteria on which the performance of a database can be measured [30]. Specific parameters and settings external as well as internal to the database management system need to be taken into consideration. These parameters include the hardware used to test the system, the internal configuration of the database engine, the operating system configuration as well as the database design and implementation [31][32].

Nowadays, the design gap between OLTP-oriented and OLAP-oriented DBMS or data warehouses is even more prominent, since big data applications demands and performance requirements increase. OLTP-oriented DBMS deliver high

throughput for updates and index-based queries. OLAP-oriented DBMS are deliver high performance for complex analytical queries, and do not support transactional workloads for loading data.

Although there have been several works on how to build Multi-tenancy systems, little work has been done on how to benchmark and evaluate these systems partly due to the diversity of the systems and the complexity of possible benchmark setups. There are many well-accepted database benchmarks, e.g., TPC benchmarks like TPC-C or TPC-H [33] [34]. These benchmarks concentrate on a certain scenario and measure a system's peak performance with respect to the given scenario. The key challenge for Multi-tenancy systems is usually not to provide the highest peak performance, but to scale well and deal with multiple changing workloads under additional requirements like performance isolation and fairness.

In order to provide standards , The Transaction Processing Performance Council (TPC) defines transaction processing and database benchmarks that are widely used in industry and academia to measure performance characteristics of database systems. TPC is a non–profit corporation. The goal of TPC benchmarks is to define a set of functional requirements that can be run on any transaction processing system, regardless of hardware or operating system. TPC provides different benchmark suites designed according to specific workload type and applications requirements [33].

TPC benchmarks are used in evaluating the performance of computer systems; the results are published on the TPC web site. Today the most important of these benchmarks are TPC-C , TPC-H and TPC-VMS.

- TPC-C benchmark , which simulates Online Transaction Processing (OLTP) Systems, was applied in 750 performance evaluations which have been published over the past two decades across a wide range of hardware and software platforms. About a dozen database platforms have used TPC-C results in their publications.[36] . OLTP workloads are composed of short-lived transactions that read or modify operational data, and are typically standardized, submitted through application layers. The TPC-C schema consists of nine relations and five transactions that are centered around the management, sale and distribution of products or services. The database is initially populated with random data and then updated as new orders are processed by the system.

- The TPC Benchmark™H (TPC-H) is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data

modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. The benchmark specifies 22 queries on the 8 relations that answer business questions.

- The TPC Virtual Measurement Single System Specification (TPC-VMS) is using for adding the methodology and requirements for running and reporting performance metrics for virtualized databases. This benchmark is still under development.

It is of vital importance to use an appropriate benchmark to evaluate Multi-tenant database systems. Unfortunately, to the best of our knowledge, there is no standard benchmark for this task. The benchmark requires the development of a new class of DBMS that can efficiently support mixed (OLTP and OLAP) workloads processing common data of a common schema and administrative tasks .

Traditional benchmarks such as TPC-C [35] and TPC-H [36] are not suitable for benchmarking Multi-tenant database systems. TPC-C and TPC-H are basically designed for single-tenant database systems, and they lack an important feature that a Multi-tenant database must have the ability for allowing the database schema to be configurable for different tenants. Therefore, some institutions and universities have developed benchmark by following the general rules of TPC-C and TPC-H In order to obtain a hybrid benchmark.

In order to enhance the benchmark to suits with our work, we introduced simple modifications but important on some other related work. we have benefited greatly from the efforts to improve the benchmark in Munich Technical University, but the researchers are interested in the academic side, as well as the contribution of Mei et al.[5] , but they do not consider the extensibility issue of the shared table, which is the heart of our work. Our enhance benchmark Called SaTbencHCloud. We will explain it in detail in chapter four. Table 2.1 summarizes the major differences between OLTP and OLAP.

Table 2.1 major differences between OLTP and OLAP.

	OLTP System Online Transaction Processing (Operational System)	**OLAP System Online Analytical Processing (Data Warehouse)**
Source of data	Operational data; OLTPs are the original source of the data.	Consolidation data; OLAP data comes from the various OLTP Databases
Purpose of data	To control and run fundamental business tasks	To help with planning, problem solving, and decision support
What the data	Reveals a snapshot of ongoing business processes	Multi-dimensional views of various kinds of business activities
Inserts and Updates	Short and fast inserts and updates initiated by end users	Periodic long-running batch jobs refresh the data
Queries	Relatively standardized and simple queries Returning relatively few records	Often complex queries involving aggregations
Processing Speed	Typically very fast	Depends on the amount of data involved; batch data refreshes and complex queries may take many hours; query speed can be improved by creating indexes
Space Requirements	Can be relatively small if historical data is archived	Larger due to the existence of aggregation structures and history data; requires more indexes than OLTP
Database Design	Highly normalized with many tables	Typically de-normalized with fewer tables; use of star and/or snowflake schemas
Backup and Recovery	Backup religiously; operational data is critical to run the business, data loss is likely to entail significant monetary loss and legal liability	Instead of regular backups, some environments may consider simply reloading the OLTP data as a recovery method

Summary

This chapter aimed to review some background concepts about cloud computing. First, it reviewed its definition , essential characteristics, deployments model , service models.

Second, it reviewed cloud storage and Multi-tenancy and its role to reduces operational and maintenance costs and offers more reliability. we also discussed a Multi-tenant data storage systems, include the two kinds of Multi-tenant data architecture and the common approaches to managing cloud database. We have focused on shared tables and shared database instances approach (STSI), which means a single database schema is used to store data from different tenants.

We also presented schema requirements for Multi-tenant databases that are related to this study.

Finally, we describe the most important benchmarks used in evaluating the performance of databases.

Chapter 3

Related works

When we studies Multi-tenant database schema designs and schema mapping techniques , many challenges are raised. One of the major issues is extension shared table dynamically , A lot of works addressed this problem and introduced some solutions to overcome it. In this chapter , we review different related works. Some of them can be a basis for supporting us in our thesis problem.

During our review we found that some researchers studied the Multi-tenant database from the perspective of IT companies , others focused on the concept of multiple tenants , While others discussed schema-mapping techniques.
So, we classified the related works according to these categories:

1- Companies.
2- Multi-tenancy.
3- Schema-mapping technique.

For each category, we introduced some related works, and provided the disadvantages of each one. Finally we introduced a conclusion in order to overcome in our research.

i. Companies

Companies like force.com does its own mapping from logical tenant schemas to one universal physical database schema to overcome the limitations of traditional DBMSs. However, this approach complicates development of the application since many DBMS features, such as query optimization . Instead, a next-generation Multi-tenant DBMS should provide explicit support for extensibility [6].

BigTable [2] is developed and deployed by Google as a structured data storage infrastructure for different Google's products. To scale up the system to thousands of machines and serve as many projects as possible, BigTable employs a simple data model that presents data as a sorted map in which each value is an uninterrupted string . We see that although Google's BigTable is a high performance, distributed and proprietary storage system designed to easily manage structured data that scales across thousands of commodity servers , BigTable is currently not used nor distributed outside Google, although it can be accessed from Google App Engine. Since its release several open source implementations have been reported in the literature namely HBase and Hypertable.

Windows Azure is consider a Microsoft's cloud infrastructure platform, it has become a major part of Microsoft's overall strategy. Windows Azure Storage is aim to let users and

applications access their data efficiently from anywhere at any time using simple API. It supports structured, unstructured data and NoSQL.
We see that the drawback of windows azure supports up to 150 databases , and this limit applies to all service tiers . In addition to limiting the number of databases per server, each service tier (edition) limits the maximum size of each database. If the size of the database reaches its MAXSIZE, you will receive an error code. [40]

ii- Multi-tenancy

Bezemer, et al.[19] gives a very clear introduction to Multi-tenancy, it defines the term and shows its main characteristics. In order to do research on Multi-tenancy, the authors aim to introduce the term Multi-tenancy and compare it against multi-user and multi-instance. They use two definitions for Multi-tenancy. The first definition states: "a Multi-tenancy application lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment.". The second definition states that "a tenant is the organizational entity which rents a Multi-tenancy SaaS solution. Typically, a tenant groups a number of users, which are the stakeholders in the organization."
we think that a deep understanding of the term and its main characteristics is required , because it is unclear at this point where the border is between Multi-tenant, multi-user and multi-instance, the authors make two short comparisons between each versus Multi-tenant to make the difference between them clear. Firstly, they compare Multi-tenant versus multi-user and state that in a multi-user application, are using the same application and are limited when it comes to configuring it. However, in a Multi-tenant application, each tenant can configure the application and the options to do so are not restrictive as in a multi-user environment. That means besides how it looks, the application can behave differently for multiple tenants while it will always behave the same for multi-user.

Curino et al , Moon et al shows that schema evolution is still an important topic, especially in scenarios where information systems must be upgraded with no or less human intervention . In their view, Multi-tenancy is efficient when giving a set of databases and workloads, it can be determined what the best way is to serve them from a given set of machines. Relational Cloud stores the data belonging to different tenants within the same database server, but does not mix data of two different tenants into a common database or table. [11,3 ,6].
From our point of view that the current database systems do not understand Multi-tenancy, therefore, the Multi-tenant application needs to be able to create tenants to the database and associate every record with the tenant id. Moreover, the application needs to adapt queries in order to only fetch data for a specific tenant, and to restrict the logged-in user from accessing data from other tenants.

In [7] Jacobs et. al discusses the trend towards Multi-tenancy for hosted applications and some main requirements, while comparing some implementations and showed the different possibilities in implementing Multi-tenant databases on standard relational

databases. They identified three approaches are: shared machine, shared process, and shared table. In the shared machine approach each tenants get their own database. The resource sharing is done on machine level. In the shared process approach the tenants share the same physical database process but own different databases. This allows better resource pooling between the tenants but still creates a lot overhead because the schemas need to be maintained separately for all tenants. The last approach is the shared table approach. however in [21] author presents a more deep comparison towards Multi-tenancy for hosted applications and some main requirements .

iii- Schema-Mapping Technique

In this section we outlines some common schema-mapping techniques for Multi-tenant database, we will use an example from [4,44] to comparison of flexible schemas for SaaS. The example show Account tables of three tenants with IDs 17, 35, and 42. Tenant 17 has an extension for the health care industry while tenant 42 has an extension for the automotive industry.

The Private Tables technique is the basic way to support extensibility, this technique provides a high level of isolation between tenants, it's allows each tenant to have his own private tables, which can be extended and changed [4] . The query-transformation layer needs to rename tables , since the meta-data is managed by the database , thus there is no overhead for meta-data in the data itself. Show figure 3-1.
The main drawback in this technique that many tables are required to satisfy each tenant needs. Therefore, this technique is unfavorable when hosting a large number of tenants.

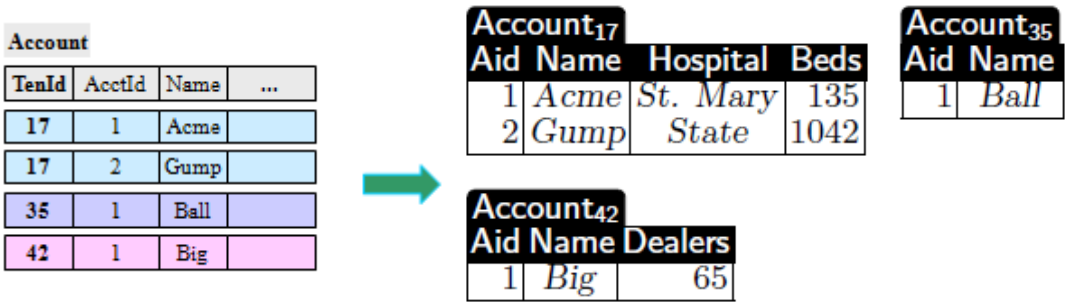


Figure 3.1: Private Table Layout [4,44]

Extension table has its origins in the decomposed storage model [41] which splits up a table of n columns into n tables of 2 columns as shown in figure3-2 .Multiple tenants can use the base tables as well as the extension. The main drawback of this approach is that reconstructing the logical source tables carries the overhead of additional joins , in additional the increase in the number of tenants will lead to increase the number of tables will have a wider variety of basic requirements.

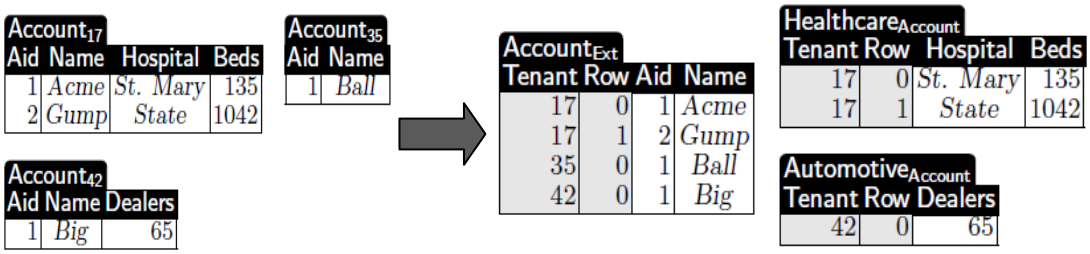


Figure 3.2: Extension Table Layout [4,44]

The universal table layout allow the creation of an arbitrary number of tables that holds a large number of generic data columns with a Tenant column and Table column. mostly, the data columns have VARCHAR datatype, it is a flexible type, and it can be used as an intermediary for the conversion from any datatype to another. The n-th column of a logical source table for each tenant is mapped to ColN in the universal table. In addition, two unique columns, Tenant_id and Table columns are used: Tenant_id identifies tenants from each other, whereas the Table column identifies the specific table of the same tenant.

Each tenant fills his columns with the needed data. The rest of the columns that are not related to him are filled with Null values. Figure 3-3 illustrates the implementation of universal table.

Universal

Tenant	Table	Col1	Col2	Col3	Col4	Col5	Col6
17	0	1	Acme	St. Mary	135	—	—
17	0	2	Gump	State	1042	—	—
35	1	1	Ball	—	—	—	—
42	2	1	Big	65	—	—	—

Figure 3.3: Universal Table Layout [4]

Pivot tables are shared between all tenants, each row field in a logical source table is given its own row. The pivot table including five columns: tenant, table, row , column and single data type column to stores the values of the logical source table rows according to their data types in the designated pivot Table. Figure 3-4 illustrates the implementation of Pivot table.

However the drawback of this approach that to reconstructing table requires more columns of meta data than actual data since rebuild an n-column logical source table requires (n − 1) aligning joins along the Row column.

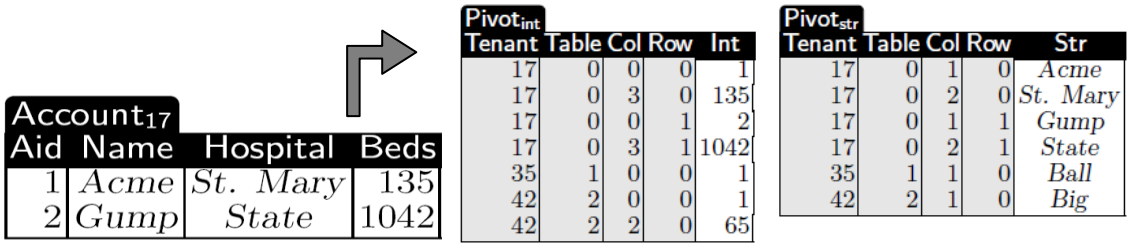


Figure 3.4: Pivot Table Layout [4]

S. Aulbach et. al [4], presented a Chunk Folding approach that is a schema-mapping technique . The approach works by vertically partitioning logical tables into chunks that in turns are folded together into several physical Multi-tenant tables and joined as needed. Show figure 3-5. The authors say that it is often when developers choose to map many single-tenant logical schemas in the application to one Multi-tenant physical schema in the database. Despite the fact that it is not easy to do this mapping, the benefits of consolidating hundreds of databases into one will save millions of dollars per year, say the authors. There is however a downside of Multi-tenancy, which is the sharing of resources. The aim of Chunk Folding is to reduce the complexity of scaling a SQL database. The authors say that the performance begins to degrade when over about 50,000 tables are used on one server. An option to mitigate the performance downgrade, is to share the tables among tenant , since some of the clients use certain features that others are not, and using all tenants data in a shared table. However, the authors do not fully agree with this approach either because in their view "the mapping techniques used by most hosted services today provide only limited support for extensibility and/or achieve only limited amounts of consolidation".
We believe that the disadvantage of this approach that it requires partitioning the tables into chunks; and the of joined as needed , this causes reduced performance when a large number of tables. However, the main weakness of the Chunk Folding technique is that the shared schema between multiple tenants must be known in advance, which is not a practical solution for Multi-tenancy.

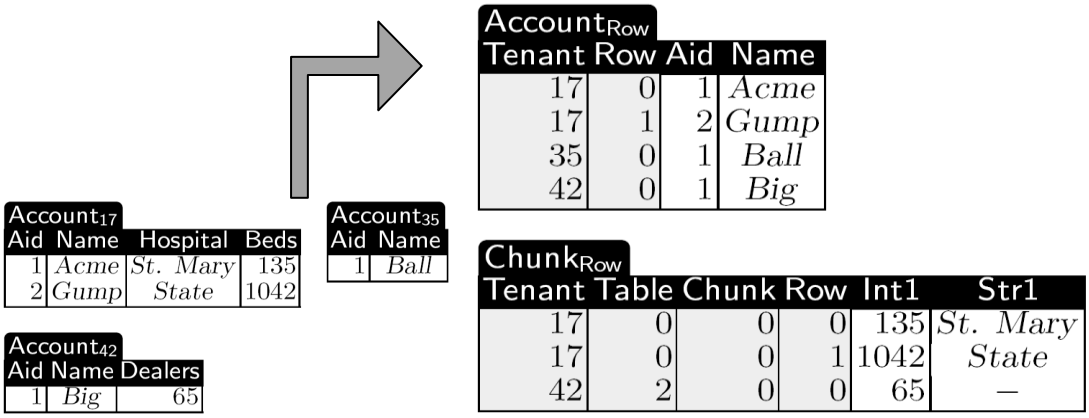


Figure 3.5: Chunk Folding Layout [4]

The extensible markup language (XML) database extension technique is a combination of relational database systems and XML [10,41] . extension of XML achieved by storing the XML document in the database as a Character Large Object (CLOB) or Binary Large Object (BLOB) as shown in figure 3-4. Tenants specific data be handled without changing original database relational schema. We believe that this approach adds many advantages to the database such as simplicity in the implementation and flexibility , however it's performance is affected by data structure [18].

Account

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

Figure 3.6: Extension XML Layout [44]

Franclin S. Foping et. al [10] have been contributed a new approach focuses on devising a mechanism to handle data between the real physical tables and the tenant tables including options for tenant schema extension but can be implemented in open source relational database products .

in [9] Stefan Aulbach et al. introduce features like native schema flexibility which is handled by prototype data model called FlexScheme which is optimized for a Multi-tenant workload they describe a method for graceful on-line schema evolution without service outages.

In[24] Schiller, et al. propose the concept of a tenant context to isolate a tenant from other tenants. They present a schema inheritance concept that allows sharing a core application schema among tenants while enabling schema extensions per tenant. They introduce a tenant context concept to determines the tenant's view of the database, and a tenant-aware schema inheritance for sharing of the application's core schema that is invariant among tenants while allowing extensions schema for tenants according to their individual needs. They have contributed to eases the development of Multi-tenant SaaS applications , and facilitates the maintenance of the application core schema .

We see that this approach need some features such as further tenant-aware data management , backup and recovery or migration of a tenant to another applications.

Summary

In this chapter we studied the research areas that are related to our work , each study which provided some solutions, but it is not enough, in [4, 6,40] , It has been overcome the limitations of traditional DBMSs, but not provide explicit support for used nor distributed outside its products.

In[4,7,10, 41] a Multi-tenant databases have been studied extensively. it's gaining much significance especially Schema Evolution, so many of authors focus on Schema-mapping techniques and sharing metadata among tenants. SaaS applications faced important challenge consolidate multiple tenants onto one database server . In addition, the application itself manages and handles metadata, but it is loses knowledge about the data and the relations, this causes the data redundancy and complexity of application development.

In this research, we analyze critical operations in the area of Multi-tenant database. Based on the results we propose a schema inheritance concept for a Multi-tenant storage capable to overcome the problems in this area .

Chapter 4

Methodology and Implementation

In this chapter we shall describe our proposed approach SaTbencHCloud , to produce the main objective of our research, then we shall describe the implementation in details in next sections.

4.1 Research Methodology

In this section, the proposed model methodology were presented and built the model to evaluate the efficiency and scalability in the cloud database. One of the key issues is to extension shared table dynamically when the database system is on-line without the need to shutdown database. To address this problem, the methodology consists of four main phases as follows:

- Research and review.
- Process the model.
- Implementation of the model.
- Experiments and Evaluation and analyzing the experimental results. It will be discuss in the details in chapter 5)

4.1.1 Research and Review

Reviewing the recent researches on how to build Multi-tenancy systems that is related to our problem . Studying the existing approaches, and noting the disadvantages of each method in order to overcome in our research. On top of that, our approach takes advantage of the of universal table and pivot table, which gave him a unique and efficient approach versus other approaches. When we reviewed most previous studies we found that there are very important contributions that we can introduce in this research. Unlike previous studies which we discussed in Chapter 3, we are going to build the model, and we shall consider these issues:

a. Benchmarking and evaluating the Multi-tenancy systems Although the diversity of the systems and the complexity of possible benchmark setups.
b. Allowing the database schema to be configurable for different tenants without shutdown database .
c. Preparing the workload with small, medium, large and very large databases with any generic relational database schema and SQL queries.
d. Supporting high availability by make the database continuously available 24 hours a day, 7 days a week.

4.1.2 process for Running the SaTbencHCloud

To produce the main objective of our research, we setting up the SaTbencHCloud , which contains: schema generation , database generation , query generation , and driver . The model will appear as shown in figure 4-1.

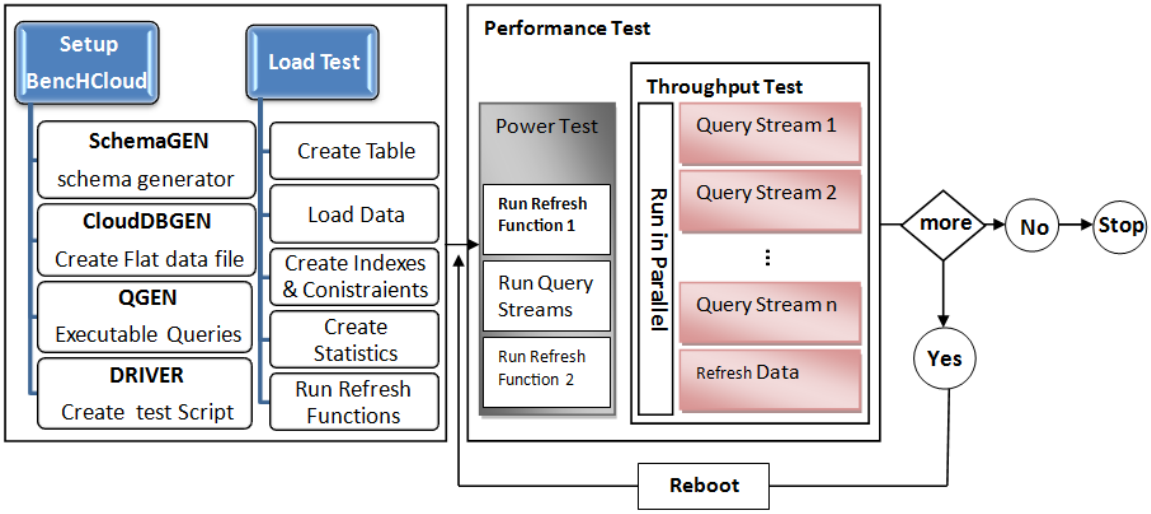


Figure 4.1: Complete process for running the proposed model (SaTbencHCloud)

4.2 Setting up the SaTbencHCloud

When benchmarking Cloud database, TPC-H is probably the first choice. But the default TPC-H transactions are not very usable when you need to test a Multi-tenancy workload. Therefore, We have introduced some updates to this benchmark for compatibility with our work by following the general rules of TPC-C and TPC-H . Our version of benchmark called SaTbencHCloud , it focus on a cloud environments with Multi-tenancy support .

The primary goal is to measure the impact of an increasing number of sessions on the reactivity (i.e. the response time) of the system while increasing the number of tables and tenants. and executes a complex mixed workload: a transactional workload based on the order entry processing of TPC-C and a corresponding TPC-H-equivalent OLAP query suite run in parallel on the same tables in a single database system .

4.2.1 Overview of the SaTbencHCloud

There are number of advantages when using SaTbencHCloud . First, the workload prepared to include a number of representative examples of ad-hoc queries with various levels of complexity. Second, SaTbencHCloud is suitable with small, medium, large and very large databases. Third, it can work in different systems. Our SaTbencHCloud benchmark comprises four modules as shown in figure 4-1. It Includes configurable database base schema with a private schema generator, a data generator, a query workload generator, and a driver. SaTbencHCloud can be used with any generic relational database schema and SQL queries.

One of the major advantages of our approach that uses two related works, they are pivot table and universal table.

4.2.2 Configurable Base Schema

The SaTbencHCloud benchmark executes a mixed workload (TPC-C for OLTP and TPC-H for OLAP). But we follow the logical database design of TPC-H to generate the configurable database base schema because it is more suitable for Multi-tenant database.
There are numerous advantages of using TPC-H , the TPC-H schema size is not fixed, it can be manipulated trough a scale factor, so schemas can be either small or large depending on the system that you want to test. While performing TPC-H you will create a performance profile, based on the execution time of all 22 TPC-H queries. To calculate the Composite-Query-per-Hour Performance metric (Qph@Size), we need to know the following things:

- Database size
- Processing time of queries in a single tenant case (Power test)
- Processing time of queries in a Multi-tenant case (Throughput test)

SaTbencHCloud benchmark produces results that are highly comparable to both TPC-C and TPC-H. The database is continuously available 24 hours a day, 7 days a week, for ad-hoc queries from multiple end users and data modifications against all tables.

Figure 4-2 illustrates the table relationships in TPC-H database, it designed to be in the third normal form.

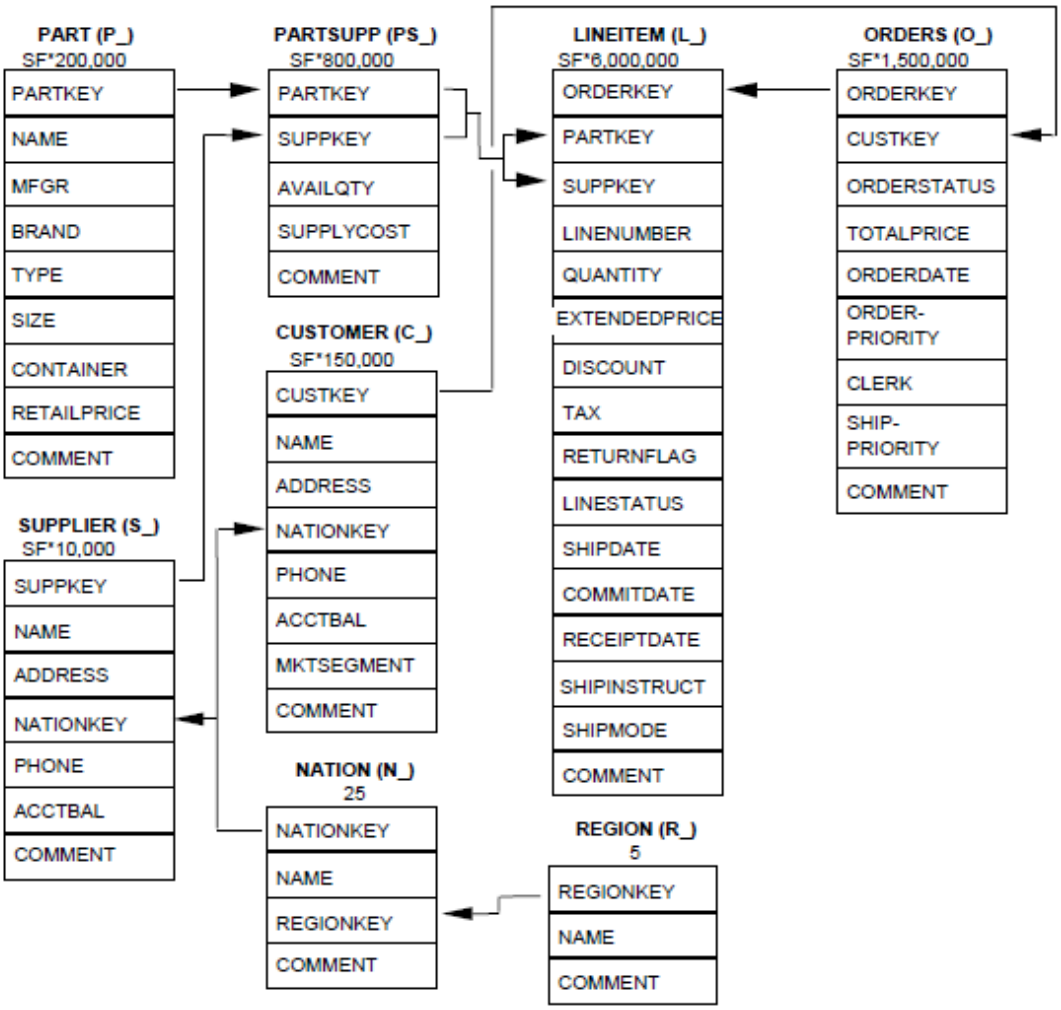


Figure 4-2: The TPC-H Schema [26]

4.2.3 SchemaGEN

TPC-H provide a Schema, it is found in the files tpch_dll.sql and tpch_ri.sql. These files contain the ANSI-SQL compliant schema and should work with most database using only minor modifications. We working modified schemas for different DBMS to be suitable with Multi-tenant database , a Tenant_id column is added to every table . Consequently, the primary key has to be a combination of the Tenant_id and the entity specific id field. We use Oracle Database 12c , it is complete with innovative Multi-tenant architecture and designed for the cloud. We use the schema generator called SchemaGEN to produce the schema for each tenant.

4.2.4 CloudDBGEN

CloudDBGEN is a bundle contains a bunch of C files (CSV format) to be compiled to form database generator. it use to populate the database with data, add constraints (primary keys, foreign keys and check constraints). it has a scaling factor that influences the amount of data generator - the default value (1) means about 1GB of raw data. CloudDBGEN is essentially an extension of DBGEN tool equipped with TPC-H. It actually uses the same code of DBGEN to generate value for each attribute. Important modification is that CloudDBGEN generates data for each tenant According to the tenant account . CloudDBGEN has been tested on a variety of platforms with change some parameters in the C file, it is run correctly . The following example shows sample code for run benchmark on Windows.

```
 [oracle@localhost tpch]$ cp makefile.suite makefile

[oracle@localhost tpch]

################

## CHANGE NAME OF ANSI COMPILER HERE

################

CC      = CL
# Current values for DATABASE are: INFORMIX, DB2, TDAT (Teradata)
#                                  SQLSERVER, SYBASE, ORACLE, VECTORWISE
# Current values for MACHINE are:  ATT, DOS, HP, IBM, ICL, MVS,
#                                  SGI, SUN, U2200, VMS, LINUX, WIN32
# Current values for WORKLOAD are:  TPCH
DATABASE= ORACLE
MACHINE = WIN32
WORKLOAD = TPCH
…
```

Figure 4.3: sample code for run CloudDBGEN on Windows.

4.2.5 Generating Dataset

To accomplish our work , first we generate a dataset , our dataset must contain different sizes . First, by running SchemaGEN we generate 3 groups of schemas for 100, 500, 1,000 tenants. These schemas are then used for evaluating the scalability of storage and query processing under different schema variability.

Next, we run CloudDBGEN to generate data for three different databases named “SaTbencHCloud_10GB”, SaTbencHCloud_100GB” and SaTbencHCloud_300GB” were respectively generated with the TPC-H workloads of scale factor 10, 100, and 300. As required by the TPC-H specification, the three different scale factors were selected in order to observe significant differences in query response between these three different scale factors . In table 4.1, we can see the different size of databases generated .

Table 4.1: size of databases generated

Database Name	# of Tenant	Scale Factor
SaTbencHCloud_10GB	100	10
SaTbencHCloud_100GB	500	100
SaTbencHCloud_300GB	1,000	300

Table 4.2 shows the different sizes of the database according to specific scale factor. Scale factors used for the test database must be chosen from the set of fixed scale factors defined as follows:

Table 4.2: Scale factors used for the test database

Scale factor (SF)	1	10	30	100	300	1000	3000	10000	30000	100000
Database sizes	1GB	10GB	30GB	100GB	300GB	1000GB	3000GB	10000GB	30000GB	100000GB

Tables have different sizes except nation and region, change proportionally to a constant known as scale factor (SF), as seen in Table 5.2. The two largest tables are Lineitem and Orders and hold about 83% of the total data.
The minimum required size for a test database is (1GB) , it holds business data from 10,000 suppliers. It contains almost ten million rows representing a raw storage capacity of about 1 gigabyte[36]. since scale factor is setting to 1. Any database size not mentioned is not permitted by the TPC. This requirement is meant to encourage comparability of the results and to ensure a significant actual difference in test database sizes [36]. According to normalization theory , the TPC-H benchmark database has been followed the third normal form . The data scale of tables for each tenant is illustrated in Table 4.3.

Table 4.3: Data scale of tables for each tenant

Table Name	Cardinality	Scale Factor 10 (10GB)
PART	SF*200,000	2,000,000
PARTSUPP	SF*80,000	800,000
LINEITEM	SF*6,000,000 (approximate)	60,000,000
SUPPLIER	SF*10,000	100,000
CUSTOMER	SF*150,000	1,500,000
ORDERS	SF*1,5000,00	15,000,000
NATION	25	25
REGION	5	5

4.2.6 Metadata-Driven Architectures

To fulfill the expectations of various tenants and their users, a Multi-tenant application must allow each tenant to create custom extensions to standard data objects and entirely new custom data objects. Inherently, Multi-tenant

applications are dynamic and polymorphic. For these reasons, a Multi-tenant application designs using a runtime engine that generates application components from metadata, that is a data about the application itself. Objects and their fields are mapped to metadata tables. This section proposes Metadata-Driven Architectures to build a Multi-tenant database schema . This database schema integrates Multi-tenant relational tables and virtual relational tables and makes them operate virtually as a single database schema for each tenant and make it a suitable for Multi-tenant database environment that can execute any business domain database. Figure 4-4 shows the details of metadata-driven architectures that is very significant for Multi-tenant applications. Table 4.4 brief description about metadata-driven fields.
The maximum number of tenants that can be supported by a Multi-tenant application can be increased as long as the resources increased while keeping the performance metrics of each tenant [37].

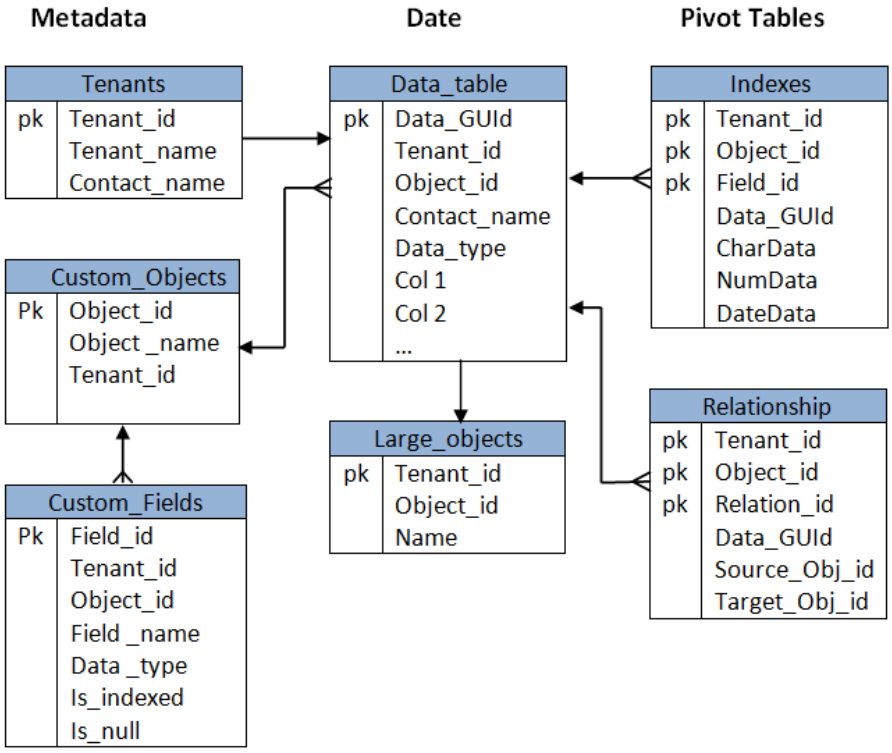


Figure 4-4: Metadata-driven Database Design

On our proposed architecture every logical database object is internally managed using metadata. The architecture details and listed as follows :

- Metadata: Tenant table is associated with a specific tenant and keeps all information that allows determining the tenant's virtual database with custom objects "Tables" and their fields are mapped to metadata tables.
- Data: Actual data is stored in a shared data table, On the other hand, the large objects are storage in a separate large object storage area.
- Pivot Tables : Index pivot table , improve and speed up the query execution time when retrieve data. To support multiple tenants, the object and field metadata

contains information about the fields, and also about the tenants. Relationship pivot table , allows tenants to create virtual relationships.

Table 4.4 - Brief description of the schema for a metadata-driven architectures

Table	Field	Description
Tenant	Tenant_id	ID of the tenant
	Tenant_name	Name of the tenant
	Contact_name	Contact name
Custom_Objects	Object_id	The unique ID of the object that contains this field
	Object _name	Name of the object " tenant virtual table "
	Tenant_id	ID of the tenant
Custom_Fields	Field_id	A unique identifier, the primary key for metadata table
	Tenant_id	ID of the tenant
	Object_id	The unique ID of the object that contains this field
	Field _name	The name of the field
	Data _type	The datatype of the field
	Is_indexed	A Boolean value representing whether an index needs to be created for this field
	Is_null	A Boolean value (flag) if the field is null
Data_table	Data_GUId	Global unique identifier
	Tenant_id	ID of the tenant
	Object_id	The ID of the object this datum is associated with.
	Contact_name	Natural name
	Data_type	Type of data
	Col 1 .. Col n	Values of the fields. Each value is mapped to a field as specified by the FieldNum value in the Custom Field Metadata table.
Large_objects	Tenant_id	ID of the tenant
	Object_id	The ID of the object
	Name	Name of the object
Indexes	Tenant_id	ID of the tenant
	Object_id	The ID of the object
	Field_id	Sequence number
	Data_GUId	Global unique identifier
	CharData	Character index since the datatype is string.
	NumData	Numaric index since the datatype is number.
	DateData	Date index since the datatype is date.
Relationship	Tenant_id	ID of the tenant
	Object_id	The ID of the object
	Relation_id	The ID of relationship
	Data_GUId	Global unique identifier
	Source_Obj_id	The ID of source object
	Target_Obj_id	The ID of target object

4.2.7 Integrated TPC-H Schema and Multi-tenant Relational Database

To give tenants the opportunity of satisfying their various requirements, a Multi-tenant database application must allows each tenant to configure his database schema and to extend an existing database schema during the application's runtime, this is call tenant-aware data management. A Multi-tenant aware application allows each tenant to design different parts of the application, and automatically adjust and configure its behavior during the application's runtime execution without redeploy the application[17]. This enables some optimizations such as provide high degrees of sharing data and suitable management features of recovery and backup. Moreover, it avoids redundancy to improve scalability and to reduce the per-tenant costs. Tenants must use resources in shared pool , but logically separate by means of virtualization. From a conceptual view, each tenant requires a virtual database that logically contains the objects that relate to the tenant and logically isolates it from other tenants [24]. Configuring Multi-tenant aware applications is a tenant self service that typically performs while applications are in operation to minimize system downtime, and allows the tenant to feel as if he is the only one using the application [42]. In the next sections, a detailed example is presented to explain how the tenants can integrated TPC-H Schema and Multi-tenant relational database.

For example, if a service provider offers a TPC-H database schema to be used by multiple tenants that fulfill various tenants' business requirements. This example assumes that the service provider has three tenants. The first tenant evaluated the TPC-H database, and he found that this database suits his business requirements. Therefore, this tenant was interested to use the original database schema. For simplicity we will use the orders table only as shown in figure 4-5 (a).

The second tenant found that he needs to use the columns that predefined in the order table add new fields to fulfill his business requirements. It including 'Ship Country' and 'Required Date'. Figure 4-5 (b) represents this case.

The third tenant found that he needs to add extra table. Thus, this tenant created virtual database relationship between the already existing physical tables and his add extra table as shown in figure 4-5 (c).

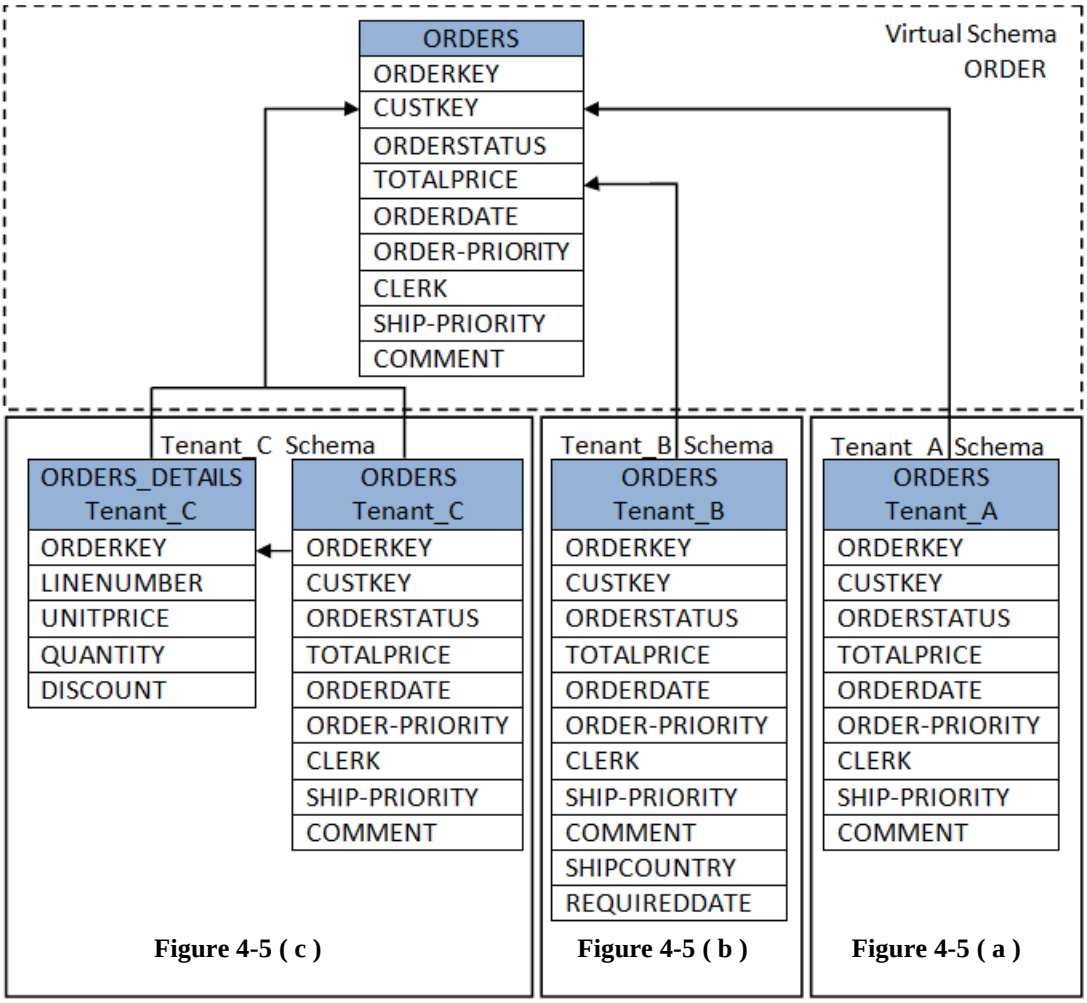


Figure 4-5: Integrated TPC-H Schema and Multi-tenant relational database

4.2.8 **Multi-tenant Architecture with Metadata Sharing**

Based on the objectives of our research, we will use the following steps:

1. First, through a tenant context, the system generates a virtual schema from a shared schema, this schema will inherits all the DB objects contained in the parent shared schema .
2. By extraction a data dictionary Associated with a tenant from the overall data dictionary we can managing a tenant metadata, it is isolated from global data dictionary , thus the schema modifications of a tenant will not affect the logical schema of other tenants.
3. To enable data sharing, a new table (Tenant_Heirarchy) is used to store the relationships between tenants based on the original table that stores information about the tenants.

Tenants Table : is used in Multi-tenant database to store the essential information about tenants.

Tenant_Heirarchy Table: is a child of tenant table, it use to store the relationships between tenants (primary and secondary tenants).

Secondary tenant: wants to share data owned by the primary tenant whose currently logged in.

Require mappings: is a flag indicates if a secondary tenant need mapping with a primary tenant.

4. Then, use Data Sharing Middleware to achieve mapping between tenant schema and virtual schema. Figure 4-6 represents these Multi-tenant application framework.

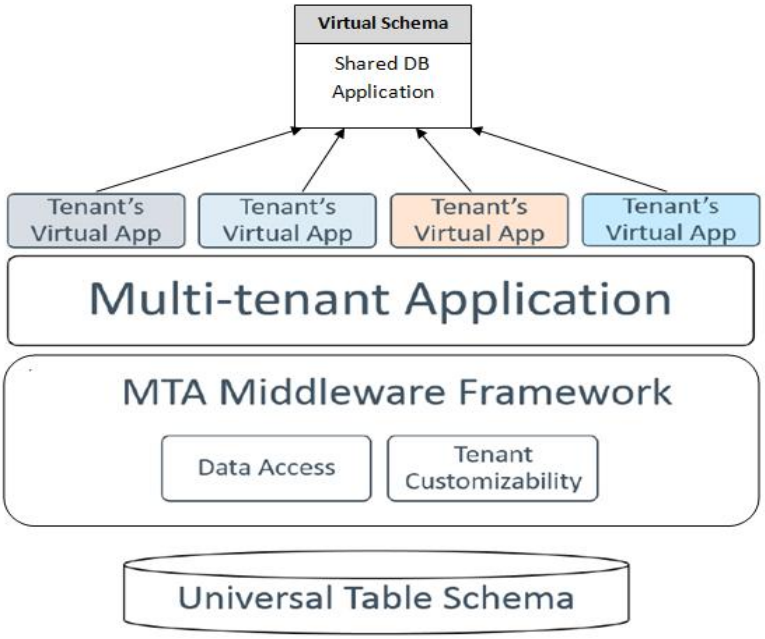


Figure 4-6: Multi-tenant Application Framework (taken from similar work with improving).

5. In the next step , the Mapping Manager module will creates and manipulates a mappings between the tenant tables. Mapping table Structure is shown in table 4-5.

Table 4.5: Structure of Mapping Table

Mapping			
mapping_Id	tenant_heirarchy_id	primary_table	secondary_table
1	1	Orders	Item $_{Tenant\_a}$
2	1	Orders	Item $_{Tenant\_b}$

6. Next, the entity relationship diagram of the architecture will be something like figure 4-7.

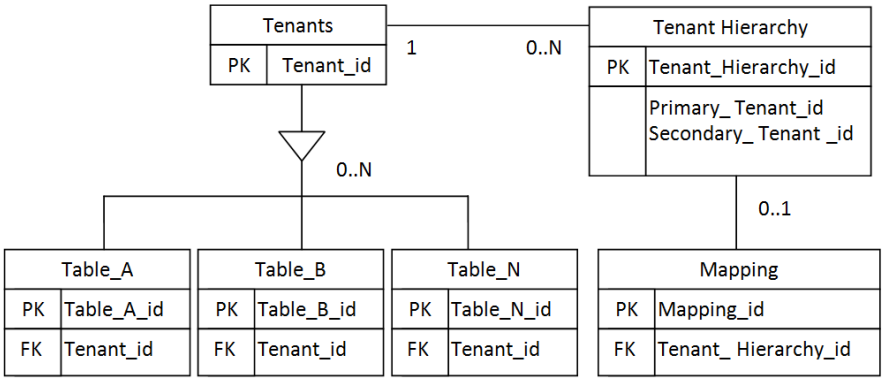


Figure 4-7 : Class diagram for Multi-tenant Architecture with metadata Sharing

4.2.9 QGEN

QGEN is a utility provided by the TPC to generate executable query text. It is written in ANSI'C' and has been ported to a large number of platforms [35]. The qgen data set contains 150 files with query substitutions values for all 22 queries for each scale factor as generated with qgen. Each file uses a different seed . The only difference is that the query optimizer add the clause RESTRICT ON TENANT statement in the query to indicate which tenant does the tuples belong to.

The 22 queries answer questions in areas such as pricing and promotions, supply and demand management, profit and revenue management, customer satisfaction, market share, shipping management. The refresh functions are not meant to represent concurrent on-line transaction processing (OLTP); they are meant to reflect the need to periodically update the database.

4.2.10 Third Party Driver

The mechanism used to submit queries and refresh functions to the system under test (SUT) and reports the execution time and throughput of the system , and measure their execution time is called a driver. The driver is a logical entity provided by the TPC , it can be implemented using one or more physical programs, processes, or systems .

Despite the fact that TPC-H benchmark offers a rich environment representative of many decision support systems, this benchmark does not reflect the entire range of decision support requirements. Since the TPC does not currently provide a readily available benchmark kit for Multi-tenant databases , a third party benchmarking software tool "Benchmark

Factory" was used to generate the TPC–H database workload rather than a driver.

Benchmark Factory for Databases is a database performance testing tool that enables you to conduct database workload replay, industry-standard benchmark testing, and scalability testing. Using the incorporated load testing tools, you can make changes to your database environment, which is typically hard to achieve in a standard testing environment, while mitigating the risks of unavoidable database changes such as patches and upgrades, operating system migrations, and adjustments to virtual machine configurations. In addition, its supports Oracle ,SQL Server , DB2, Sybase, MySQL, and other databases .

Benchmark Factory allows you determine system throughput and capacity for database systems and simulate thousands of concurrent users with a minimal amount of hardware, show figure 4-8. All test results are collected and stored in the repository for data analysis and reporting. Benchmark Factory collects a vast amount of statistics, including overall server throughput (measured in transactions per second, bytes transferred, etc.) and detailed transaction statistics by Benchmark Factory is a database performance and code scalability testing tool that simulates users and transactions on the database and replays production workload in non-production environments. This enables developers, DBAs, and QA teams to validate that their databases will scale as user load increases, application changes are made, and platform changes are implemented.

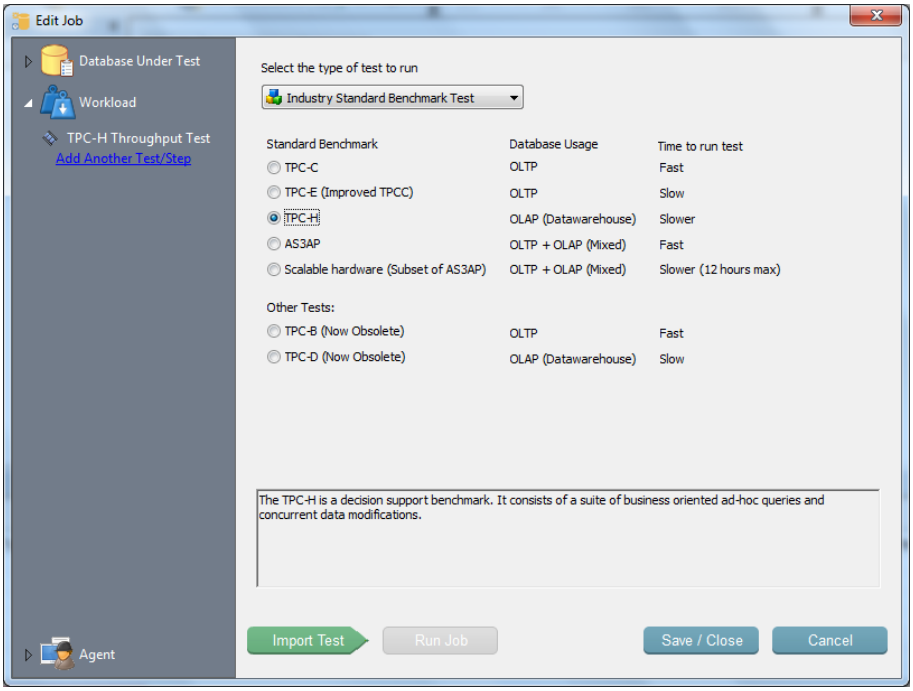


Figure 4-8: Benchmark Factory for Databases is a database performance testing tool

Summary

This chapter has presented our research methodology for an efficient approach for supporting Multi-tenant schema inheritance in RDBMS for SaaS. Our work was in the context of schema mapping techniques. We believe that we have given a contribution in managing metadata by allowing to extend shared tables and creating objects according parent schema of a Multi-tenant database system based on the standard RDBMS.

We explained in detail how to enable each tenant to make a configuration on a shared table on-the-fly. In order to achieve integration between the tenants, our approach allowed to integrate physical Multi-tenant relational tables and virtual relational tables and makes them operate virtually as a single database schema and make it a suitable for Multi-tenant database environment. We described the benchmark which we use , workload and main modules , and how it works.

Chapter 5

Experiments and Evaluations

This chapter describes the information needed to empirically evaluate the efficiency and scalability of the SaTbencHCloud approach. The main objective of this research is to generate a new virtual schema that inherit both shared data and metadata from shared schema, Thereby, it allows extending tables and creating objects according parent schema of a Multi-tenant database system , this is called scalability database system .Scalability is defined as the system ability to handle growing amounts of work in a graceful manner [28]. In our experiments, we consider the scalability of SaTbencHCloud approach by measuring system throughput as database scale increases. Two sets of experiments are evaluated in terms of different dimensions of data scale: tenant amounts and number of columns in the shared table. We using the original shared table as the baseline in the experiments.

5.1 Experimental Settings and Results

In this section we will present the experimental settings and results to supporting Multi-tenancy schema inheritance in RDBMS for SaaS and make a comparison with other techniques. In general there are two types of tests: the load test and the performance test. The load test involves loading the database with data and running the queries. The latter involves measuring the system's performance against a specific workload. We will customize the tests and discuss the exact steps that need to be taken and the values to be measured. We first present settings for benchmark databases generation . Two sets of experiments are examined to evaluate the scalability of the Multi-tenant system, we considering the throughput and response time in relation to the number of tenants and the effect of column amounts.

5.1.1 Generating Dataset

Our experiments require database configuration and data generation , it consists of three phases:

creating the tables in the database, populating the data, and finally loading the populated data in the DBMS. To meet the experience requirements, our dataset must contain different sizes with 10GB , 100GB , and 300GB for 100 , 500, and 1,000 tenants respectively. Table 5-1 shows these settings.

Table 5-1 : Database Configuration Settings

Database Name	Size in GB	# of Tenants
SaTbencHCloud_10GB	10	100
SaTbencHCloud_100GB	100	500
SaTbencHCloud_300GB	300	1,000

After the process of generating the data into the three databases, each queries was run against these databases set to their default settings. We recording the performance of query results with the response time.

The performance measurements were done using “Benchmark Factory” tool which was used to trace all the SQL events that were taking place on the server. The performance measurements of interest were:

- The Average query response (in seconds) .
- The Average CPU time (in millisecond) .
- The Average Disk Read which provides the average number of reads per second of data from the disk.
- The Average Disk write provides the average number of writes per second of data to the disk.

Since SQL queries is configured to execute by default settings ,we recording the results to compare the results obtained with our approach. Figure 5-1 shows snapshot of some results of the implementation on IBM Bluemix platform on my account.

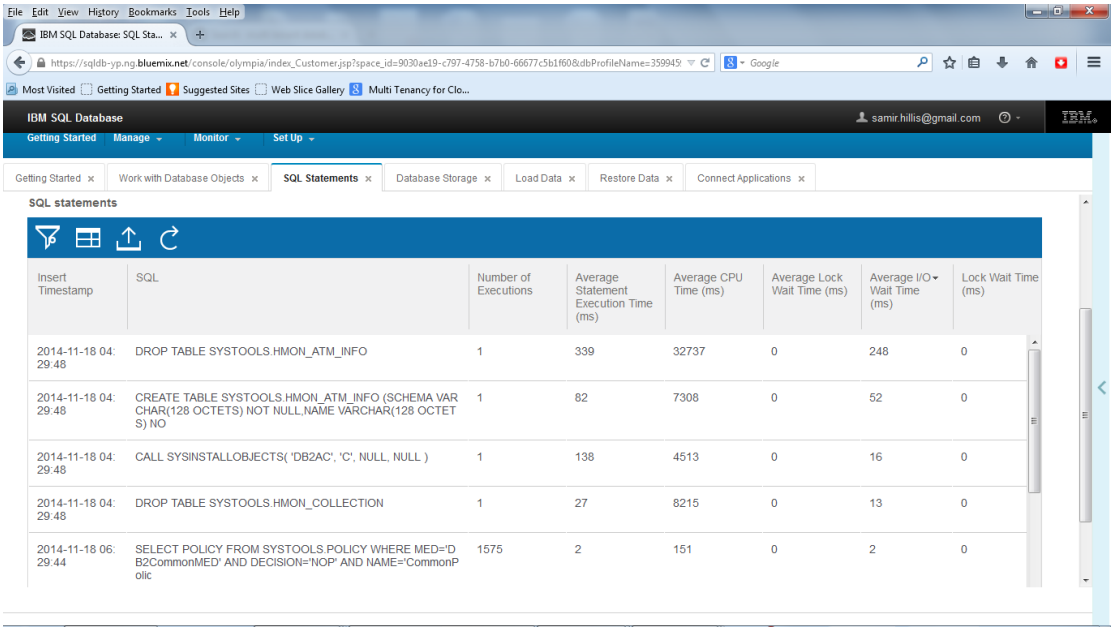


Figure 5-1: snapshot results of the implementation on IBM Bluemix platform

5.1.2 Experimental Environment and Tools

Our experiments are conducted with a desktop pc , running windows 7 operating system with Single Intel® Xeon® Processor E5-2620 v2 (15M Cache, 2.1 GHz) . Number of Cores: 6 cores with 16 GB memory RAM . We evaluate two kinds of Multi-tenant database systems. One is shared table , and the other is SaTbencHCloud Approch. The comparison must be under the same database server, for this reason we implement our experiment by using Oracle Database 12c and oracle TopLink 12c . Shared-table Multi-tenant can be enabled declaratively using the @Multitenant annotation by include it with an @Entity or @MappedSuperclass, or in an Object Relational Mapping (ORM) XML file using the <multitenant> element. We also use a free tool for benchmark database, called benchmark factory .

```
@Entity                          -- @MappedSuperclass
@Table(name="CUST")
@Multitenant(SINGLE_TABLE)
public class CUSTEMER {
}
```

5.1.3 Metadata Management

The Shared Table approach allows consolidating a large number of tenants onto one database instance. The number of tenants is not limited by the available main memory because the size of the data dictionary remains constant if a new tenant is created.

In Multi-tenant Database we exploiting the fact that the data dictionary stores two different kinds of meta data: logical and physical . The logical meta data expresses the structure of the relational model, i. e. by tables, attributes and constraints. The physical meta data describes how data is stored on disk and how to access to this. In shared table, we argue that the contents of the data dictionary look closely alike between tenants, especially with respect to the logical meta data.

5.2 Effect of Tenants

In this section, we present and evaluate the experimental results of SaTbencHCloud approach and Shared table under different tenant amounts. SaTbencHCloud approach implement schema inheritance that allows deriving a schema from another schema. Thereby, a derived schema inherits the objects that are defined in the parent schema. it allows extending and creating objects according to a defined set of rules. Therefore, it defines three different schema types: shared schema, virtual schema and tenant schema.

- **Shared Schema**

The hierarchically schema describes a virtual schemas where a core application may be customized based on Individual tenants needs . because a virtual schema is without table instances. Consequently, it is impossible to store data using a virtual schema. Virtual schema inherits all the database objects contained in the parent schema. The relation between schemas are hierarchically. thereby, a virtual schema can inherit from another virtual schema or more , this means that it inherits all the database objects contained in the parent schema.

- **Virtual Schema**

The hierarchically schema describes a virtual schemas where a core application may be customized based on Individual tenants needs . because a virtual schema is without table instances. Consequently, it is impossible to store data using a virtual schema.

- **Tenant Schema**

Tenant schema relates to a specific tenant. Each tenant possesses an associated tenant schema that represents a part of its context. A tenant schema must inherit from a virtual schema. A tenant schema includes table instances and a tenant schema is final with respect to inheritance. That is, another schema cannot inherit from a tenant schema.

We evaluate the scalability for both systems, "SaTbencHCloud approach and Shared table" by measuring the ability to serve an increasing number of tenants without too much query performance degradation and the column number of tables of database can handle . We examine the system scalability in terms of two aspects: the performance of storage and system throughput.

Figure 5-2 illustrates a simple example for e-commerce was used in [24] . The upper part of the figure models the virtual schema Shop that defines a table Item. The table Item has two attributes Name and Price. The lower part of the figure illustrates two derived tenant schemas. The tenant schema Kermit Shoes in the left and the schema Gonzo Books in the right. The schema Kermit Shoes extends the table Item by an attribute color, whereas the schema Gonzo Books extends it by an attribute pages and another attribute ISBN. In addition, the tenant schemas contain the respective instances of the table Item.

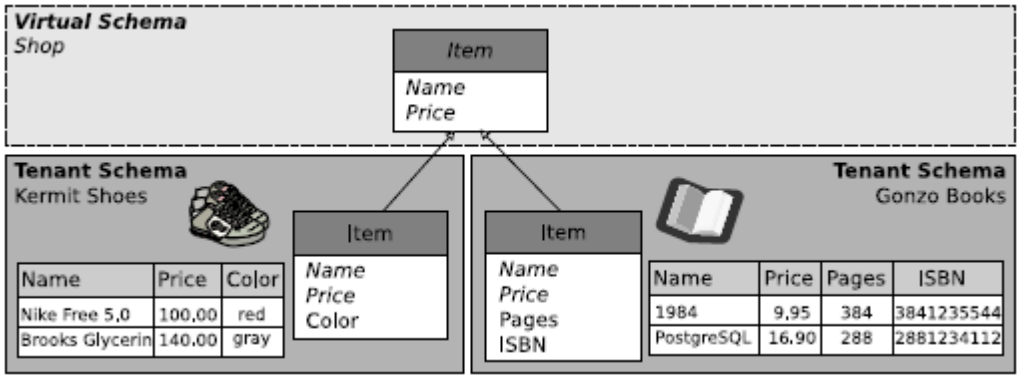


Figure 5-2: Illustration of schema inheritance concept (24).

5.2.1 Storage Capability

We compare the disk space usage of shared table and SaTbencHCloud under different tenant amounts as shown in figure 5-3. It can be clearly seen that SaTbencHCloud outperforms STSI in terms of storage requirement in all the experiments to store the same number of records. it uses an average of about 70% storage space compared with the STSI. Our interpretation of this that shared table consumes large disk space to store null values. On the other hand, SaTbencHCloud extract a data dictionary associated with a tenant from the overall data dictionary and exploitation some situations of data needs to be shared between tenants , rather than migrating data from tenant to another that requires storage consuming and may cause data duplication .

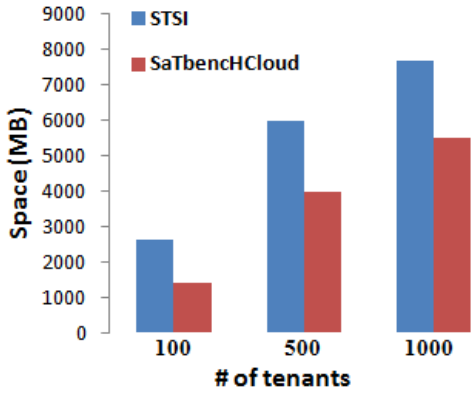


Figure 5.3 : Disk space usage with different number of tenants

5.5.2 Throughput Test

A throughput test is using to measure the ability of the system to process the most queries in the least amount of time. We now investigate the performance of SaTbencHCloud approach and STSI on concurrent operations. The throughput test must be executed under the same conditions for both approaches. The Driver runs all queries and the Multi-tenant database system in a “client/server” configuration to simulate a real Multi-tenant environment. all the processes are executed in parallel against indexed attributes. To ensure the accuracy of the results, we execute TPC-H queries workload with its default settings and compare it with SaTbencHCloud approach result. We discuss the usability of our approach.

Data manipulation language (DML) Performance

Based on our proposal we divide DML operations into three categories, The first is the DML from original database schema. The second is DML when the tenant add new

columns . The third is the DML when the tenant add new tables. For each workload we repeat the experiments five times and obtain the average time. As show in figure 5.4 we compare the operation costs among STSI and SaTbencHCloud approach according to the example which was explained in paragraph 4.2.1.3 The experiment will perform on the three databases with workloads of scale factor 10, 100, and 300 Gigabytes. We use driver to run the queries for each tenant account. We call the selection operations: sel1, sel2 and sel3 and call the insert operations: ins1, ins2 and ins3 respectively for short . Similarly with the deletion and update.

When scale factor (SF) = 10 , we set the number of tenants = 100, compared with STSI we see that sel1 and sel2 have much better performance than STSI . For sel3 performance will decline but it remains the best of the STSI since the costly join operation that required create virtual database relationship between physical tables and virtual table . Results of the experiment are shown in figure 5-4.

When scale factor (SF) = 100 , we set the number of tenants = 500 and when scale factor (SF) = 300 we set the number of tenants = 1000. , we can see that the performance of SaTbencHCloud approach is remains slower than SF = 10, but it is outperforming STSI in terms of system throughput. We can conclude that SaTbencHCloud approach is not affected by increasing the number of tenants.

Our interpretation of the efficiency of SaTbencHCloud approach uses fewer disk I/Os to fetch the records of DML operations to memory than STSI because it displays the data for the one tenant only at a moment. On the other hand, Index pivot table associated with a specific tenant improve and speed up the query execution time when retrieve data , The index is built on the tenant's identity column . In contrast STSI use a big indexes records from all tenants. The lookup becomes inefficient with large number of tenants.

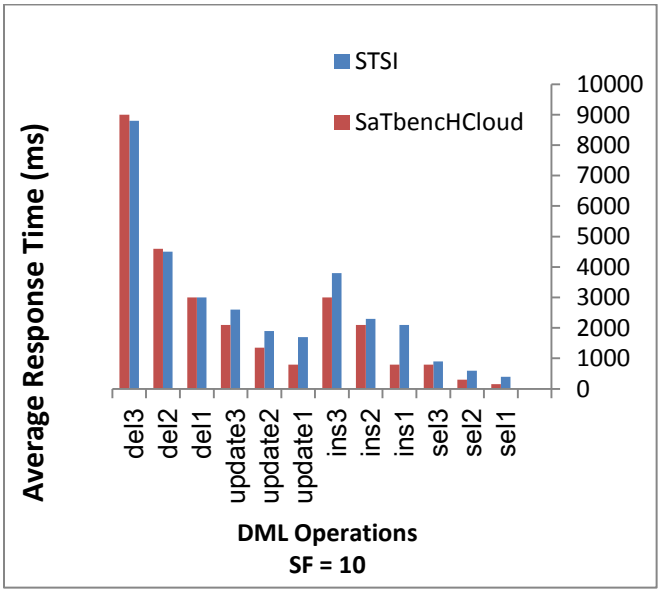


Figure 5.4: DML Performance when scale factor = 10

For the insert operations , STSI will need to insert many NULLs in all columns in the shared table even if the tenant was not need of any one to maintaining the big index (BI) . However, we can show that SaTbencHCloud throughput is about twice as high as STSI when SF = 10 , but it goes down to 30% when SF = 100, show figure 5.5. Finally, when SF=300 SaTbencHCloud performance remains is better than STSI more than 20%. With an increasing number of tenants in the system requires more I/O , and joins operations, as a result the throughput slightly reduces. Compared to STSI, SaTbencHCloud is more efficient in performing table scan and index lookup.

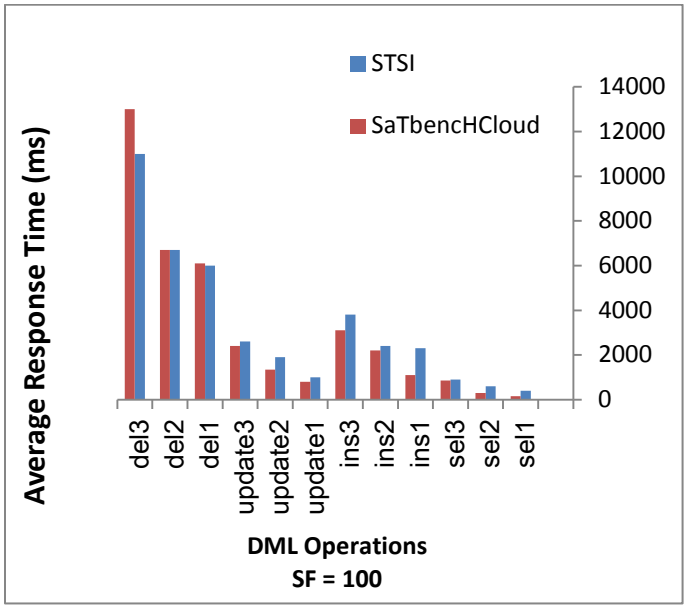


Figure 5.5: DML Performance when scale factor = 100

Update operations followed the same behavior as selection and insertion. It was clear that the performance of SaTbencHCloud better than the performance STST.

In violation of all experiments the performance of our approach fell in delete operations , since STSI is the best when scale factor (SF) = 100 and when scale factor (SF) = 300 by Percentage between 10% and 15% . We believe that the reason for that is under referential integrity that requires to check the records before deleted , this procedure will cost some time. Figure 5.6 show the results of the experiment DML Performance when scale factor = 300.

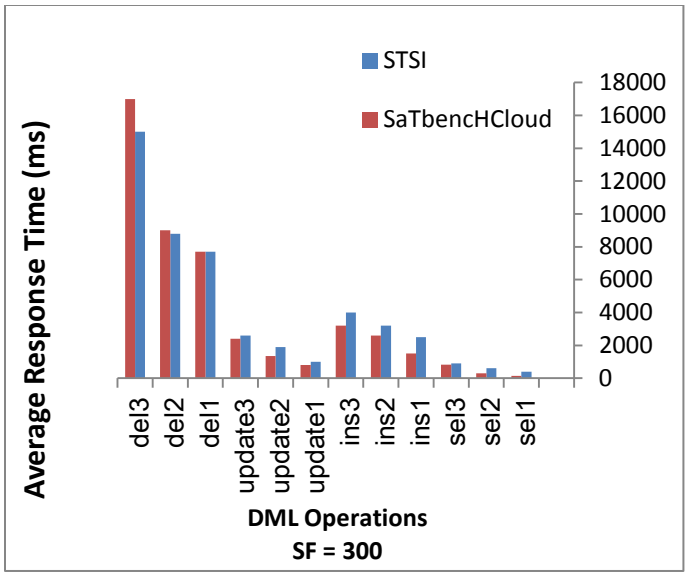


Figure 5.6: DML Performance when scale factor = 300

5.6 Effect of Columns

Database as a service is designed to support a large number of tenants and each of them have different requirements , but a few of columns are common , for this reason we need to handle the situation that the base schema is very sparse and contains a large amount of configurable columns owned by different tenants. One of the big challenges in the shared table model is decide the number of custom fields (columns in table) for tenants , Providing less number of columns might restrict the ability tenants who wish to use a Multi-tenant database systems and flexibility of extend the table. We investigate the scalability of SaTbencHCloud approach vs STSI with an increasing number of columns and the impact on the efficiency of the system performance and the use of suitable storage space.

5.6.1 Storage Capability

In this experiment, we will examine storage capability for each of SaTbencHCloud approach and STSI with the increasing number of columns. We assume that the number of added columns in the shared table varies from 10 ,100 ,300 in our three different databases respectively . Figure 5.7 illustrates the disk space usage of SaTbencHCloud approach and STSI. The figure shows that SaTbencHCloud approach requires less storage space compared with STSI about a percentage 50% .

Our interpretation that SaTbencHCloud approach operates according to the idea of tenant context , this means that there is a degree of integration between Multi-tenant relational tables and virtual relational tables mean that storage data is associated with a particular tenant according to the columns defined by this user without leading to store any values

of other tenants which shows a good scalability in respect to the system storage . This concept already applied in the column-oriented databases. Also any schema modifications of one tenant will not affect the logical schema of other tenants.

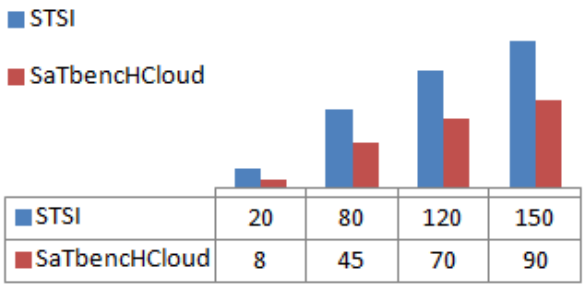


Figure 5.7: Disk space usage with different number of columns

5.6.2 Throughput Test

Our objective now is to evaluate the effect of Increase columns on the system throughput. We will test the three databases under different workloads. We will use QGEN to generate executable query, then we will execute the same queries against these databases After the extension of the table by adding new columns and the response of the two approaches with the process . Figure 5.8 displays the system throughput and response time for SaTbencHCloud approach and STSI. As is clear, there is a decline in the performance of two approaches when increasing the number of columns, but it does not affect the scalability.

It is clear that SaTbencHCloud approach offers the best performance because it has the ability to selectively I/O in columns to improve the performance.

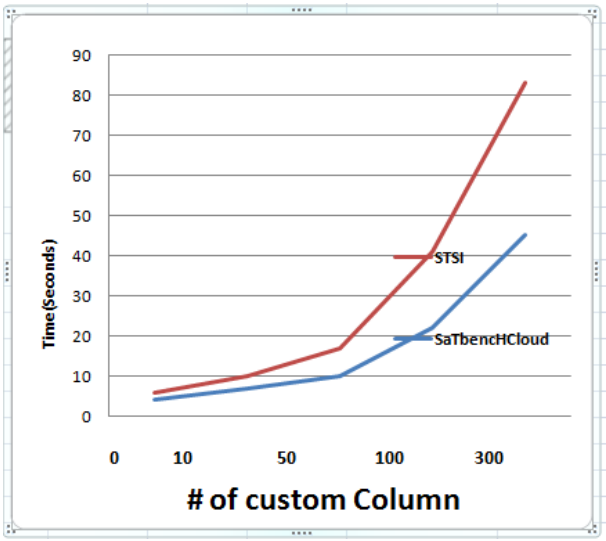


Figure 5-8 : System throughput and response time for SaTbencHCloud approach and STSI

Summary

In this chapter, different types of experiments have been constructed to evaluate the efficiency and scalability in the cloud database. We performed these experiments using different sizes of small, medium, large and very large databases, our goal is to measure the impact of an increasing number of sessions on the reactivity of the system while increasing the number of tables and tenants. These experiments covered most of possible situations, which we considered as an important contribution in this thesis.

Three groups of schemas for 100, 500, 1,000 tenants have generated . These schemas are then used for evaluating the scalability of storage and query processing under different schema variability to observe significant differences in query response between these three different scale factors. We using the original shared table as the baseline in the experiments. We noted the results for each experiment in order to evaluate and discuss these results .

In evaluation section, we evaluated our approach in terms of performance, scalability, flexibility and efficiency by measuring system throughput as data scale increases. we considering the throughput and response time in relation to the amount of tenants and the effect of column amounts. We have provided our interpretation of the results of each experiment.

Chapter 6

Conclusion and Future Work

In this chapter, we concluded our work, results, and the future work directions. We discussed three approaches for designing a Multi-tenant database architecture . It is clearly that the shared table approach is a most popular today among hosting service providers, so we studied this approach and focused on the problem of inability to customize and enable extension dynamically by Schema-Mapping Techniques when the system is on-line without affect the logical schemas of other tenants .

Many researchers discussed extension tables problem, and introduced some solutions. Most prior works cared allowing the tenant to have his own private tables, which can be extended and changed. None of them discussed to take advantage of the shared data ; also, they did not provide any contribution in the metadata management. One of the major advantages of our approach that uses two related works, they are pivot table and universal table.

The idea of our solution was proposed SaTbencHCloud , that it is an efficient approach for supporting Multi-tenant schema inheritance in RDBMS for SaaS tailored to Multi-tenancy. We offers different schema types for different situations. we focused on meta data management to overcome the null values, and bring the data by the tenant's identity, as well as building tenant indexes. Our experiments results show that our approach decreases main memory consumption and lookup times of the data dictionary compared to STSI .

To measure the performance of the database we used TPC-H benchmark that are widely used in industry and academia to measure performance characteristics of database systems. In order to enhance the benchmark to suits with our work, we introduced simple modifications but important on some other related work. One of advantages of our SaTbencHCloud that is suitable with small, medium, large, and very large databases , and it can be used with any generic relational database schema and SQL queries, It is also give results that are highly comparable with other benchmarks.

Our approach has proved its efficiency under different tenant numbers and in storage requirement in all the experiments to store the same number of tuples.

SaTbencHCloud approach feet higher throughput when uses fewer disk I/Os to fetch the records of DML operations to memory than STSI because it displays the data for the one tenant only at a moment. On the other hand, Index pivot table associated with a specific tenant improve and speed up the query execution time when retrieve data , The index is built on the tenant's identity column . In contrast STSI use a big indexes records from all tenants.

Experience has shown that SaTbencHCloud outperforms STSI in terms of storage requirement under different tenant amounts . it uses an average of about 70% storage space since it get rid of the problem of storage of null values, and requires less storage space compared with STSI about a percentage 50% in terms of effect of columns.

The throughput of system testing proved that SaTbencHCloud throughput is about twice as high as STSI for select operation in the case of small database. It is also best at about 30% in the case of medium database, and about 20% for big database. Update and insert operations followed the same behavior as selection . It was clear that the performance of SaTbencHCloud better than the performance STST.

In violation of all experiments the performance of our approach fell in delete operations , since STSI is the best when SF = 100 and when SF = 300 by Percentage between 10% and 15% . We believe that the reason for that is under referential integrity that requires to check the records before deleted , this procedure will cost some time.

Evaluate the effect of Increase columns on the system throughput shows that there is a decline in the performance of two approaches when increasing the number of columns, but it does not affect the scalability.

In our future work, we intend to complete and efficient support for Multi-tenancy , and to facilitate the migration of applications feature between cloud database services providers according to security requirements.

Some of most important trends of the future work are listed below:

- Query optimizer.
- Cloud management.
- Techniques of Import historical data to the cloud server.
- Supporting BigData and unstructured data.
- Migration between cloud service providers and security issues associated.

References

[1] "The NIST Definition of Cloud Computing" . National Institute of Standards and Technology. September 2011.

[2] 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 OSDI, 2006.

[3] Hyun Jin Moon, Carlo Curino, and Carlo Zaniolo. "Scalable Architecture and Query Optimization for Transaction-Time DBs with Evolving Schemas". In Elmagarmid and Agrawal (2010), pages 207‑218. ISBN 978-1-4503-0032-2.

[4] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. "Multi-tenant databases for software as a service: schema mapping techniques". In SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1195–1206, New York, NY, USA, 2008. ACM.

[5] Mei Hui, Dawei Jiang, Guoliang Li, Yuan Zhou, "Supporting Database Applications As A Service". IEEE International Conference on Data Engineering, 2009.

[6] Craig D. Weissman and Steve Bobrowski. the Design of the force.com "Multitenant Internet Application Development Platform" . In Cetintemel et al. (2009), pages 889‑896. ISBN 978-1-60558-551-2.

[7] D. Jacobs and S. Aulbach. "Ruminations on multi-tenant databases". In A. Kemper, H. Schoning, T. Rose, M. Jarke, T. Seidl, C. Quix, and C. Brochhaus, editors, BTW, volume 103 of LNI, pages 514–521. GI, 2007.

[8] Curino, C., Jones, E., Popa, R., Malviya, N., Wu, E., Madden, S., Balakrishnan, H.,Zeldovich, N. 2011. "Relational Cloud: A Database Service for the Cloud" . In CIDR, pages 235–240.

[9] Stefan Aulbach, Michael Seibold, Dean Jacobs, and Alfons Kemper. "Extensibility and Data Sharing in Evolving Multi-tenant Databases". In Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE), pages 99‑110, 2011.

[10] Franclin S. Foping, Ioannis M. Dokas, John Feehan and Syed Imran "A New Hybrid Schema-Sharing Technique for Multitenant Applications" , IEEE – Digital Information Management 2019, Cork Constraint Computation Centre University College Cork Ireland 1-4 Nov. 2009

[11] Carlo Curino, Hyun Jin Moon, and Carlo Zaniolo. "Automating Database Schema Evolution in Information System Upgrades". In Tudor Dumitras, Iulian Neamtiu, and EliTilevich, editors, HotSWUp. ACM, 2009. ISBN 978-1-60558-723-3.

[12] D. Jacobs. "Enterprise software as service". ACM Queue, 6(3):36–42 .

[13] Z. H.Wang, C. J. Guo, B. Gao,W. Sun, Z. Zhang, and W. H. An. "A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing".In e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on, pages 94–101, Oct. 2008.

[14] R. Elmasri and S. B. Navathe. "Fundamentals of Database Systems", 5th Edition. Addison-Wesley, 2007.

[15] http://cloudcomputing.sys-con.com/node/1610582, (last visited 17-04-2014)

[16] Mateljan, V., Cisic, D., Ogrizovic, D.: "Cloud Database-as-a-Service (DaaS) " ROI. In MIPRO, Proceedings of the 33rd International Convention, 1185—1188 (2010).

[17] F. Chong and G. Carraro, "Architecture Strategies for Catching the Long Tail," Microsoft Corporation, http://msdn.microsoft.com/en-us/library/aa479069.aspx, Tech. Rep., April 2006, (last visited 09-05-2014).

[18] Yaish, H., Goyal, M., Feuerlicht, G.: "An Elastic Multi-tenant Database Schema for Software as a Service". In the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing, 737—743 (2011).

[19] Bezemer, C., Zaidman, A., Platzbeecker, B., & Hart, A. (2010). "Enabling Multi-Tenancy : An Industrial Experience Report". Innovation, 1-8. IEEE. doi:10.1109/ICSM. 2010.5609735

[20] Indu Arora and Anu Gupta . "Cloud Databases: A Paradigm Shift in Databases " , IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 4, No 3, July 2012

[21] Vladislav Lazarov. "Comparison of Different Implementations of Multi-Tenant Databases" . Bachelor's Thesis, Chair for Database Systems, Department of Computer Science, Technique Universality Munchen, Germany, July 2007.

[22] http://cloudscaling.com/resources , (last visited 17-04-2014).

[23] Carlo Curino , Evan P. C. Jones, Raluca Ada Popa, Nirmesh Malviya "Relational Cloud: A Database-as-a-Service for the Cloud." 5th Biennial Conference on Innovative Data Systems Research, CIDR 2011, January 9-12, 2011 Asilomar,California.

[24] Oliver Schiller, Benjamin Schiller, Andreas Brodt: "Native Support of Multi-tenancy in RDBMS for Software as a Service" Proceedings of the 14th International Conference on Extending Database ACM New York, NY, USA ,2011

[25] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao." A Framework for Native Multi-Tenancy Application Development and Management". In E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on, pages 551 –558, July 2007.

[26] http://www.elementsolutions.com/2013/08/08/part-2-cloud-computing-demystified-3s-4d-5e-is-all-you-need-to-know, (last visited 11-05-2014)

[27] Zhang, Q., Cheng, L., & Boutaba, R. 2010, "Cloud computing: State-Of-The-Art and Research Challenges". Journal of Internet Services and Applications, vol. 1, no. 1, pp.7-8.

[28]. Burgess G, What is the TPC Good For? Or, the Top Reasons in Favour of TPC Benchmarks, http://www.tpc.org/information/other/articles/TopTen.asp, (last visited 17-03-2015)

[29]. Watts, D., Slavko, B., Watson, C., Understanding IBM eServer xSeries Benchmarks, http://www.redbooks.ibm.com/redpapers/pdfs/redp3957.pdf, (last visited 20-03-2015)

[30] Andr´e B. Bondi. Characteristics of scalability and their impact on performance. In WOSP '00: Proceedings of the 2nd international workshop on Software and performance, pages 195–203, New York, NY, USA, 2000. ACM.

[31]. Scalzo B., Ault M., Burleson D., Fernandez C., Klein K., Database Benchmarking, Practical Methods for Oracle & SQL Server, Rampant TechPress, USA, April 2007

[32] Microsoft SQL Server 2014, http://www.microsoft.com/en-us/server-cloud/products/sql-server/ (last visited 20-03-2015)

[33] TPC: Transaction Processing Performance Council , http://www.tpc.org/ (last visited 23-03-2015)

[34] TPoX: Transaction Processing over XML (TPoX) (2012) , http://tpox.sourceforge.net/ (last visited 23-03-2015)

[35] Tpc-c. http://www.tpc.org/tpcc/. (last visited 23-03-2015)

[36] Tpc-h. http://www.tpc.org/tpch/default.asp/. (last visited 23-03-2015)

[37] Foping, F.S., Dokas, I.M., Feehan, J. & Imran, S. 2009, 'A new hybrid schema sharing technique for multitenant applications', Digital Information Management, 2009. ICDIM 2009. Fourth International Conference on, IEEE, pp. 210-215.

[38] Kwok, T., Thao, N. & Linh, L. 2008, 'A Software as a Service with multi-tenancy support for an electronic contract management application', *Services Computing, 2008. SCC '0*8. IEEE International Conference on, IEEE, vol. 2, pp. 179-186.

[39] Jiyi Wu et al, "Recent Advances in Cloud Storage", in Third International Symposium on Computer Science and Computational Technology(ISCSCT '10), Jiaozuo, P. R. China, 14-15,August 2010, pp. 151-154.

[40] https://msdn.microsoft.com/en-us/library/azure/ee336245.aspx (last visited 05-05-2015)

[41] D. Jia, W. Hao-yu, and Y. Zhao-jun, "Research on data layer structure of multi-tenant e-commerce system", IE&EM, 2010, pp. 362 – 365.

[42] Mietzner, R., Metzger, A., Leymann, F. & Pohl, K. 2009b, 'Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications', *Principles of Engineering Service Oriented Systems, 2009. PESOS 2009. ICSE Workshop on*, IEEE, Vancouver, Canada, pp. 18-25.

[43] Bezemer, C.P., Zaidman, A., Platzbeecker, B., Hurkmans, T. & t Hart, A. 2010, 'Enabling multi-tenancy: An industrial experience report', *Software Maintenance (ICSM), 2010 IEEE International Conference on*, IEEE, Timisoara, Romania, pp. 1-8.

[44] Stefan Aulbach ,Dean Jacobs$ Alfons Kemper , Michael Seibold , "A Comparison of Flexible Schemas for Software as a Service".