A Hybrid Multi-Tenant Database Schema for Multi-Level Quality of Service

Ahmed I. Saleh
Computers and Systems Department,
Faculty of Engineering,
Mansoura University, Egypt

Mohammed A. Fouad
Information Systems Department,
Faculty of Computer and
Information Sciences,
Mansoura University, Egypt

Mervat Abu-Elkheir
Information Systems Department,
Faculty of Computer and
Information Sciences,
Mansoura University, Egypt

***Abstract*—Software as a Service (SaaS) providers can serve hundreds of thousands of customers using sharable resources to reduce costs. Multi-tenancy architecture allows SaaS providers to run a single application and a database instance, which support multiple tenants with various business needs and priorities. Until now, the database management systems (DBMSs) have not had the notion of multi-tenancy, and they have not been equipped to handle customization or scalability requirements, that are typical in multi-tenant applications. The multi-tenant database performance should adapt to tenants workloads and fit their special requirements. In this paper, we propose a new multi-tenant database schema design approach, that adapts to multi-tenant application requirements, in addition to tenants needs of data security, isolation, queries performance and response time speed. Our proposed methodology provides a trade-off between the performance and the storage space. This proposal caters for the diversity in tenants via defining multi-level quality of service for the different types of tenants, depending on *tenant rate* and *system rate*. The proposal presents a new technique to distribute data in a multi-tenant database horizontally to a set of allotment tables using an *isolation point*, and vertically to a set of extension tables using a *merger point*. Finally, we present a prototype implementation of our method using a real-world case study, showing that the proposed solution can achieve high scalability and increase performance for tenants who need speedy performance and economize storage space for tenants who do not have demanding quality of service.**

***Keywords*—Multi-Tenancy; Flexible Database Schema Design; Data Customization**

I. Introduction

As the majority of small and medium enterprises are pressured to reduce their expenditure in information technology via cutting down costs spent on buying software licenses and updating the hardware. Therefore, a lot of software vendors turn to the principle of sharing hardware resources, software and services over the Internet among a large number of customers, this environment is called cloud computing, and its customers are called tenants. The cloud software delivery model is called Software as a Service (SaaS), and multi-tenancy is the primary characteristic of SaaS [1], as it allows SaaS vendors to run a single instance of an application and a database to serve multiple tenants with various requirements.

Multi-tenancy increases resource utilization, as well as sharing the same database instance to multiple tenants. However, the more the company shares resources, the more risks it faces because an outage of a shared resource can potentially affect many customers. Shared resources also add to the complexity of the solution [2]. The primary multi-tenant application challenge is how to make the application ready for future tenants' requirements, and enable it to fulfill their interests and business needs, without changing code or database schema and without doing too much work.

Managing data in multi-tenancy database can be divided into three major schemas: *Separated Databases Schema*, which is optimum for security, isolation, and customization, but on the other side, it incurs the highest costs and storage space, moreover, it is hard to maintain a large number of databases; *Separated Tables Schema*, whose cost is low as compared to separated databases, and is suitable for small database applications, where the number of tables per tenant is small, but it has scalability issues since it needs to maintain a large numbers of tables; *Shared Tables Schema*, which achieves the best storage space, the lower costs and good scalability at the expense of poor performance. Each of the aforementioned approaches has special requirements in designing schema process, and selecting the appropriate approach for every application depends on a number of changeable factors, such as the nature of application, the number of participative customers, the number of tables and the importance of data security. Table I shows a brief comparison among the three approaches.

When investigating a pool of potential SaaS customers, we found that there are two types of customers: the first segment, is customers who have high workloads and focused on quality of services requirements, such as performance, security and isolation assurance as fundamental requirements; on the other hand, there are some customers, how have low workloads, are focused on minimizing tenancy costs by reducing the hardware resources required in the shared system as much as possible and sacrificing workload performance.

Based on these tenants' requirements, we propose a flexible multi-tenant database schema, using a set of factors and rates to be used as shift keys between a separated databases schema, a separated tables schema or a shared tables schema in a multi-tenancy database, in order to achieve a good scalability and a high performance with a low storage space, while supporting a large number of tenants with different level of performance.

TABLE I. THE MAIN APPROACHES TO MANAGING DATA IN MULTI-TENANT DATABASE COMPARISON

	Separated Database	**Separated Tables**	**Shared Tables**
Data isolation	high	middle	low
Customizability	high	high	low
Scalability	low	middle	high
Maintenance cost	high	high	low
Optimal use of storage space	low	middle	high
General cost	high	middle	low

The proposed multi-tenant database schema satisfy the two segments of customers, and it can be describe as the following:

- Firstly, it is appropriate for tenants who have a high workloads through separating their tables, to confirm the data isolation, high security and reduce joining operation to get the optimal performance in a shared multi-tenant database.
- Secondly, it economizes the costs for tenants who have a low workloads or who do not have demanding quality of service level, through saving storage space, by sharing the same table for these tenants as possible, while preserving a minimum acceptable level of efficiency to get the optimal cost.

The rest of the paper continues with a background review of existing schema mapping techniques in Section II, followed by a full presentation of our multi-tenant database schema technique in Section III. Our case study will be presented in Section IV, and the paper will be concluded in Section V.

II. RELATED WORK

The three major approaches to manage data in multi-tenant databases are summarized in Table I, and discussed in detail in [3] [4]. In addition to, there are several multi-tenant database schema mapping techniques ware presented in [5],[6], the majority of these techniques are derived from the three major approaches of managing data in multi-tenant databases, in order to create a logical isolated database schema for every tenant in multi-tenant databases. We classify multi-tenant database schema mapping techniques in to three segments: techniques using a single approach to represent data in a multi-tenant databases, techniques mixing two or three data isolation approaches, and techniques mixing unstructured data as XML data type and structured data as relation database.

A. *Techniques Deploying a Single Approach*

Some SaaS providers prefer to use a single multi-tenant data storage approach to represent data in multi-tenant databases, to avoid the complexity in database design.

Private Tables' Technique is derived from the separate tables schema, where each tenant has a logical schema consisting of a set of extensions tables. This technique was explicated in [5], and provides a high performance, however neglects the storage space and scalability. It was preferred to use when applications have a few tenants or a few tables [1].

Universal Table' Technique is derived from the shared tables schema, and was referred to as the most flexible technique in [7]. This technique was adopted by SalesForce.com. In the other side, this technique wastes storage space, because of the great use of *null* values, moreover, it harms query performance because it does not support indexes.

Pivot Tables' Technique is derived from the shared tables schema, it was explicated in [1]. This technique eliminates *null* values and supports more flexible extensions at the expense of increasing query processing time for inserting, updating and deleting operations.

The proposed technique in [8], hosted every tenant data in a separated database, and divided tenants' databases into two classes: high performance and low performance. The main important point in this technique is that it measures the tenant workload by transactions per second, and uses this metric to measure overall workloads to get the hardware provisioning policy and the associated scheduling policy.

B. *Techniques Mixing the Separate Tables Schema and the Shared Tables Schema approaches*

There are a lot of techniques ware proposed to representing data in multi-tenant databases such as in [1], [3], [6], [9], [10]. These techniques mixing separate tables and shared tables schemas to achieve balance between scalability, performance, data isolation and storage space with the best cost. However, these techniques achieve some features, at the expense of other features.

M-store' technique was proposed in [3], it saves storage space and prevents null values, at the expense of reconfiguration. Our proposal solves *null* problem relatively to tenant need of performance.

Chunk Table' technique and Chunk Folding' technique ware proposed in [6], these were flexible and reduce the number of tables, at the expense of increasing the queries complexity, because of the huge joining operation. These techniques just focus on vertical partitioning into logical tables 'chunks', however, our proposal adds vertical and horizontal partitioning to save a specific level of performance for every tenant.

An Elastic Schema' technique was proposed in [1], it works on increasing query performance and storing the data in the database as a character large object (CLOB) or binary large object (BLOB) values, to eliminate the impact of BLOB and CLOB values, and divides tables into common tables and virtual extensions. In our proposed technique every attribute has a special *attribute rate* depending on: the type, size in memory and rate of participation between tenants, in order to decide merging or separating this attribute from the base table.

The technique proposed in [9] works on reducing joining operations, by measuring an attribute's importance based on how many tenants share it. The attributes kept in the base table if they have high rate of participation. Unfortunately, this technique ignores the workload for individual tenants, however, in our proposed technique we solve this problem by evaluating the importance of attributes, via collecting the tenants workloads incorporating with the attributes workload.

C. *Techniques Mixing an Unstructured Data and a Structured Data*

In [10], the technique works on splitting up the common content tables, shared by all tenants, away from the extension tables. The extension tables contain additional information, tenants may need to supply, these tables are stored in XML document. Using XML technique satisfies SaaS providers and tenants' needs, because the extension data can be handle without changing original database schema. However, it down of performance in queries mechanism, because the collecting between unstructured data in XML and structured data in relation database takes more time.

III. Flexible Multi-Tenant Database Schema

A. *Flexible Multi-tenant Database Schema Overview*

The previous section outlined the common schema-mapping techniques for managing data in multi-tenancy databases. These techniques focus on realizing all tenants' requirements, and ignore the multiple levels of tenants' workloads. However, in multi-tenancy architecture, a single application and database instance should comply with the tenants' needs as a whole, but the tenants on the same server may have multiple requirements with varying qualities [8], according to a set of business factors, such as: information system workload and importance of data security. These factors motivate us to propose a new schema mapping technique to support multiple levels of data isolation, data security and performance for the tenants in the multi-tenant database.

Multi-tenancy in the database tier can be achieved by sharing databases at different levels of isolation, which results in different multi-tenancy database models according to the requirements for each system. There are three main approaches for isolating data in the multi-tenant database: firstly, shared server and separated databases for each tenant, which provide the highest degree of isolation, but it is much more costly; secondly, shared database and separated tables for each tenant, which is lower than the previous approach in isolation issue, but more fair in the cost; thirdly, shared tables approach which provide the worst degree of data isolation, but it is not costly for the majority of tenants and it achieves a good scalability.

Usually, SaaS providers select the appropriate data isolation approach to each application depending on a set of factors that can change over time, such as: the number of tenants, the size of tenants' data, the importance of data isolation, and the desirable security degree. Unless multi-tenant databases are equipped to handle the changes in these factors such as increasing the number of tenants or the size of their data, they will waste a lot of time and effort in reconstructing database architecture, which is illogical and unacceptable, because SaaS applications should be scalable to support the inconstant customers' needs, without affecting the existing tenants' services.

Building multi-tenant applications with incomplete and inconsistent requirements calls for building a flexible multi-tenant database schema to manage the additional tenants information. Building a flexible multi-tenant database schema should take into account all the influential factors of building the multi-tenant system, including the tenants requirements and the SaaS provider expectations, because multi-tenant database schema should scale to multiple levels of tenants, with multiple requirements and multi-quality of service. This motivates us to propose a new dynamic partitioning mechanism to isolate data in a multi-tenant database, that contains a mixed mode of the three main isolation approaches, in order to improve the server utilization and minimize wasted storage space, while keeping appropriate quality of service for each tenant.

The shared tables schema is referred to as "pure multi-tenancy" in [4] [11]. By the same token, we depend mainly on this approach in our system, in addition to using some metrics to separate extensions tables, to get the desired performance for some tenants, who have a high workload. Finally, we may use a separated database in a special cases, to ensure a desired security level for data, so our proposal combines characteristics of shared tables, separated tables and separated databases architectures.

B. *The Standard Components to Build a Multi-tenant Database*

A flexible multi-tenant database should be based on three components: firstly, *metadata-driven schema architecture*, which allows tenants to add customizable extensions to the common objects or create entirely new customizable objects. The metadata tables save data of each tenant, such as: users data, desirable entities, customized attributes, and reference to the tenants' data in the temporary tables "*generated tables*", generated tables is provisional tables made in order to store the actual data for tenants; secondly, *global unique identifier*, which generates a new unique id for every inserting operation from any user in any table, which will be used to store the data in the generated tables to minimize the volume of metadata columns in the generate tables; thirdly, *runtime table generator* is the most important component, which decides where to save the data in the generated tables or create a new generated table to save the data.

C. *Data Partition in a Multi-tenant Database*

The Data distribution in multi-tenancy databases should based on the current transactions and the expected transactions on the data. Before we decide what is the appropriate data distribution approach, we ought to answer the following questions:

a) What are the main functions for the system, and the required resources?

b) What are the additional tenants' requirements, and the required quality of services for each tenant?

c) What are the required memory and processing power to access various data type?

d) What is the expected workload for every table?

e) What is the critical point to horizontal data isolation?

f) What is the critical point to vertical data merging?

Note that the requirements of the system are split to the functional requirements that are the services the system should provide and the minimum quality of services to be accepted, and the non-functional requirements are the other services that improve the system properties e.g. performance, response time.

The proposed technique realizes scalability not only by supporting a large number of customers, but also supporting multiple levels of qualities, through supporting multiple levels of data isolation. *Runtime table generator* is responsible for distributing the tenants' data in the *generated tables*, according to set of suggested factors: *system rate, tenant rate, attribute rate*, and *table rate*. These factors take in account the volume of activity of the enterprise and the system workload, to provide appropriate data isolation approach and level of performance for the enterprises however the volume of activity.

System rate will be determined by the SaaS provider, to define the functional system needs and the minimum cost for the system, by calculating the required storage space, memory, processing power, maintenance procedures and schedules backup.

Tenant rate will be determined for each tenant individually, it is used to measure the non-functional requirements of tenants such as the desirable level of data isolation, the level of data security, the required query performance and number of extra backups. It is used to define the additional cost for every tenant according to his individual non-functional requirements.

Attribute rate is concerned with the growth of the attribute usage, by taking into account the attribute data type and constrains and whether it was an index attribute.

Table rate is a special rate used to determine every generated table workload, by measuring the growth of the data within each table. On the grounds of the experimental study conducted, which we will discuss in details later, we cater for a set of measurements to compute the four rates as in Table II, and present the suggested weights from these measurements.

The multi-tenant solution must fits functional requirements of the system, which represent by '*system rate*', plus the non-functional requirements of the tenant, which represent by '*tenant rate*'. In that manner, general tenant cost equals the minimum system cost '*system rate*' plus the extra tenant cost '*tenant rate*'.

Our proposal system save multimedia objects, like images and secure documents, in a set of separate databases for files and documents. However, each database has a level of security, a data encryption system and a special backup system that refer by the *database rate*. For example, if a tenant has secure documents and hopes to save them in a separated database, he should have a *tenant rate* high enough to allow him to use one of these separated databases, otherwise his tenant rate is not enough to allow him to use this feature.

TABLE II. THE SUGGESTION MEASUREMENTS TO COMPUTE THE SYSTEM RATE, TENANT RATE, ATTRIBUTE RATE AND TABLE RATE

	Measurement Description	Weight
System rate	The number of prospective tenants	20 %
	Average of tables per tenant and prospective data	20 %
	The nature of prospective tenants	10 %
	The prospective non-functional requirements of tenants	10 %
	Number of functions and procedures	10 %
	Minimum security level	10 %
	Minimum isolation level	10 %
	Minimum performance level	10 %
Tenant rate	Maximum number of users for this tenant	20 %
	Growth rate of tenant data (average of transactions per period)	20 %
	Special isolation rate	10 %
	Special security rate	10 %
	Special performance rate	10 %
	Number schedule backup	10 %
Attribute rate	Date type	5 %
	Column type(primary key, index)	5 %
	Foreign key	5 %
	Have constraints(unique, check)	5 %
Table rate	Table growth rate = number of rows / number of days from create	10 %
	Growth rate per tenant= number of rows / number of tenants	10 %
	Total current "*general attributes rate*"	80 %

In multi-tenancy environment, SaaS providers wish to reduce wasted storage space and maximize the sharing of resources, although the tenants wish to maximize the isolation and performance qualities, neglecting storage space. Therefore, we propose to use two thresholds the *isolation point* and *merger point*, to detect when to isolate data in multiple tables or to share data in the same table. Data in a multi-tenancy database will be partitioned horizontally to a set of allotment tables according to *isolation point*, and vertically to a set of extension tables according to *merger point*.

Isolation point is a threshold point that determines the necessity of horizontal partition data, in order to constrict the tables workloads with a concrete point, and it was assigned by the system variables using a several methods according to the nature of application.

The following equations used to calculate the *Isolation point* depending on the isolation factor only, where IP is the isolation point, IR is the isolation rate, SR is system rate, TR is tenant rate, AR is attribute rate and GTR as generate table rate. By the same token, $\text{IR}_{\text{current System}}$ is the current isolation degree of the system, to be accepted by the customers, and it detect by the system provider depending on the nature of the system, $\text{IR}_{\text{average of tanants}}$ is the average of special isolation rates for the tenants who sharing the system, $\text{IR}_{\text{general}}$ the average degree of data isolation rate for both the tenants and the system, note that $\text{IR}_{\text{general}}$ must be less than or equals 1.

$$IR_{total} = IR_{current\ System} + IR_{average\ of\ tanants} \quad (1)$$

$$IR_{maximum} = IR_{maximum\ system} + IR_{maximum\ tanants} \quad (2)$$

$$IR_{general} = \frac{IR_{total}}{IR_{maximum}} \quad (3)$$

$$IP = \left(1 - IR_{general}\right) * SR_{maximum} \quad (4)$$

In (5), $AR_{general}$ is a general attribute rate, that calculate a new attribute expected workload, and the required resources by a special tenant.

$$AR_{general} = TR * AR \quad (5)$$

Equation (1), (2), (3) and (4) calculate the isolation point, that support multi-levels of data isolation feature. In the same manner we can rebuild these equations where IR replace with the security rate to support multi-levels of data security.

Merger point is a threshold point, that responsible for vertical partition data, in order to reduce the volume of joining operations, it was assigned by the SaaS provider according to a set of factors, such as: the available storage space, the space will be allocated to each tenant, the number of tables will be allocated for each tenant, the expected extra customization attributes for each tenant, and availability to save a null values.

In the proposed system, when a tenant needs to create a custom table or alter an existing table, the *runtime table generator* searches for the best isolation approach by selecting the ideal *generated tables* to save the data in them. The algorithm in "Pseudocode 1" explain how the *runtime table generator* add a new custom table 'ACT' to a tenant 'test'. The algorithm starts by saving the data of a new customized table in the metadata-driven schema tables (line 3), then selecting or creating the appropriate generated tables, to store the tenant data and refer to it in the metadata-driven schema tables. In (lines 4-6) the algorithm check if the new *general attribute rate* is more than or equal the '*isolation point*', then creating a new generated table with this schema and returns its identifier as a reference. Otherwise the algorithm searches in all generated tables, and if it finds a generated table has the new customized table schema and its table rate plus the new general attribute rate are less than or equal to the *isolation point*, then returning this generated table Identifier as a reference (lines 7-12). In the failure case, the algorithm attempts to find all the tenants who need this schema, and if the summation of their *tenant rates* plus *tenant rate* of 'test' is more than or equal to the *isolation point*, a new generated table is created and its identifier is returned (lines 14-19). The last case, a new customized table is divided to a set of sub tables, and the algorithm determines whether this schema is available or it needs to create a new generated tables and return the tables references (lines 20-23).

Pseudocode 1 Creating a new Customized table 'ACT' to tenant 'test'.

```
1:  IP  ← Isolation Point
2:  MT ← Meta-data Table
3:     insert ACT into MT
4:     if ( TenantRate (test) >= IP )  then
5:     create (a new generate table with ACT schema)
6:     return ( the new generate table ID )
7:     else   for each (GT in Generated Tables)
8:                   GT ← current generated table
9:                   if (Schema(GT)=Schema(ACT))
                     and (TableRate(GT)+TableRate(ACT)<=IP)
10:                  then return ( the current generated table ID )
11:                  end if
12:            end for
13:    end if
14: X ← distributed generated tables require ACT schema
15: if (TableRate (X) + TableRate (ACT) >= IP) then
16:        create (a new generate table with ACT schema)
17:        update MT replace with the new generate table ID where X
18:        move data in X into the new generate table
19:        return ( the new generate table ID )
20:        else   for each( sub table in ACT )
21:               [: : :] /* Symmetric to lines 7 to 19 */
22:               [: : :] /* Symmetric to lines 5 to 6  */
23:               end for
24: end if
```

D. *Analyzing the performance of the proposed multi-tenant database schema-mapping technique*

The traditional DBMSs usually consists of 4 basic operations: selection, insertion, deleting and updating operation, and each operation is a collection of an I/O processes in the DBMS. The final target of our solution is twofold: firstly, the system aim at reducing the number of the I/O processes by saving the tenant data in one table. This means that a tenant inserts or updates a row to one physical source table, which means a small response time and a high throughput rate. This serves the tenants whose *tenant rate* is high enough, or high number of tenants with the same schema to allow the merge of their data in one table, which is expected to provide a high quality of service while wasting a lot of storage space; secondly, the system aim at reducing the number of generated tables by maximizing the sharing tables between tenants, by dividing the tenants entities vertically to a set of extension tables in order to eliminate the null values and economize on the storage space for tenants who do not have demanding quality of service, where a single operation can be divided into several I/O processes to the corresponding physical tables, which means a high response time and a lower throughput rate. This approach provides a relatively low quality of service while economizing on storage space.

IV. Experimental Study

In this section, we conduct a case study for a customer relationship management system in multiple hotels to evaluate our proposal. In the beginning, the hotels organizations are divided into multiple levels categories, as illustrated in Table III. All hotels have a common entities such as: room, travel agent and guest entity, and a common procedures such as: reservation operation. However, multiple hotels have different services types, citizenships of clients, and various quality of services. As a result, the common entities of multiple hotels will vary in attributes such as: table of customers. Table IV illustrates a sample of the variety on schema for an entity 'customer' for seven hotels.

The case study consists a sample of customers from “tenant1” to “tenant7”, representing the different segments of hotels. According to the column "Tenant Rate" in Table IV, there are a small enterprises such as “tenant1”and “tenant2”, who have a small workload and do not have demanding quality of service, the system ought to economize storage space for these tenants. However, tenants as “tenant7” and “tenant6” reject to share their tables with others.

In Table V, we compare the performance of our proposed approach with the three main approaches of data isolation in multi-tenancy database. We realize the proposed solution by applying the following steps:

TABLE III. THE MAIN CATEGORIES OF THE HOTELS COMPARISON

Hotel Class	Percentage of Market	Avg. of Users	Avg. of Reservations	Avg. of Transactions	Security Need	Variability need
three stars	80%	2	224	20	Low	Low
four stars	10%	3	689	176	Medium	High
five stars standard	5%	6	734	477	High	High
five stars deluxe	5%	10	1518	158	V. High	V. High

TABLE IV. VARIANT DEFINITIONS OF 'CUSTOMER' ENTITY FOR VARIANT TENANTS

Tenant name	Hotel Level	Tenant Rate	Attribute 1	Data Type	Attribute 2	Data Type	Attribute 3	Data Type	Attribute 4	Data Type
TENANT7	*****D	80	Id	*Int*	Name	*Char 150*	Web Site	*Char 100*	Email	*Char 100*
TENANT6	*****S	69	Id	*Int*	English Name	*Char 150*	Arabic Name	*N Char 200*	Phone	*Char 20*
TENANT5	*****S	49	Id	*Int*	Name	*Char 100*	Address	*Char 500*		
TENANT4	****	34	Id	*Int*	Name	*Char 200*	Tel	*Char 20*		
TENANT3	****	28	Id	*Int*	Short Name	*Char 50*	Full Name	*Char 200*	Site	*Char 100*
TENANT2	***	20	Id	*Int*	Name	*Char 100*	Email	*Char 200*		
TENANT1	***	9	Id	*Int*	Name	*Char 50*	supervisor	*Char 200*		

a) Table VI, we studied the functional requirements for the prospective tenants and the critical system needs to get the system rate.

b) Table VII, we got the tenant rate for each tenant, through studying the non-functional requirements for them, and calculated the extra costs for each tenants.

c) Using equation (4) to get the isolation point through depending on isolation rate.

d) Table VIII illustrate the generated tables GT_i where $i = 1,2,\dots 9$. schemas, and how to use these to represent the entity 'customer' for all tenants.

A. Implementation with the current isolation rate

In the current case, the system provider set the *system isolation rate* = 5, the minimum= 0 and maximum = 10. Note that the maximum *system rate* equal 100.

$$\text{Total isolation rate} = \frac{5}{10} + \frac{31}{70} = 0.5 + 0.44 = 0.94$$

$$Maximum\ isolation\ rate = \frac{10}{10} + \frac{70}{70} = 1 + 1 = 2$$

$$General\ isolation\ rate = 0.94/2 = 0.47$$

$$Isolation\ point = (1 - .47) * 100 = 53$$

In our case study, isolation point equal 53, so that tenants such as: “tenant$_6$”, “tenant$_7$” who have tenant rate more than or equal 53 will be in separated tables, but the other tenants will be in a shared tables according to their tenant rates.

B. Implementation the minimum system isolation rate

In this case, data isolation quality is not necessary in the system, so the system in common is opt for working like a *shared tables schema*, so it resulted *isolation point* =78. In the final analysis, only “tenant$_7$” who have a *tenant rate* >= 78 will be in a separated tables, and other tenants will be in *a shared table schema*. In conclusion, this case work like a shred tables schema and neglect the performance for most tenants. Finally, it use less number of tables and less storage space.

TABLE V. CHARACTERISTIC OF REALIZATION THE FOUR SCHEMAS

Solution	Tenant cost	customizable	Space requirements	Handle db size	Prospective tenants
Separate databases	V. High	V. High	V. High	Low	5 star Hotels
Separate schemas	Medium	High	Medium	High	4 star and part of 3 star
Shared Table	Low	High	Low	V. High	3 star
Propose Solution	made to order	V. High	made to order	High	All Hotels

$$General\ isolation\ rate = (0 + .44) / 2 = 0.22$$

$$Isolation\ point = (1 - 0.22) * 100 = 78$$

C. Implementation the maximum system isolation rate

In this case, it is opt for working like a *separated tables schema*, where all tenants having a *tenant rate* >= 28 will be in a separated tables. In conclusion, this case use more tables and neglect the storage space size and the number of tables.

TABLE VI. ILLUSTRATE SYSTEM RATE MEASUREMENT

Measurement Description	**Weights rate**
The number prospective tenants	8
Average number of table per tenant	5
the nature of prospective tenants	2
the needs of prospective tenants	3
Number of functions and procedures	3
Security Rate	4
Isolation Rate	5
Performance Rate	4
the system rate	**51**

TABLE VII. ILLUSTRATE THE TENANTS' RATES IN THE CASE STUDY

	No of users	**Growth rate**	**Isolation Rate**	**Security Rate**	**Performance rate**	**No of Backup**	**tenant rate**
Max. Value	**20**	**20**	**10**	**10**	**10**	**10**	**80**
Tenant 1	2	3	1	1	1	1	9
Tenant 2	6	6	1	1	4	2	20
Tenant 3	8	8	2	3	5	2	28
Tenant 4	8	10	5	2	6	3	34
Tenant 5	15	16	5	5	5	3	49
Tenant 6	18	20	7	8	8	8	69
Tenant 7	20	20	10	10	10	10	80
	AVG		31/70	30/70	39/70		

TABLE VIII. ENTITY 'CUSTOMER' SCHEMA IN THE GENERATED TABLES

Table name	**Column 1**	**Column 2**	**Column 3**	**Column 4**	**Column 5**	**Shared by tenants**	**Table rate**	**Available points**
GT1	UID	INT	char 150	char 100	char 100	T7	80	0
GT2	UID	INT	char 150	N char 200	char 20	T6	69	0
GT3	UID	INT				T5	49	4
GT4	UID	char 100				T5	49	4
GT5	UID	char 500				T5	49	4
GT6	UID	INT	char 200	char 100		T2,T3	48	5
GT7	UID	char 50				T1,T3	37	16
GT8	UID	INT	char 200			T1,T4	43	10
GT9	UID	char 20				T4	34	19

In Table VIII, the generated tables start with the *Global Unique Identification* column, that is represented by (UID) column, and then list the attributes which the generated tables ware contained. The seventh column 'Shared by tenants' shows who the tenant are used this generated table, such as GT1 is specified for "tenant$_7$". The eighth column is the table rate equal summation of its *tenants' rates*. finally the last column is the available points is *table rate* subtracted from *isolation point*, the tables have available point more than zero can be shared with tenants have tenant rate less than or equal this rate.

In the final analysis, the three implementation cases achieved the required theory, because the system had a multiple levels of data isolation. Fig. 1 illustrates the variety of data isolation approaches in the system, where vertical axis refer to the percentage of tenants and horizontal axis refer to the $\text{System Isolation Rate}_= i \text{ where } i = 0, 0.1, 0.2, \dots 1$. Furthermore, Fig. 2 illustrates how the needs of tenants can automatically affect the isolation rate. Decreasing the isolation rate mean increasing the separated tables or the storage space.

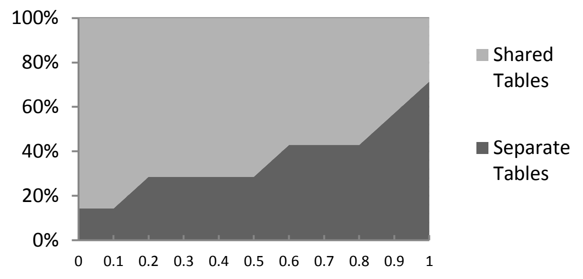

Fig. 1. Example of variation of data isolation schemas in a single system

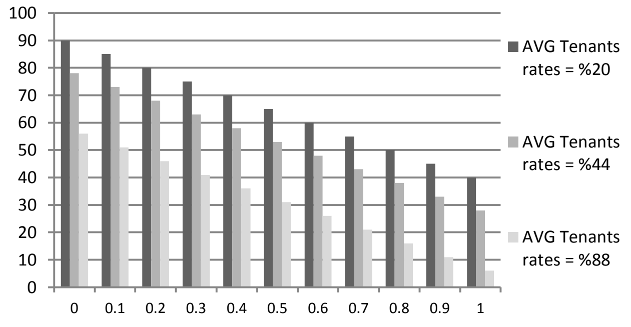

Fig. 2. The effect of change the avarage of tenants rates on isolation rate

V. CONCLUSION

In this paper, we trade-off between the performance and the storage space, in the major approaches of managing data in multi-tenant databases, as well as highlighted the standard components to build a customizable multi-tenant database schema. This paper proposes a new multi-tenant database schema-mapping technique, contain a mixed mode of the multi-tenancy data isolation approaches, in order to improve the database server utilization and minimize the wasted storage space, while keeping the appropriate quality of service for each tenant, by selecting the effective way to part the data in multi-tenant databases horizontally by the isolation point and vertically by the merger point.

REFERENCES

[1] H. Yaish, M. Goyal, and G. Feuerlicht, "An Elastic Multi-tenant Database Schema for Software as a Service," 2011 IEEE Ninth Int. Conf. Dependable, Auton. Secur. Comput., pp. 737–743, Dec. 2011.

[2] R. F. Chong, "Designing a database for multi-tenancy on the cloud Considerations for SaaS vendors," pp. 1–12, 2012.

[3] M. H. M. Hui, D. J. D. Jiang, G. L. G. Li, and Y. Z. Y. Zhou, "Supporting Database Applications as a Service," 2009 IEEE 25th Int. Conf. Data Eng., pp. 832–843, Mar. 2009.

[4] M. N. A. Khan, A. Shahid, and S. Shafqat, "Implementing a Storage Pattern in the OR Mapping Framework," Int. J. Grid Distrib. Comput., vol. 6, no. 5, pp. 29–38, Oct. 2013.

[5] S. Aulbach, M. Seibold, and S. a P. Ag, "A Comparison of Flexible Schemas for Software as a Service," Proc. 35th SIGMOD Int. Conf. Manag. data, pp. 881–888, 2009.

[6] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-tenant databases for software as a service: schema-mapping techniques," Proc. ACM SIGMOD Int. Conf. Manag. Data, pp. 1195–1206, 2008.

[7] Liao, K. Chen, and J. Chen, "Modularizing Tenant-Specific Schema Customization in SaaS Applications," AOAsia '13 Proc. 8th Int. Work. Adv. Modul. Tech., pp. 9–11, 2013.

[8] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan, "Towards Multi-Tenant Performance SLOs," Data Eng. (ICDE), 2012 IEEE 28th Int. Conf., pp. 702 – 713, 2012.

[9] J.. Ni, G.. Li, J.. Zhang, L.. Li, and J.. Feng, "Adapt: Adaptive database schema design for multi-tenant applications," ACM Int. Conf. Proceeding Ser., pp. 2199–2203, 2012.

[10] S. Foping, I. M. Dokas, J. Feehan, and S. Imran, "A new hybrid schema-sharing technique for multitenant applications," 2009 Fourth Int. Conf. Digit. Inf. Manag., 2009.

[11] Bezemer and A. Zaidman, "Challenges of Reengineering into Multi-Tenant SaaS Applications," 2010.