Survey on Multi-Tenant Data Architecture for SaaS

Li heng[1], Yang dan[2] and Zhang xiaohong[3]

**[1] College of Computer Science, Chongqing University
Chongqing, 401331, China**

**[2] School of Software Engineering, Chongqing University
Chongqing, 401331, China**

**[3] School of Software Engineering, Chongqing University
Chongqing, 401331, China**

Abstract

A multi-tenant database is the primary characteristic of SaaS, it allows SaaS vendors to run a single instance application which supports multiple tenants on the same hardware and software infrastructure. This application should be highly customizable to meet tenants' expectations and business requirements. This paper examines current solutions on multi-tenancy that provide flexible data model, and reviews their architecture, approaches and performance. Experimental results show that different mapping technique has its own benefits and drawbacks, and the ideal database system for SaaS need to be developed.

Keywords: *Multi-Tenant, Software as service, Schema mapping, meta data.*

1. Introduction

Software as a Service (SaaS) is an emerging software application service and one of the hot topics in the software industry. Expressed most simply, SaaS can be defined as follows: "Software deployed as a hosted service and accessed over the Internet" [1]. Instead of paying for the software license, the end user subscribe for a paid application. In February 2000, SaaS concept started when Salesforce.com launched their web-based service and became the early SaaS adopters. In February 2001 the term Software as a Service or SaaS published for the first time in a white paper called "Software as a Service: Strategic Backgrounder" [2]. SaaS began to flourish in 2005-2006, because the internet speed had significantly increased, had become affordable, and customers had started to be more comfortable to establish business over the internet.

A particularly important challenge in a SaaS application is concerned with enabling multi-tenancy at the data tier [3, 4]. Systems at the data tier of a SaaS application are accessed by the same application for each tenant, who has own unique needs that a rigid, inextensible default data model won't be able to address. Put simply, the challenge is to consolidate multiple tenants onto one data tier resource, e. g. one database server, which can be extended for different versions of the application and dynamically modified while the system is on-line, while at the same time isolating them among one another, as if they were running on physically segregated resources.

In this article, first we'll introduce three distinct approaches for creating data architectures. Then, based on the above three approaches, we'll explore some multi-tenant database schema mapping techniques for ensuring security, creating an extensible data model, and scaling the data infrastructure. Finally, we make a experimental comparison of those techniques described in section 3 for implementing flexible schemas for SaaS.

2. Three Approaches to Managing Multi-Tenant Data

The typical character of SaaS applications is 'single-instance multi-tenancy', according to this feature, three main approaches have been proposed [5].

2.1 Separate Database

Storing tenant data in separate databases is the simplest approach to data isolation.

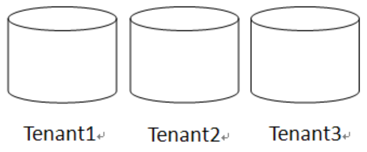


Fig. 1 separate databases

In this approach, each tenant has its own set of data that remains logically isolated from data that belongs to all other tenants, database security prevents any tenant from accidentally or maliciously accessing other tenants' data.

It is very easy to extend the application's data model to meet tenant's individual needs, and can simply restoring a tenant's data from backups. However, it costs higher for the relatively high hardware and maintenance requirements. This approach is suit for customers who are willing to pay extra for added security and customizability. For example, customers in fields such as social security or banking often have very strong data isolation requirements.

2.2 Shared Database, Separate Schemas

This approach involves housing multiple tenants in the same database, with each tenant having its own set of tables that are grouped into a schema created specifically for the tenant.

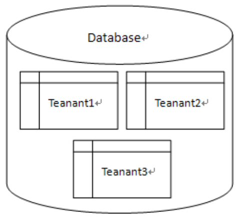


Fig. 2 separate schemas

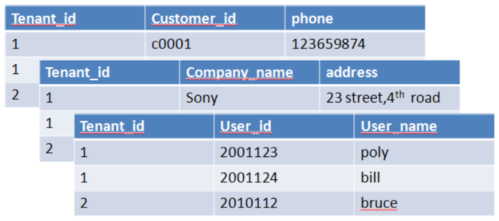
The separate-schema approach is relatively easy to implement, and tenants can extend the data model as easily as with the separate-database approach. This approach offers a moderate degree of logical data isolation for security-conscious tenants, though not as much as a completely isolated system would, and can support a larger number of tenants per database server.

A significant drawback of the separate-schema approach is that tenant data is harder to restore in the event of a failure. If each tenant has its own database, restoring a single tenant's data means simply restoring the database from the most recent backup. With a separate-schema application, restoring the entire database would mean overwriting the data of every tenant on the same database with backup data, regardless of whether each one has experienced any loss or not. Therefore, to restore a single customer's data, the database administrator may have to restore the database to a temporary server, and then import the customer's tables into the production server—a complicated and potentially time-consuming task.

The separate schema approach is appropriate for applications that use a relatively small number of database tables, on the order of about 100 tables per tenant or fewer. This approach can typically accommodate more tenants per server than the separate-database approach can, so you can offer the application at a lower cost, as long as your customers will accept having their data co-located with that of other tenants.

2.3 Shared Database, Shared Schema

A third approach involves using the same database and the same set of tables to host multiple tenants' data. A given table can include records from multiple tenants stored in any order; a Tenant ID column associates every record with the appropriate tenant.



Tenant_id	Customer_id	phone
1	c0001	123659874
1		
2		

Tenant_id	Company_name	address
1	Sony	23 street,4th road
1		
2		

Tenant_id	User_id	User_name
1	2001123	poly
1	2001124	bill
2	2010112	bruce

Fig. 3 shared schema

Of the three approaches explained here, the shared schema approach has the lowest hardware and backup costs, because it allows you to serve the largest number of tenants per database server. However, because multiple tenants share the same database tables, this approach may incur additional development effort in the area of security, to ensure that tenants can never access other tenants' data, even in the event of unexpected bugs or attacks.

The procedure for restoring data for a tenant is similar to that for the shared-schema approach, meanwhile, individual rows in the production database must be deleted and then reinserted from the temporary database. All the tenants that the database serve will be suffer noticeably in the procedure.

The shared-schema approach is appropriate when it is important that the application be capable of serving a large number of tenants with a small number of servers, and prospective customers are willing to surrender data isolation in exchange for the lower costs that this approach makes possible.

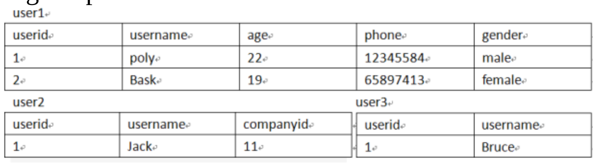
3. Multi-Tenant Data Architecture Techniques

Based on the above three approaches, Several works have been presented in [6], [7], [8], [9] on design and implement multi-tenant database schema, such as Private Table, Extension Table, Universal Table and so on, each technique has its' own characteristics and applicable scenarios, This section will explore eight techniques of multi-tenant database schema.

3.1 Private Table

The most basic way to support extensibility is to give each tenant their own private tables which can be extended and changed. Aulbach et al. [8], [9] state that Private Tables' technique allows each tenant to have his own private tables. Simply by renaming tables, we can transform the query from one tenant to another, and we don't need to use extra columns like " tenantid" to distinguish and isolate tenants data.

Fig.4 show three tenants, each tenant has different business requirements, so there exists three different user tables. In contrast, many tables are required to satisfy each tenant needs, therefore this technique can be used if there are few tenants using it, to produce sufficient database load and good performance.



user1

userid	username	age	phone	gender
1	poly	22	12345584	male
2	Bask	19	65897413	female

user2

userid	username	companyid
1	Jack	11

user3

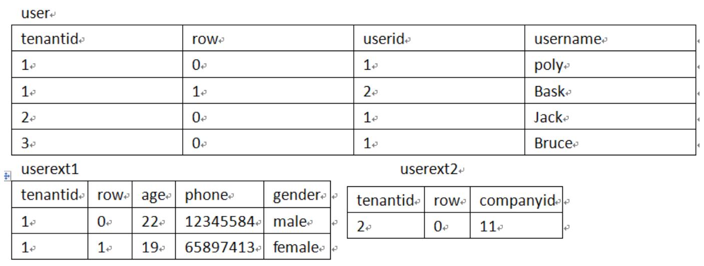
userid	username
1	Bruce

Fig.4 Private Tables

3.2 Extension Table

Because multiple tenants may use the same base tables, Aulbach et al. [8], [9] report that the Extension Tables are separated tables joined with the base tables by adding tenant column as well as row column to construct a logical source tables. This approach has its origins in the Decomposed Storage Model [11], where an n-column table is broken up into n 2-column tables that are joined through surrogate values. Multiple tenants can use the base tables as well as the extension.

The Extension Tables in Fig. 5 show how the columns of the user tables for the three tenants split-up between the base table "user" and the two extension tables. All of these three tables have two fixed common columns "tenantid" and "row". The "tenantid" column is used to map data records in the base table and the extension tables with the tenant who owns these records. The "row" column is used to give each record in the base table a row number and map it with other records in the extension tables. The last two columns of "user" table are shared between all the tenants. The table "userext1" is used by tenant 1. The "userext2" used by tenant2. This technique provides better consolidation than the Private Tables explained above. Nevertheless, the number of tables will be increased by increasing the number of tenants and the variety of their business requirements.



user

tenantid	row	userid	username
1	0	1	poly
1	1	2	Bask
2	0	1	Jack
3	0	1	Bruce

userext1

tenantid	row	age	phone	gender
1	0	22	12345584	male
1	1	19	65897413	female

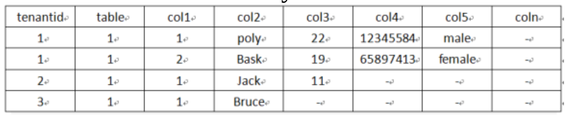
userext2

tenantid	row	companyid
2	0	11

Fig.5 Extension Tables

3.3 Universal Table

Aulbach et al. [8] refer to Universal Table as a table that contains additional columns of the base application schema columns which enable tenants to store their required columns. A Universal Table is a generic structure with a Tenant column, a Table column, and a large number of generic data columns. The data columns have a flexible type, such as VARCHAR, into which other types can be converted. The n-th column of each logical source table for each tenant is mapped into the n-th data column of the Universal Table. As a result, different tenants can extend the same table in different ways.



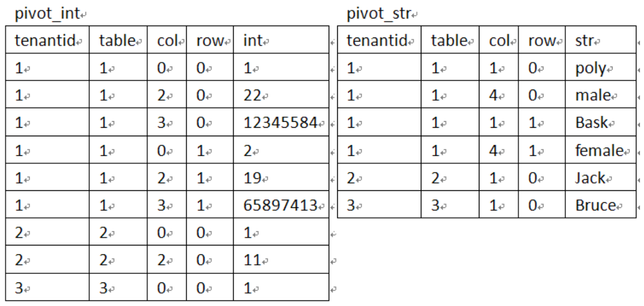
tenantid	table	col1	col2	col3	col4	col5	coln
1	1	1	poly	22	12345584	male	-
1	1	2	Bask	19	65897413	female	-
2	1	1	Jack	11	-	-	-
3	1	1	Bruce	-	-	-	-

Fig.6 Universal Table

The Universal Table in Fig.6 shows how the tenants data records are stored in one universal table. The "tenantid" column is used to map records with their tenants. The "table" column is used to map records to particular tables. Columns "col1" until "coln" are the universal columns that stores any data tenants wish to store. This is a flexible technique which enables tenants to extend their tables in different ways according to their needs. However it has the obvious disadvantage that the rows need to be very wide, even for narrow source tables, and the database has to handle many null values. Furthermore, indexes are not supported in universal table columns, as the shared tenant's columns might have different structure and data type. This issue leads to the necessity of adding additional structures to make indexes available in this technique.

3.4 Pivot Table

In a Pivot Table, each row field in a logical source table is given its own row. There are four columns in the Pivot Table including: tenant, table, column, and row that specify which row in the logical source table they represent. In addition, the single data type column that stores the values of the logical source table rows according to their data types in the designated pivot Table. The data column can be given a flexible type, such as VARCHAR, into which other types are converted, in which case the Pivot Table becomes a Universal Table for the Decomposed Storage Model.



pivot_int

tenantid	table	col	row	int
1	1	0	0	1
1	1	2	0	22
1	1	3	0	12345584
1	1	0	1	2
1	1	2	1	19
1	1	3	1	65897413
2	2	0	0	1
2	2	2	0	11
3	3	0	0	1

pivot_str

tenantid	table	col	row	str
1	1	1	0	poly
1	1	4	0	male
1	1	1	1	Bask
1	1	4	1	female
2	2	1	0	Jack
3	3	1	0	Bruce

Fig.7 Pivot Tables

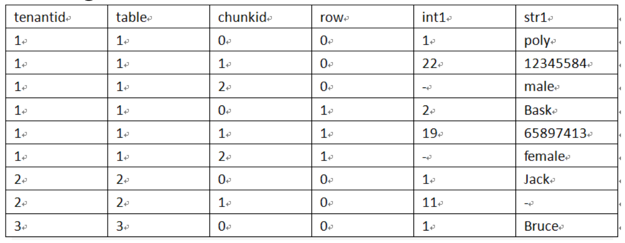
The Pivot Tables in Fig. 7 show how data with a specific data type are stored in a specific pivot table. In our example we have two tables, the first pivot table is “pivot_int” which stores integer data values, and the second one is “pivot_str” which stores String data values. To efficiently support indexing, two Pivot Tables can be created for each type: one with indexes and one without. Each value is placed in exactly one of these tables depending on whether it needs to be indexed.

The performance benefits from this technique can be achieved by eliminating NULL values, and from selectively read from less number of columns. However, this approach eliminates the need to handle many null values. However it has more columns of meta-data than actual data and reconstructing an n-column logical source table requires (n−1) aligning joins along the Row column. This leads to a much higher runtime overhead for interpreting the meta-data than the relatively small number of joins needed in the Extension Table Layout.

3.5 Chunk Table

A Chunk Table is similar with a Pivot Table except that it has a set of data columns of various types, with and without indexes, and the Col column is replaced by a Chunk column. This technique partitioned logical source table into groups of columns, each group assigned to a chunk ID and mapped into an appropriate Chunk Table.

The Chunk Table in Fig. 8 shows how a set of data columns with a mixture of data types are structured. The table “ChunkTable” has six columns. The “tenantid” column is used to map each record in a chunk table with its tenant. The “table” column is used to map a record to particular logical table. The “chunkid” column is used to compound data for more than one logical column for a particular logical table. The “row” column is used to map a data value to a particular logical row in a particular logical table. The “int1” and “str1”column are used to store integer or string data values for different logical columns in the logical table.



tenantid	table	chunkid	row	int1	str1
1	1	0	0	1	poly
1	1	1	0	22	12345584
1	1	2	0	-	male
1	1	0	1	2	Bask
1	1	1	1	19	65897413
1	1	2	1	-	female
2	2	0	0	1	Jack
2	2	1	0	11	-
3	3	0	0	1	Bruce

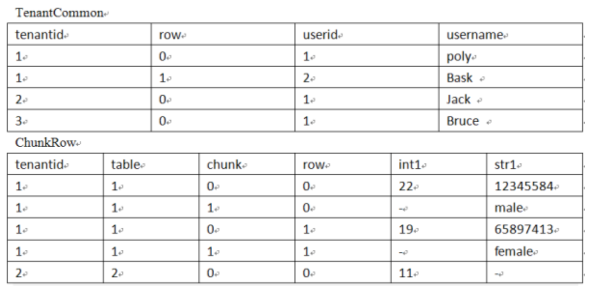
Fig.8 Chunk Tables

This technique has advantages over Pivot Table as it reduces metadata storage ratio, and the overhead of reconstructing the logical source tables, and also has advantages than Universal tables by providing indexes, and reducing the number of columns. Although, this technique is flexible, it adds complexity to the database queries.

3.6 Chunk Folding

Chunk Folding[8] is a technical where the logical source tables are vertically partitioned into chunks that are folded together into different physical multi-tenant tables and joined as needed. Aulbach et al. [8] state that the performance of this technique enhanced by mapping the most used tenants’ columns of the logical schema into conventional tables, and the remaining columns in the Chunk Tables which are not used by the majority of tenants.

Fig.9 illustrates a case where base Accounts are stored in a conventional table and all extensions are placed in a single Chunk Table. The “tenantid” column in both tables is used to map each record with its tenant. The “row” column in both tables is used to map a data value to a particular logical row in a particular logical table. The first table consists of four columns. The last column in this table is shared by the three tenants. The “table” column in the second table is used to map a record to particular logical table. The “chunk” column is used to compound data for more than one logical column for a particular logical table. The “int1” and “str1” column is used to store integer or string data values for different logical columns in the logical table.



TenantCommon

tenantid	row	userid	username
1	0	1	poly
1	1	2	Bask
2	0	1	Jack
3	0	1	Bruce

ChunkRow

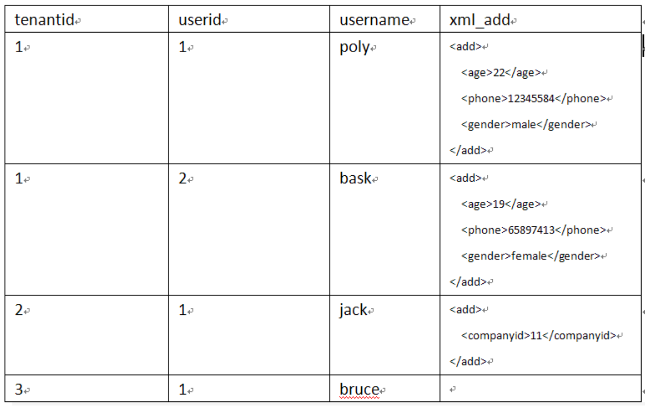
tenantid	table	chunk	row	int1	str1
1	1	0	0	22	12345584
1	1	1	0		male
1	1	0	1	19	65897413
1	1	1	1		female
2	2	0	0	11	

Fig.9 Chunk Folding tables

3.7 XML Table

The XML database extension technique is a combination of relational database systems and Extensible Markup Language (XML) [6,7]. Aulbach et al. [10] state that the extension of XML can be provided as native XML data type, or by storing the XML document in the database as a Character Large Object (CLOB) or Binary Large Object (BLOB). This technique satisfies tenants' needs, because their data can be handled without changing original database relational schema.

The XML technique in Fig. 10 shows how this technique combines relational database systems and XML. The "xml_add" column is used to store an XML structure which includes the rest of the logical columns which tenants might need to fulfill their business needs.



tenantid	userid	username	xml_add
1	1	poly	<add> <age>22</age> <phone>12345584</phone> <gender>male</gender> </add>
1	2	bask	<add> <age>19</age> <phone>65897413</phone> <gender>female</gender> </add>
2	1	jack	<add> <companyid>11</companyid> </add>
3	1	bruce	

Fig.10 XML tables

3.8 Elastic Extension Tables

The Elastic Extension Tables (EET) technique proposes a new way of designing and creating an elastic tenant database.

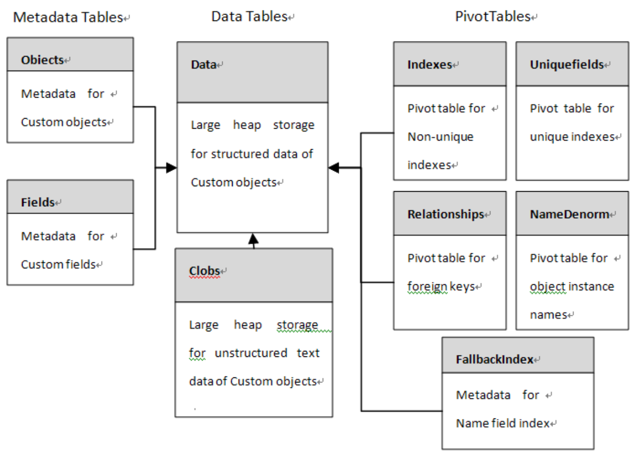

Metadata Tables
Data Tables
PivotTables
Objects
Metadata for Custom objects
Fields
Metadata for Custom fields
Data
Large heap storage for structured data of Custom objects
Clobs
Large heap storage for unstructured text data of Custom objects
Indexes
Pivot table for Non-unique indexes
Uniquefields
Pivot table for unique indexes
Relationships
Pivot table for foreign keys
NameDenorm
Pivot table for object instance names
FallbackIndex
Metadata for Name field index

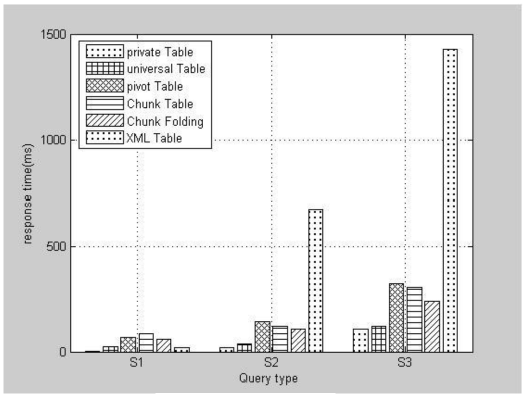
Fig.11 EET tables

Fig.11 shows the EET technique which is designed by Force.com. The tables in the left area are called "Metadata tables". They are used to record the objects which are customized by the end user and the field structure contained in the objects. The tables in the center area are called "Data tables", in contrast with "Metadata tables", they are used to record filed data of those objects. The left tables are called "Pivot Tables", which are used to accelerate the reading of some special data to improve overall system performance.

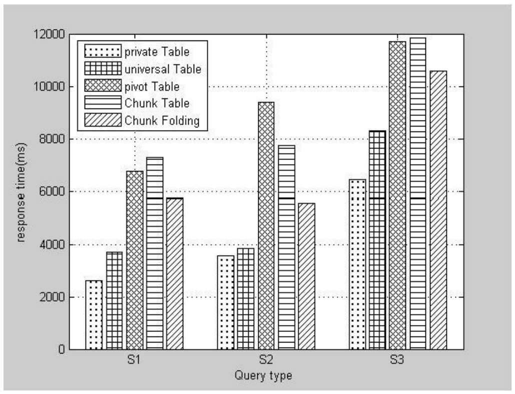
4. Experiments

This section describes an experimental comparison of above techniques for implementing flexible schemas for SaaS. Since there is no standard data set for this task, we generate our own multi-tenant data set from table customer in our student database. We append a tenantid column so that it can be shared by multiple tenants. Tenants have different sizes and tenants with more data have more extension fields. In the experiments, we simulate a real multi-tenant scenario by sending query and update requests from many tenants concurrently, and then evaluate the solutions by analysis the response time data captured during those experiments. The experiment has 3 requests classless. In order to avoid influence each other, multiple copies of the test schema are created. The experiment was run on a sqlserver database server with a 3.0 GHz Intel Xeon processor and 1 GB of memory.
S1: Select all attributes of a single entity as if it was being displayed in a detail page in the browser.
S2 : Select all attributes of 1000 entities as if they were being displayed in a list in the browser.
S3 : Select all attributes of 5000 entities as if they were being displayed in a list in the browser.

Fig.12 shows the results of the experiment. The horizontal axis shows the different request classes, and the vertical axis shows the response time in milliseconds on a log scale.



(a)Overall



(b)100 tenants

Fig12.experiment results

From Fig.12 We can see that private table has the smallest response time while XML table has is the highest. Universal table is faster than the others except private table. Pivot Table, Chunk Table and Chunk Folding are slower because an additional join is required. We can't say which technique is better, because each technique has its own advantages and drawbacks. For example, private table is the fastest technique, and the tenants can extend their needs easily, but it is suitable for those applications which confronted with small tenants. Sometimes, Universal table is a good choice, for it is fast and can flexible extend. The disadvantage of this technique is that the database has to handle many null values in wide table. The left techniques seemingly make a balance in efficiency, space and flexible, but they are just in the theory stage, and lack of an effective vertical partitioning algorithm to get the most appropriate results.

5. Conclusions

In this paper a short survey of Multi-Tenant Data Architecture was presented. First we introduced three approaches to managing multi-tenant data, then, we concluded 8 techniques on design and implement multi-tenant database schema, finally, we presented the results of several experiments designed to measure the efficacy of those techniques and made a comparison.

The conclusion we draw from this paper is that the ideal database system for SaaS has not yet been developed .Choose which technique or what approach depend on the Circumstances alter cases , for example, how many tenants use the system, or how to extension a data mode a tenant need. A goal of our on-going work is to develop algorithms that implement a schema mapping technique which is suitable for different circumstances.

Acknowledgment

This work is supported by State Natural Sciences Foundation Projects of China under Grant (91118005), Natural Science Foundation Project of Chongqing under Grant (CSTC 2011BA2022).

References

[1] F. Burno. "Exeuting an IP Protection Strategy in a SaaS Environment", http://www.slideshare.net/Rinky25/saas-environment, Jul. 22, 2011.

[2] Nitue, "Configurability in SaaS (software as a service) applications", ISEC , 2009, pp. 19-26.

[3] F. Chong and G. Carraro. Architecture Strategies for Catching the Long Tail. Microsoft Corp. Webs ite, 2006

[4] G. C . Freder ick Chong and R . Wolter. M ulti-Tenant Data Architecture. Microsoft Corp . Website, 2006.

[5] D. J acobs and S . Aulbach. Ruminations on Multi-Tenant Databases . In Proc. of BTW Conf., pages 514–521, 2007.

[6] F. S. Foping, I. M. Dokas, J. Feehan, and S. Imran, "A new hybrid schema-sharing technique for multitenant applications", ICDIM , 2009, pp. 1-6.

[7] 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.

[8] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multitenant databases for software as a service: Schema mapping techniques", SIGMOD , 2008, pp. 1195-1206.

[9] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and M. Seibold, "A Comparison of Flexible Schemas for Software as a Service", SIGMOD , 2009, pp. 881-888.

[10] G. P. Copeland and S. N. Khoshafian. A decomposition storage model. InSIGMOD '85:Proceedings of the 1985 ACM SIGMOD international conference on Management of data, pages 268–279, New York, NY, USA, 1985. ACM

[11] R.Mietzner, T.Unger, R.Titze, and F.Leymann, "Combining Different Multi-tenancy Patterns in Service-Oriented Applications", EDOC, 2009, pp. 131 -140.

Li Heng is a lecture in Chongqing University. Currently, he is a PhD student in College of Computer Science of Chongqing University. His interests are in cloud computing, data mining & machine learning .

Yang Dan is a professor of Chongqing University. Current research interests: data mining, computer vision, machine learning, enterprise informatization .

Zhang Xiao Hong is a professor of Chongqing University.