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

Tawfiq S. Barhoom[1], *Samir A. Hillis*[2]
[1] *Islamic University of Gaza , tbarhoom@iugaza.edu.ps*
[2]*Islamic University of Gaza, samir.hillis@gmail.com*

Abstract— 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). 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 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 poor performance and high space overhead. In this paper, we propose an efficient approach for supporting multi-tenancy schema inheritance. We trade-off STSI and our approach. 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 DML operations.

Index Terms— *cloud computing, Database as a Service (DBaaS),Multi-Tenant database , schema-mapping technique.*

I 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 decreased 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 on 24/7/365 basis.

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.

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 [16]. 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] which runs from the same code base, and can thus be maintained centrally [6] .

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 [18].

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]*

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 the same way, a single database is shared amongst customers to store all tenants' data: this is known as "multi-tenant database".

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 tenants (application instances) can be representations of organizations that obtained access to the multitenant. 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.

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] .

II 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, 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 [15]. 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. there are three Approaches to Managing Multi-Tenant databases as shown in Figure 1: 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.

We review existing multi-tenant database schema design methods

(a) **Separate Database:** In this approach, a separate database is assigned to each tenant for data storage. Each database contains some metadata used to redirect each tenant to the correct database. This approach is considered expensive in both implementation and maintenance.

(b) **Independent Tables and Shared Instances:** 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 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. Selecting the appropriate approach depends on different criteria. 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].

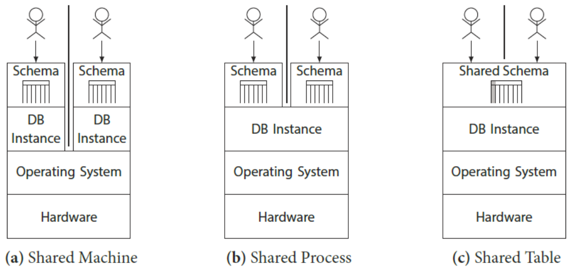


Figure 1: Types of Multi-Tenant data storage systems [22]

III 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 [7,13] .
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].

IV Related Works

Recently, cloud computing became a dominant field in the information technology world. It prevails over both academia and industry. many studies have been done by companies and researchers to supporting outsourcing database as a service, and extending relational DBMS. Cloud Service Providers (CSP) provide many services such as storage, platform and applications.
Companies like force.com does its own mapping from logical tenant schemas to one universal physical database schema (Weissman & Bobrowski) to overcome the limitations of traditional DBMSs. However, this approach complicates development of the application because of 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 uninterpreted 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.
Bezemer, et al.[17] 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.
Curino et al. and 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].
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 multitenant tables and joined as needed.
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 [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.
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[19] Schiller, et al. proposes 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.
Jiacai Ni, et al. [22] build the physical tables from the attribute level instead of the tenant level by extract the highly important attributes from the tenants and build several base tables using such important attributes and propose an adaptive method to build base tables and supplementary tables based on database schemas of different tenants and query workloads.

V Contributions

We used TPC-H schema [21]. The schema comprises 8 tables. database generator will be used it use to populate the database with data.

- We propose a new Virtual Schema that inherit both shared data and metadata from Shared Schema. Thereby, it allows extending tables and creating objects according to parent schema of a multi-tenant database system based on the standard RDBMS.
- We enhance TPC-H benchmark to suit cloud computing, we called it SaTbencHCloud.
- We contribute a tenant data dictionary that allows integration with multi-tenant relational database.

Middleware for Table/ Metadata Sharing

schema inheritance concept: Schema inheritance 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: Multi-tenant applications use tables to store data that is specific to the application and invariant between tenants. In such a case, the tenants only read the table while the provider or an appropriate application maintains its contents.

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 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. Another schema cannot inherit from a tenant schema.

Tenant Context is associated with a specific tenant and keeps all information that allows determining the tenant's virtual database. In other words, the concept of a tenant context determines the tenant's view of the database by isolate a tenant from other tenants.

VI Experimental

This section describes the information needed to empirically evaluate the efficiency and scalability of the SaTbencHCloud. Scalability is defined as the system ability to handle growing amounts of work in a graceful manner [20]. In our experiments, we consider the scalability of SaTbencHCloud by measuring system throughput as data 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 The Multi-Tenant Databases Benchmark.

Benchmarking a database is the process of performing well defined tests on that particular database for the purpose of evaluating its performance. 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 [21]. Today the most important of these benchmarks is TPC-H. In order to enhance the benchmark to suits with our work, we introduced simple modifications but important on some other related work such as that offered by salesforce.com, but they do not consider the extensibility issue of the shared table, which is the heart of our work.

Setting up the SaTbencHCloud

Our version of benchmark called SaTbencHCloud , it focus on a cloud environments with multi-tenancy support. SaTbencHCloud comprises four modules as shown in Figure 2 a shows configurable database base schema, 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.

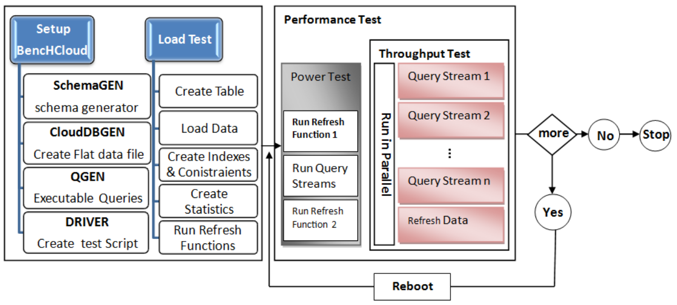


Figure 2: Complete process for running the SaTbencHCloud

SchemaGEN

TPC-H provide a Schema, it should work with most database using only minor modifications. We add 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 Multitenant architecture and designed for the cloud. We use the schema generator called SchemaGEN to produce the schema for each tenant.

CloudDBGEN

CloudDBGEN is use to populate the database with data, it has a scaling factor that influences the amount of data.

QGEN

QGEN is a utility provided by the TPC to generate executable query text. 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.

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.

Metadata-Driven Architectures

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 multitenant database environment that can execute any business domain database. Figure 3 shows the details of metadata-driven architectures that is very significant for multi-tenant applications.

Table 5.1 brief description about metadata-driven fields.

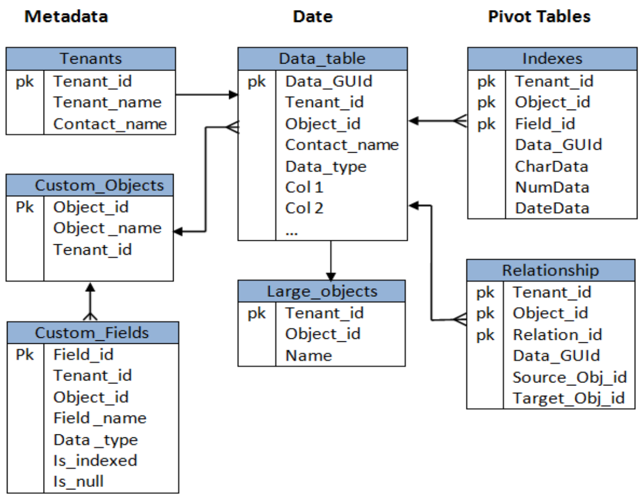

Metadata
Date
Pivot Tables
Tenants: pk Tenant_id, Tenant_name, Contact_name
Data_table: pk Data_GUId, Tenant_id, Object_id, Contact_name, Data_type, Col 1, Col 2, ...
Indexes: pk Tenant_id, pk Object_id, pk Field_id, Data_GUId, CharData, NumData, DateData
Custom_Objects: Pk Object_id, Object _name, Tenant_id
Large_objects: pk Tenant_id, Object_id, Name
Relationship: pk Tenant_id, pk Object_id, pk Relation_id, Data_GUId, Source_Obj_id, Target_Obj_id
Custom_Fields: Pk Field_id, Tenant_id, Object_id, Field _name, Data _type, Is_indexed, Is_null

Figure 3: Metadata-driven Database Design

Integrated TPC-H Schema and multi-tenant relational database

We assumes that the service provider has three tenants. The first tenant was interested to use the original database schema. For simplicity we will use the orders table only as shown in Figure (4-1).

The second tenant found that he needs to use the columns predefined in the order table add new fields to fulfill his business requirements. It including 'Ship Country' and 'Required Date'. Figure (4-2) 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 (4-3).

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, shown in figure 2. 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. Then, we present hardware and software settings. 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 amount of tenants and the effect of column amounts. First, by running SchemaGEN to 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.

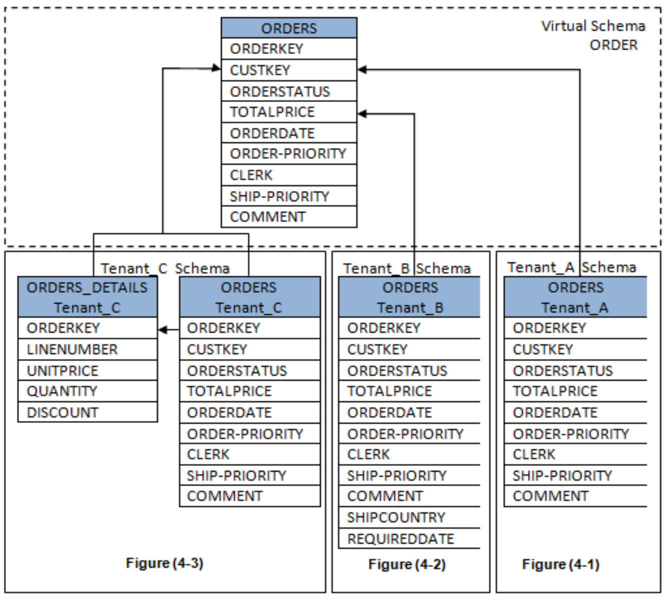

Virtual Schema ORDER
ORDERS: ORDERKEY, CUSTKEY, ORDERSTATUS, TOTALPRICE, ORDERDATE, ORDER-PRIORITY, CLERK, SHIP-PRIORITY, COMMENT
Tenant_C Schema
ORDERS_DETAILS Tenant_C: ORDERKEY, LINENUMBER, UNITPRICE, QUANTITY, DISCOUNT
ORDERS Tenant_C: ORDERKEY, CUSTKEY, ORDERSTATUS, TOTALPRICE, ORDERDATE, ORDER-PRIORITY, CLERK, SHIP-PRIORITY, COMMENT
Tenant_B Schema
ORDERS Tenant_B: ORDERKEY, CUSTKEY, ORDERSTATUS, TOTALPRICE, ORDERDATE, ORDER-PRIORITY, CLERK, SHIP-PRIORITY, COMMENT, SHIPCOUNTRY, REQUIREDDATE
Tenant_A Schema
ORDERS Tenant_A: ORDERKEY, CUSTKEY, ORDERSTATUS, TOTALPRICE, ORDERDATE, ORDER-PRIORITY, CLERK, SHIP-PRIORITY, COMMENT
Figure (4-3)
Figure (4-2)
Figure (4-1)

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

Next, we running CloudDBGEN to generate data for three different databases named SaT_10GB, SaT_100GB and SaT_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.

Effect of Tenants

In this section, we present and evaluate the experimental results of SaTbencHCloud and STSI under different tenant amounts. SaTbencHCloud 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.

Storage Capability

We compare the disk space usage of shared table and SaTbencHCloud under different tenant amounts as shown in figure 5. It can be clearly seen that SaTbencHCloud outperforms STSI in terms of storage requirement in all the experiments to store the same number of tuples. 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.

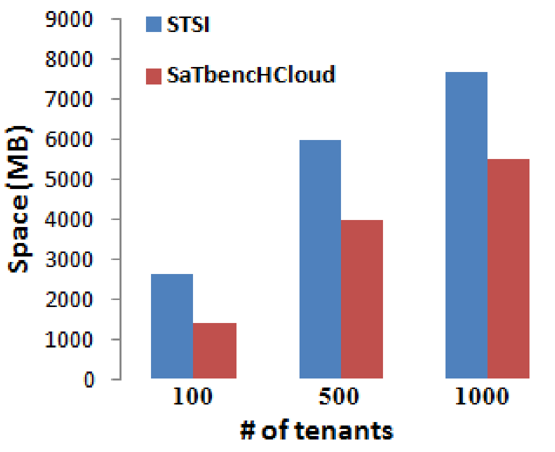


Figure 5: Disk space usage with different number of tenants

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 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 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 shown in figure 4 we compare the operation costs among STSI and SaTbencHCloud according to the example which was explained above. The experiment will perform on the three databases with workloads of scale factor 10, 100 , and 300 Gigabytes. 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 SF = 10 , we assume that the number of tenants = 100, compared with STSI we see that sel1 and sel2 have much better performance than STSI. Show figure 6. 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 . The same thing applies to additions, deletions and updates. When SF = 100 as shown in figure 7 and SF = 300 as shown in figure 8, we can see that the performance of SaTbencHCloud is remains slower than SF = 10, but it is outperforming STSI in terms of system throughput. We can conclude that SaTbencHCloud is not affected by increasing the number of tenants. Our interpretation of the efficiency of SaTbencHCloud 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.

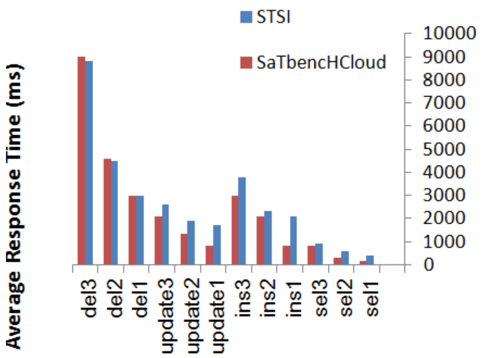


Figure 6: DML Performance when scale factor = 10

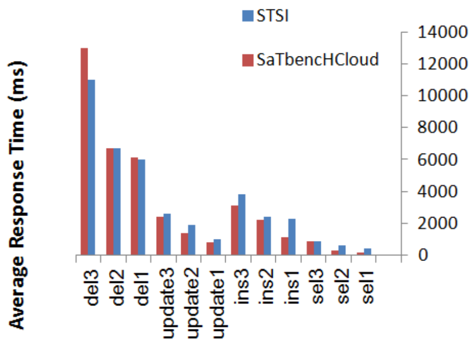


Figure 7: DML Performance when scale factor = 100

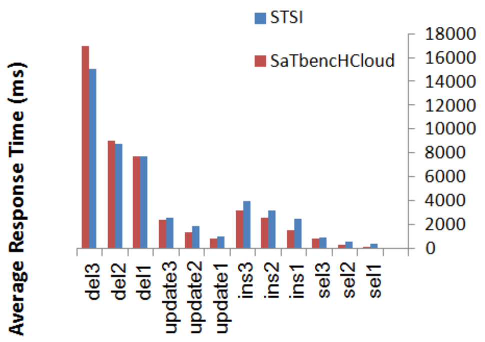


Figure 8: DML Performance when scale factor = 300

Effect of Columns

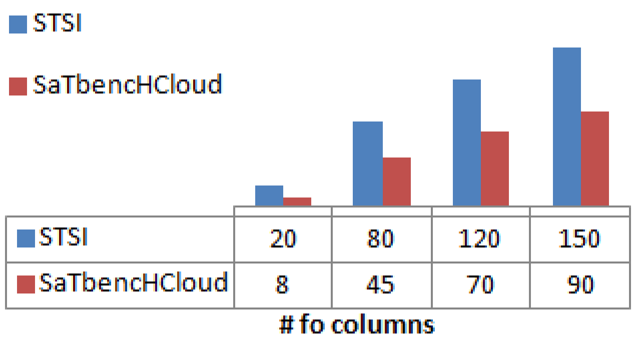
Database as a service is designed to support a large number of tenants and each of them have different requirements, but a few of columns are common, for this reason we need to handle the situation that the base schema is very sparse and contains a large amount of configurable columns owned by different 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 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.

Storage Capability

In this experiment, we will examine storage capability for each of SaTbencHCloud and STSI with the increasing number of columns. We assume that the number of columns in the shared table varies from 10 , 100 and 300 in our three different databases respectively. Figure 9 illustrates the disk space usage of SaTbencHCloud and STSI. The figure shows that SaTbencHCloud requires less storage space compared with STSI. Our interpretation that SaTbencHCloud 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.

Throughput Test

Our objective now is to evaluate the effect of Increase columns on the system throughputs. We will test the three databases under different workloads. We will use QGEN to generate executable query, then we will execute the same



🟦 STSI	20	80	120	150
🟥 SaTbencHCloud	8	45	70	90

Figure 9: Disk space usage with different number of columns

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 10 displays the system throughput and response time for SaTbencHCloud 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 offers the best performance because it has the ability to selectively I/O in columns to improve the performance. The rate of improvement of 30%.

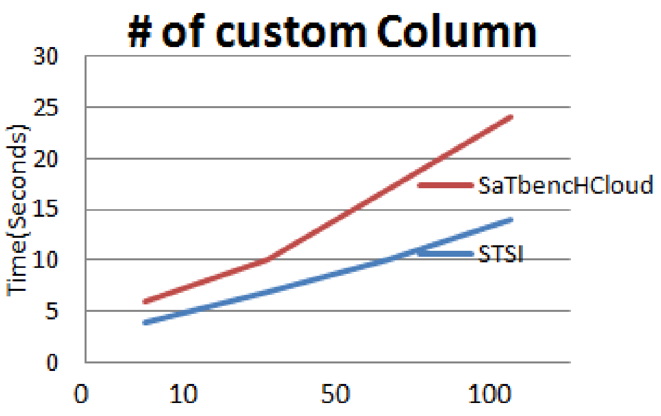


Figure 10: Throughput Test

VIII CONCLUSION AND FUTURE WORK

In this paper we have proposed SaTbencHCloud , that it is an efficient approach for supporting multi-tenancy 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.
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.

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

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20]. 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) .

[21] TPC: Transaction Processing Performance Council , http://www.tpc.org/ (last visited 23-03-2015).
[22] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao.” A Framework for Native Multi-Tenancy Application Development and Management”.

[22] Ni, Jiacai, et al. "Adaptive Database Schema Design for Tenant Data Management." Knowledge and Data Engineering, Transactions on 26.9 (2014): 2079-2093.

Authors Profile
Dr. Tawfiq S. Barhoom received his Ph.D degree from ShangHai Jiao Tong University (SJTU), in 2004. This author is the Dean of IT, Islamic University-Gaza. His current interest research include , Secure Software, Modeling, XMLs security, Web service and its Applications and Information retrieving.

Samir A. Hillis received his BSc in IT from AL-Quds open University at 2000. He works at Palestine Technical College since 2000 as lecturer and now he is the head of registration department. Currently he is MSc candidate in Islamic University of Gaza , his interest in cloud computing, Database management systems, Data Mining and Mobile Applications.