Spring 4-9-2014

A Review Of Multi-Tenant Database And Factors That Influence Its Adoption.

Olumuyiwa Matthew
University of Wolverhampton, o.o.matthew@wlv.ac.uk

Carl Dudley
University of Wolverhampton, carl.dudley@wlv.ac.uk

Robert Moreton
University of Wolverhampton, r.moreton@wlv.ac.uk

A REVIEW OF MULTI-TENANT DATABASE AND FACTORS THAT INFLUENCE ITS ADOPTION

Olumuyiwa Matthew*

Carl Dudley**

Robert Moreton***

**MSc in Information Technology Management, MCPN, AMBCS,*

The University of Wolverhampton, United Kingdom.

Email: o.o.matthew@wlv.ac.uk

***Oracle Professor in Database Technology University of Wolverhampton, UK.*

UK Oracle User Group Official, oracle ACE Director

Email: carl.dudley@wlv.ac.uk

****Emeritus Professor in IT from university of Wolverhampton, UK.*

Email: r.moreton@wlv.ac.uk

Abstract

A Multi-tenant database (MTD) is a way of deploying a Database as a Service (DaaS). This is gaining momentum with significant increase in the number of organizations ready to take advantage of the technology. A multi-tenant database refers to a principle where a single instance of a Database Management System (DBMS) runs on a server, serving multiple clients organizations (tenants). This is a database which provides database support to a number of separate and distinct groups of users or tenants. This concept spreads the cost of hardware, software and other services to a large number of tenants, therefore significantly reducing per tenant cost. Three different approaches of implementing multi-tenant database have been identified. These methods have been shown to be increasingly better at pooling resources and also processing administrative operations in bulk. This paper reports the requirement of multi-tenant databases, challenges of implementing MTD, database migration for elasticity in MTD and factors influencing the choice of models in MTD. An insightful discussion is presented in this paper by grouping these factors into four categories. This shows that the degree of tenancy is an influence to the approach to be adopted and the capital and operational expenditure are greatly reduced in comparison with an on-premises solution

Keywords: Multi-tenant, Database, Service, Tenant, Models.

1. INTRODUCTION

Deploying Database as a Service (DaaS) is gaining momentum with a significant increase in the number of organizations ready to take advantage of the technology. This architecture will reduce the cost incurred in the development and deployment of an on-premises database system both in hardware and software required for such a system. Multi-tenancy is a familiar word that comes from the real world of estate building. An estate building will provide a multi-tenant housing service to any number of tenants (Banville and Holzel 2012). This could be an individual, couple, family and groups of different sizes. Similarly, a multitenant database is one which provides database support to a number of isolated and different groups of users. The concept of multi-tenancy was developed from the service providing technology known as Software as a Service (SaaS). SaaS is a form of cloud computing that involves offering software services in an on-line and on-demand fashion with the internet as the delivery mechanism (Walraven et al 2014). Software as a Service constitutes a fast-growing business model for the sales of software that is based on the principle of outsourcing. With SaaS, a service provider hosts an application or software on its infrastructure and delivers it as a service to several organizations. An organization, also referred to as a tenant, subscribes for the service and accesses it across the Internet through standard web technology. (Schiller et al 2011 p117)

A multi-tenant database refers to a principle where a single instance of the Database management system (DBMS) runs on a server, serving multiple client organisations (tenants). Multi-tenant database is one which provides database support to a number of separate and distinct groups of users, also referred to as tenants. A tenant is simply any logically defined group of users that requires access to its own set of data. This definition was substantiated by Bezemer et al (2010 p1) as an architectural pattern in which a single instance of the software is run on the service provider's infrastructure, and multiple tenants access the same instance. This concept provides the ability of a system to provide database management services to different users or customers without causing interference with each other's processes. This reduces the effort made in production and the cost incurred in the development.

In a multi-tenant enabled service environment, user requests from different organizations and companies (*tenants*) are served concurrently by one or more hosted application instances and databases based on a scalable, shared hardware and software infrastructure (Gao et al 2011

p324). Such a database system must be able to maintain or even increase its performance or efficiency level under larger operational demands. This multi-tenant database system is a new technology that can be implemented in both host based and cloud based environments.

Today many companies want to outsource their data to a third party which hosts a multi-tenant database system to provide data management service. Each company is called a tenant. The multi-tenant data management system spreads the cost of hardware, software and professional services to a large number of tenants and thus significantly reduces per- tenant cost by increasing the scale. Thus the multi-tenant database system provides excellent performance, low-space requirement and good scalability (Ni et al 2012 p2199). With multi-tenancy, there is no need for customers having separated on site systems for their personal applications since such services and resources could be rendered to them through a service provider at a much more reduced cost.

A single instance of the software running on a single server will serve multiple client organisations (tenants), this is a principle also used in software architecture and is also referred to as multi-tenancy. Multi tenancy can minimize Hardware/Software costs and human costs per tenant. Multi-tenant database system has been exploited to store, manage, and retrieve data of tenants. A service provider hosts the multi-tenant database system and each tenant subscribes to the services by doing necessary configuration, loading data to the data centre, and then interacts with the services through a standard method, e.g., Web Services. Thus, the cost of ownership of database applications and the maintenance costs are transferred from the individual tenant to the service provider. (Ying et al 2011).

In this paper, we present past researchers' works in relation with multi-tenancy in database. And factors that influence the choice of MTD are highlighted to illustrate a background insight into the concept. The paper is organised as follows; section 2 reviews the literature on MTDs. while section 3 explores the factors influencing the choice of MTD. The detailed discussion of the re-grouped factors are presented in section 4. Conclusions are provided in the last section.

2. RELATED WORKS

With multi-tenancy features, SaaS providers can considerably ease operations and reduce delivery costs with a large number of tenants. As illustrated by Gao et al (2011 p324) that in a multi-tenant enabled service environment, one or more hosted application instances and

databases service the request from different organisations and companies concurrently on a scalable, shared hardware and software infrastructures.

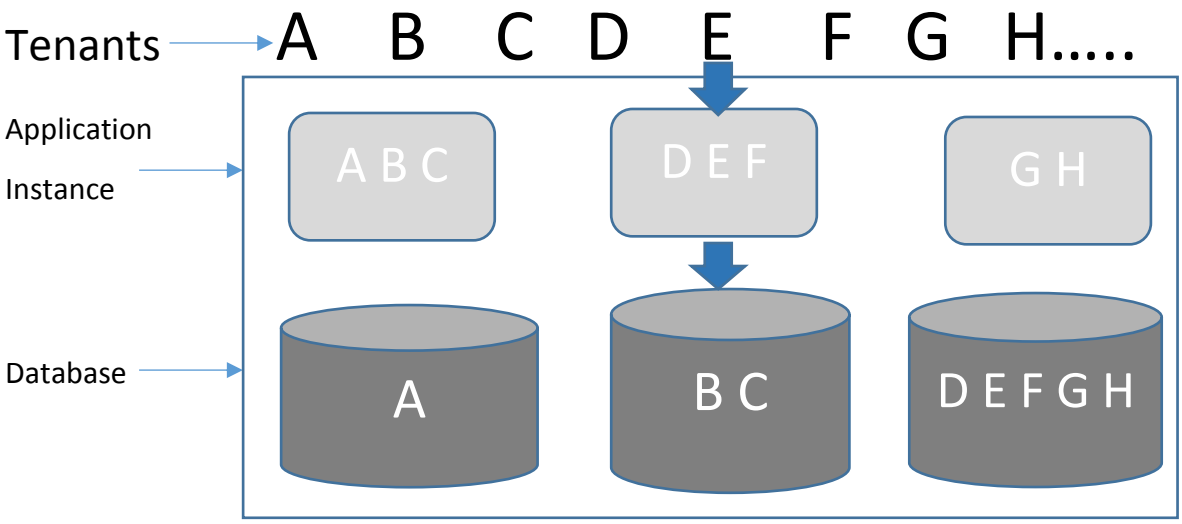

Figure 2 - A Multi-Tenant Enabled Service Environment.

Gao et al (2011) focus on the database layer of multi-tenancy as the most important and challenging part of SaaS application, looking at cost effective database sharing model and also on data security isolation among tenants must be guaranteed irrespective of the sharing model. The fact that different tenants with different service level demands and customization needs will further make this multi-tenancy adoption a constraint in practice. Gao et al came up with a cost-effective, secure, customizable, scalable, and non-intrusive multi-tenant database which will accelerate the migration and development of SaaS.

This Multi-tenancy architecture was also put forward by Walraven et al (2014 p2) as an architecture that can be applied at various levels of the software stack: at the infrastructure level (i.e. virtualization), at the OS and middleware level, and even at the application level. Banville and Holzel (2012 p2) also explained multi-tenancy and categorized them based on isolation and sharing of resources. There are four basic classes as shown in figure 3: isolated, infrastructure, application and shared tenancy. In isolated tenancy, each tenant has its own instance of the application running with its own instance of database, as well as its own infrastructure to support the deployment (Banville and Holzel 2012 p3). In infrastructure

tenancy, each tenant has its own instance of the application running with its own instance of database while they all share the same infrastructure. In this kind of multi-tenancy, each tenant has different application and different database but all on the same infrastructure. In application tenancy, each tenant has its own instance of database while at the application and infrastructure level they all run on the same platform.

Finally, shared tenancy has a scenario where all tenants run on the same instance of application, database and infrastructure. This approach is said to be the purest approach of multi-tenancy approaches where everything is been shared (Banville and Holzel 2012 p3).
As we move further to the right side of the figure 3, the sharedness of tenancy in each paradigm increases (Banville and Holzel 2012 p3). This means that as you move further to the right side of the diagram, the degree of sharing of the resources increases, taking it to a point where all the required resources are shared.

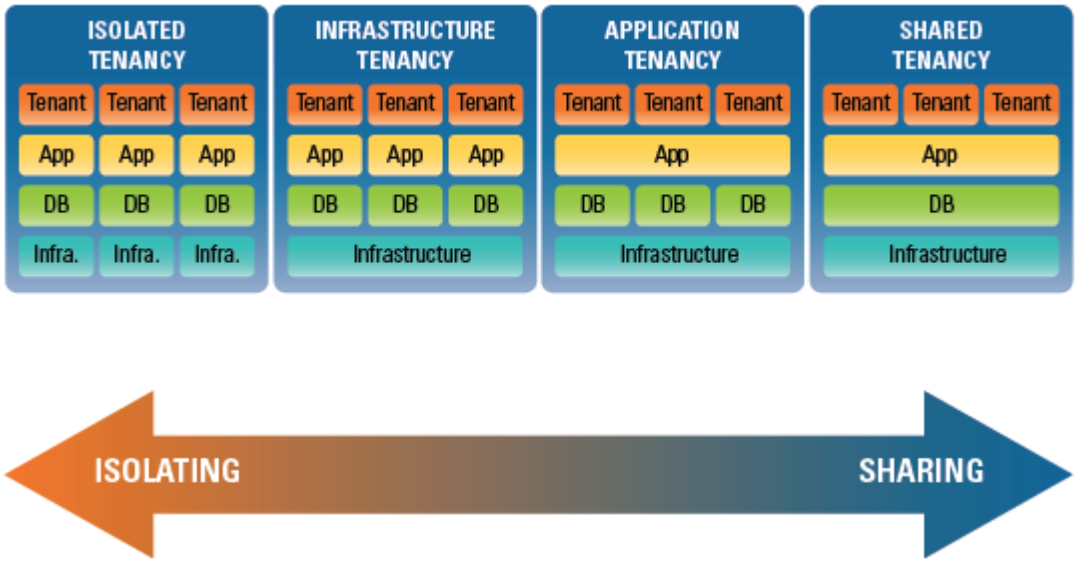

Figure 3 - The multi-tenancy continuum

There are some advantages and disadvantages noted by Banville and Holzel (2012 p3) concerning each of the approaches in figure 3. For example, if a tenant requires a customized version of the application and/or the database schema or has special infrastructure requirements, then isolated tenancy is the ideal approach. However, as the number of isolated tenancy deployments increases, the cost of maintaining them also increases. An application

provider maintaining an isolated tenancy deployment for an increasing number of tenants will have to invest considerably more time and effort to deploy code changes or carry out database maintenance for all of those tenants.

Also shared tenancy has a comparative reduction in maintenance cost due to one instance of application, database and infrastructure that is shared by all the tenants. It is easy to change code, maintenance is a straightforward and less costly than with isolated tenancy. Isolated tenancy has a high level of scalability due to the ease with which additional tenants can be added to the deployment. While shared tenancy has disadvantages of making it difficult to customize a particular tenant's code, customize schema, or provide any type of specialized service to a particular tenant. Tenants in a shared tenancy configuration may also have increased concerns over data security since their data is stored together on the same database instance (Banville and Holzel 2012 p3).

Walraven et al (2014) focuses on the application-level multi-tenancy which is reported to achieve the highest degree of resource sharing between tenants. Walraven et al also says that end users from different tenants are simultaneously served by a single application instance on top of the shared infrastructure. There is a crucial disadvantage of inherent limitations in variability when compared with infrastructure-level and middleware –level multi-tenancy. And only requirements that are common to all tenants are satisfied, no support for different and varying requirement of the different tenants. This increase amount of variations and an increasing amount of tenant – specific configuration has led to two essential challenges related to development and customization of multi-tenant applications identified by Walraven et al (2014 p3). There are;

1. SaaS providers need to be able to manage and reuse the different configurations and software variations in an efficient way, without compromising scalability; e.g. by avoiding additional over-head when provisioning new tenants.
2. Part of realizing the scalability benefits of SaaS is achieved by self-service: shifting some of the configuration efforts to the tenant side, e.g. by allowing the tenant to manage his tenant-specific requirements and by automating the run-time configuration process. Therefore, tenants require additional support to manage the configuration in a tenant-driven customization approach.

2.1 MULTI-TENANT DATABASE IMPLEMENTATION

Multi-tenant database architecture is very important for service providers who base their platform on SaaS. This helps in meeting up with the demands of customers or tenants on that platform of the provider. Multi-Tenants database architecture is very useful when one instance of database is serving to multiple clients. Only one set of hardware resources is needed to fulfil the requirements of all users. Multi-tenant is based on subscriber model, so user has freedom to avail the facility as per business requirement or can turnoff.

Jacobs and Aulbach (2007 p2) identified three different approaches in implementing multi-tenant databases which are: shared machine, shared process, and shared table. These approaches are increasingly better at pooling resources and executing administrative operations in bulk. However they increasingly break down the isolation between customers, weakening security and increasing contention for resources. Grund et al (2008 p2) make comparison based on the three approaches of Jacobs and Aulbach (2007), it states that in the shared machine approach, each tenants get their own database and resource sharing is done on machine level. In the shared process approach, the tenants share the same physical database process but own different databases. This allows better resource pooling between the tenants but still creates a lot overhead because the schemas need to be maintained separately for all tenants. The last approach is the shared table approach. Using shared tables the application schema is created once and the different tenants are mapped directly into this schema using different schema mapping techniques.

Schiller et al (2011 p118) further explain that these three approaches vary in the degree of consolidation and that the Shared Machine approach allows consolidating only a few tenants onto one machine due to the large main memory footprint of a database instance. The Shared Process approach consumes less main memory per tenant, yet main memory consumption increases quite fast with the number of tenants, as each tenant obtains a dedicated schema instance. In contrast, the main memory consumption of the Shared Table approach remains constant if the number of tenants increases. According to Schiller et al (2011 p118, the shared table approach seems promising for a provider that targets the long tail because it offers the lowest overhead per tenant and, thus, is suitable for a large number of small tenants, e. g. 1,000 tenants each having less than 50 MB of data and at most 5 concurrent users.

Reinwald (2010) in Elmore et al (2011 p2) came out with an improvement on the classification of database multi-tenancy models explored in the past by Jacob and Aulbach (2007), from the three to six classifications. Elmore et al (2011 p2) illustrate that these six classification were more improved to show different isolation level between tenants that are possible in all these models. This classification makes selection of a target model interesting and helpful.

Table 1: Multitenant database models, how tenants are isolated, and the corresponding cloud computing paradigms. (Elmore et al 2011 p2)

#	Sharing Mode	Isolation	IaaS	PaaS	SaaS
1	Shared Hardware	VM	✓	✓	
2	Shared VM	OS User		✓	
3	Shared OS	DB Instance		✓	
4	Shared Instance	Database		✓	
5	Shared database	Scheme		✓	
6	Shared table	Row		✓	✓

Elmore et al (2011) analysed the above table of classification by explaining how the six classifications can be collapsed into the traditional models of multi-tenancy. The models corresponding to rows 1–3 share resources at the level of the same machine with different levels of abstractions, i.e., sharing resources at the machine level using multiple Virtual Machines (VM Isolation) or sharing the VM by using different user accounts or different database installations (OS and DB Instance isolation). There is no database resource sharing. Rows 1–3 only share the machine resources and thus correspond to the shared machine model in the traditional classification. Based on Elmore et al (2011) the first three classifications can be collapsed into the traditional shared machine model since all share resources. Elmore et al (2011) also explains that the row 4 and 5 are involve sharing the database process at various isolation levels—from sharing only the installation binary (database isolation), to sharing the database resources such as the logging infrastructure, the buffer pool, etc. (schema isolation), to sharing the same schema and tables (table row level isolation). Rows 4–5 thus span the traditional classes of shared process (for rows 4 and 5). Finally Elmore et al (2011) also

explains that row 6 is the shared table model that uses a design which allows for extensible data models to be defined by a tenant with the actual data stored in single shared table. The design often utilizes 'pivot tables' to provide rich database functionality such as indexing and joins (Aulbach et al (2008) in Elmore et al 2011 p2).

Ni et al (2012 p2199) makes a brief comparison of two state-of-the-art approaches which are Independent Tables Shared In-stances (ITSI) and Shared Tables Shared Instances (STSI) to design schema. In this comparison ITSI has a poor scalability because there is need for maintenance of very large numbers of tables while STSI achieve very good scalability but with a very poor performance and high space.

In a multi-tenant setting, the degree of multi-tenancy becomes an additional factor that impacts performance, both for the overall system and the performance that is experienced by each individual tenant. In general, increasing the degree of multi-tenancy decreases per-tenant performance, but reduces the overall operating cost for the DaaS provider (Lang et al 2012 p702).

2.2 REQUIREMENT OF MULTI-TENANT DATABASES

Multi-tenancy can be applied at the database layer of a hosted service, in order to reduce the high cost of positioning and operating database. Jacob and Aulbach (2007 p2) said that in addition to managing customers' private data, a multi-tenant database could manage customer metadata and shared public data. These metadata should include data like contact information, their location and their features.

The target application for a hosted service will generally have a base schema that specifies all of its standard data. A multi-tenant database should maintain an instance of this base schema for each customer. The unified query language should ensure that Data Definition Language (DDL) statements for modifying the base schema and Data Manipulation Language (DML) statements for transforming existing data within it are applied to all customers in the farm, within the context of a rolling upgrade. The ability to perform such operations in bulk on the individual databases is essential to minimize downtime during an upgrade (Jacob and Aulbach 2007 p2).

2.3 CHALLENGES OF IMPLEMENTING MULTI-TENANT DATABASE

Ying et al (2011 p335) explained some of the challenges associated with multi-tenancy database development against the traditional database. The first challenge is the data isolation among tenants. Many tenants can share the same database, but the database must ensure the data of these tenants are isolated among each other and no one can get their data other than themselves. The second challenge is to achieve the economics of scales; the database must have the capability of on-demand scale to support large volume of tenants. This means that irrespective of growth in number of tenants and their demand on the database, it must be capable of meeting the demand. The third challenge is to be transparent to current existing application/skill, that is, the cloud developers can easily deploy the existing applications to on multi-tenant database without a large of code change, and the developers can create new multi-tenant application without using new technical knowledge. The fourth challenge is to support different isolations for the same application. This means that the use of different application by different tenant should ensure data isolation to each tenant regardless of the number of tenants involved.

These same constraint was also mentioned by Fang and Tong (2011 p95) that because of the peculiarity need of each tenant, there are problems of; 1, whether the database can afford the increase of both data and request accompanied with the growth of tenants. 2, how the database can meet the specific needs of one tenant efficiently and safely without affecting the others. It seems that the basic challenges associated with this technology since remain the same and different models or approaches were proposed to handle each of these challenges

2.4 DATABASE MIGRATION FOR ELASTICITY IN MULTI-TENANT DATABASE

Multi-tenant DBMSs often collocate multiple tenants' databases on a single server for effective resource sharing. Due to the variability in load, elastic load balancing of tenants' data is critical for performance and cost minimization. On demand migration of tenants' databases to distribute load on an elastic cluster of machines is a critical technology for elastic load balancing (Das et al 2010 p1). Multi-tenant database (MTD) must be able to handle the varying load requirement of each tenant on the database. This might require

adding or sometimes reducing resources depending on whether there is load increase or reduction.

Elasticity is the ability to adapt to varying loads by adding more resources when the load increases, or consolidating the system to lesser number of nodes as the load decreases; all in a live system without disruption in the service (Das et la 2010 p1). Elmore et al (2011 p301) explains that elasticity is the ability to scale up to deal with high load while scaling down in periods of low load. Therefore elastic load balancing is a supreme feature in the design of a modern database management system for a multi-tenant environment.

Efficient database migration in multi-tenant databases is an integral component to provide elastic load balancing. Furthermore, considering the scale of the system and the need to minimize the operational cost, the system should be autonomous in dealing with failures and varying load conditions. Migration should therefore be a first class notion in the system having the same stature as scalability, consistency, fault-tolerance, and functionality (Elmore et al 2011 p1). Database migration is a major factor which must be considered in effecting an efficient multi-tenant database.

3 FACTORS INFLUENCING THE CHOICE OF MULTI-TENANT DATABASE APPROACHES

There are several factors that influence the decision to adopt Multi-tenant database. These factors also help in determining the most suitable and appropriate approach of Multi-tenants database. The use of the system should be one of the influencing factors towards the decision. What is the area of usage of the system? Elmore et al (2011 p5) emphasise that the tenant application and usage requirements should be the primary consideration in deciding the right model of multi-tenant database. Sometimes users (tenants) are not equipped with necessary information about this before taking decision on what approach to adopt. Their decision is sometimes influenced by what vendors tell them. There is need to examine all these basic factors before approaching a service provider in order to make the right decision on this.

Some of these factors are itemized by Keemti (2010) as follows.

- Size of tenant database.
- Number of tenants.
- Number of users per tenant.
- Growth rate of tenants.

- Growth rate of tenant database.
- Security.
- Cost.

This research will examine each of the above factors as relevant e.g cost of infrastructure, management and development, and additionally look at:

- Flexibility – ability to create multiple tables by tenants.
- Time- time to build and configure.
- Regulatory consideration (UK/EU countries)

Some of these factors were actually mentioned by different scholars depending on the context of their research. It is possible to group them into four major headings as follows:

- Economic (Time and Cost) consideration.
- Growth consideration.
- Security consideration.
- Regulations consideration.

4 DISCUSSION

Factors identified above and further grouped into four will be used to discuss how they affect the decision to adopt and evaluate the multi-tenant database implementations. This discussion is based on the researcher's assumptions and also avenue for further research.

4.1 Economic Consideration.

The economy of any project has to do with the cost and the time it takes to complete the project. Cost and time are major factors to be considered under this heading. Cost is vital when considering the appropriate approach to be adopted in the implementation of database multi-tenancy. This cost is referred to as total cost of ownership (TCO) which is broken down into three major types. These are infrastructural cost, management cost and application development cost (Wang et al 2008 pp94-95). Infrastructural cost includes the cost of hardware, software and utilization costs. Management cost are cost related to the operational activities and processes like lifecycle management, monitoring data backup and restore; while application development cost are cost related to meeting each customer additional unique requirement (Wang et al 2008 pp94-95).

Each of the three implementation approaches has different TCO depending on infrastructures, management and applications that are involved. Pippal (2011 p213) explained that separate database approach requires a high cost of maintenance since each tenant has separate database and each has some metadata used to relate to the correct database for each tenant. Chong et al (2006) illustrate economic consideration using two approaches: shared approach and isolated approach. The cost of shared approach tends to require a larger development effort because of the relative complexity of developing a shared architecture hence very high initial cost but in the long run the operational cost will be very low because they can support more tenants per server.

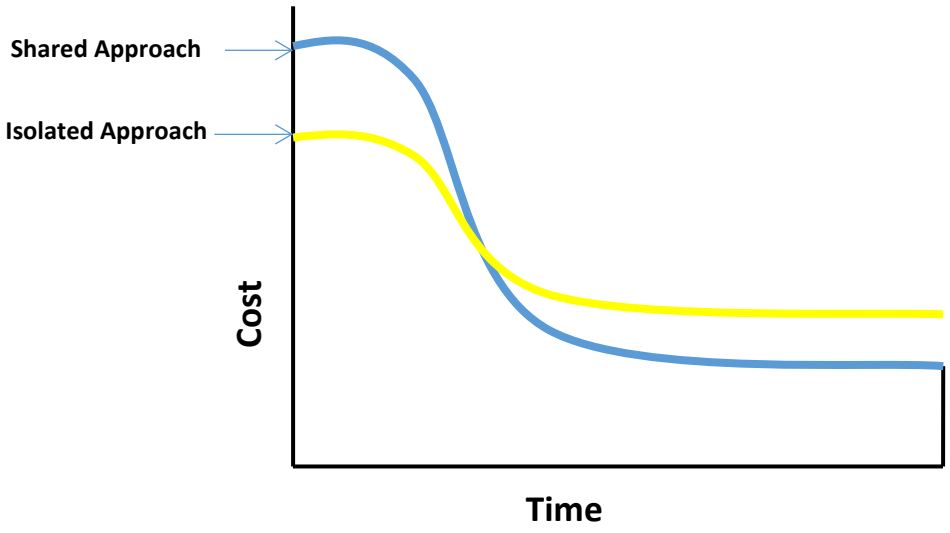

Figure 4 - Cost over time for a hypothetical pair of SaaS applications; one uses a more isolated approach, while the other uses a more shared approach.

Economic factors can constrain the development attempt which obviously can influence the choice of appropriate approach. And considering the time required to meet up with your customer (tenants) need, you might want to consider an approach that will require less time of development than a large-scale development approach. Aswathanarayana (2011 p4) also explain that cost saving is one of the major reasons for going into multi-tenancy in SaaS. And when talking about economical consideration we will always illustrate it in terms of cost saving or ROI (Return on investment) and time. Aswathanarayana (2011) argue that every application instance usually consumes a certain amount of overhead memory and processing power, which amount to a significant cost with many customers. This is reducing as many customers (tenants) share a single instance. Another aspect of cost benefit is the cost

associated with each tenant with regard to licensing cost, in multi-tenancy everything can be run on a single software instance. All tenants are now run on one single software and it means only one software license will be required.

The advantage of several company or tenant with each varying number of users sharing the same set of IT resources both hardware and software in multi-tenant database, this is fully utilized thereby reducing the capital expenditure (CAPEX) that each tenant would have incurred in the business. Aswathanarayana (2011) support this by saying that multi-tenant environment maximizes the sharing of resources by allowing load balancing among tenants and reducing the CAPEX. There is efficient use of these resources because of the load balancing the idle time is completely remove or reduced to a very barest minimum.

The cost of managing the software and hardware is greatly reduced in multi-tenancy since there is only one instance of the application is involved. Instead of caring out upgrade of software and hardware for each tenants involved. Only one off upgrade maintenance is done. Aswathanarayana (2011 p4) puts it by saying multi-tenancy greatly reduces operational complexity and cost in managing the software by simplifying the release management process since there is only one instance of the application. Same upgrades are common for all customers and the package typically need to be installed only on a single server. All forms of management and maintenance are done once in a single location. Gao et all (2011 p324) illustrate the reason behind the widespread of multi-tenancy one of which is the low cost for companies(tenants) in terms of hardware, software and utility of hosting centre (bandwidth, power, space etc.) and also lower cost of human resources to maintain processes and lifecycle via optimization and automation. The economic consideration in terms of cost is one factor so common in many scholars' prepositions. Aulbach et al (2011 p99) put it that the total cost of ownership (TCO) is reduced relative to the on-premise solutions. The operational expenditures is reduce based on the fact that fewer processes are required for management. This invariably leads to reduction in capital expenditures and as well there will be increase in the resource utilization in the long run.

Bezemer et al (2010 p1) explained this economic factor by saying that multi-tenant model has twofold benefit; application development to the users becomes easier for the service provider and also the utilization rate of the hardware can be improved as multiple tenants share same resource. Bezemer et al (2010) also illustrate that these two factors invariably reduce the overall cost of the application and makes multi-tenancy concept interesting for customers

who might have limited financial resource. The concept is more financial benefits to the SME businesses who might not have financial weight to have on-premises solution or system for their operation, thereby given them the opportunity to get same resources from a service provider on the basis of pay per use.

Myer (2007) in Schaffner et al (2012 p157) brought more light into this discussion with illustration on cost of hosting one instance EC2 on Amazon's Elastic Compute Cloud with MySQL database for one year. This shows that it is very high for a single tenant system to bear. This is expressly shown with the diagram below. This is an indication that the more consolidation the better in terms of cost. This another evidence to support the fact that the more tenants you have on a system the better it is for the tenants as this will drastically reduce the subscription expenses. The cost itemised here is excluding the cost involved in administering the database.

Table 2-Yearly Cost to Host RightNow Database on Amazon EC2 (Schaffner et al 2012 p157)

	Single Tenant	Multi-tenant
Standard on-demand pricing	$2,233,800.00	$148,920.00
One-year reserved pricing	$1,470,900.00	$98,060.00

4.2 Growth consideration.

In multi-tenant database growth is very vital factor that must be considered when choosing one of the implementation approaches you desire. The nature, number and need of the intended tenants expected to render services to will definitely affect the type of multi-tenant database model or architecture you will eventually adopt.

I have decided to group all points 1-5 of the above mentioned factors as part of growth consideration factors. Aulbach and Jacobs (2007pp3-4) carried out experiment on memory (storage) and disk usage of the five different databases. This experiment focused on the shared process approach. It shows that memory storage required to handle 10,000 schema instances vary from 79MB to 2,061MB across the five databases.

Table3 - Figure Storage requirements for schemas instances in megabyte (Aulbach and Jacobs 2007 p3)

	Memory 1 instance	Memory 10,000 instance	Disk 1 instance	Disk 10,000 instance
PostgreSQL	55	79	4	4,488
MaxDB	80	80	3	1,168
Commercial1	171	616	200	414,210
Commercial2	74	2,061	3	693
Commercial3	273	359	1	13,630

This shows that the size of tenant database in terms of storage capacity is a factor needed to be considered during the decision about choice of approach. How much storage space do you expect the average tenant's data to take? Schiller et al (2011 p118) give an illustration that the shared table approach has a promising for a service provider that has target for long tail because it offers the lowest over-head per tenant and thus suitable for a large number of small tenants. An example of 1000 tenants with each uses less than 50MB of data and at most 5 concurrent users. This is an indication that the number of tenants on the database and the number of users per tenant are all factors which must be thoroughly examined and also contribute to the performance of database system which invariably is also a factor to be considered when taking decision on what approach to be used.

The degree of tenancy is an influence to the approach to be adopted. How many prospective tenants is your system going to serve? The rate of growth of tenancy ie at what rate the tenant will increase? Though you might not be able to estimate the use with authority assign to them but you must be able to reason out in terms of the size. Is it going to be a database for hundreds of tenants, for thousand, ten thousand or more? Myer (2007) in Schaffner (2013 p1) reveal that already in October 2007, the SaaS CRM vendor RightNow had 3,000 tenants which are distributed across 200 MySQL database instances with 1-100 tenants per instance. This means that the CRM can evolve overtime. It's been designed to accommodate more tenants. The growth rate of tenants on it is never limited to a small number. Lang et al (2012 p702) buttress this point by referring to multi-tenant setting that the degree of multi-tenancy becomes an additional factor that impacts performance that is experienced by every tenant on it. Lang et al (2012) says that per-tenant performance decreases with an increase in the degree

of multi-tenancy but other way reduces the overall operating cost for the service provider. Two approaches were used by Lang et al to explain this which indicate the importance of growth. Chong et al (2006) explains with a simple diagram where the choice will tend to consider these factors. The larger you expect your tenant base to be the more likely you will want to consider a more shared approach (Chong et al 2006). When all or some the of your tenant want to store very large amount of data, it is better to consider the isolated or separate-database approach. And isolated approach is also good when there are a larger number of concurrent users (end users) per tenant (Chong et al 2006). The diagram illustrates with arrows pointing to the direction the decision is likely to go.

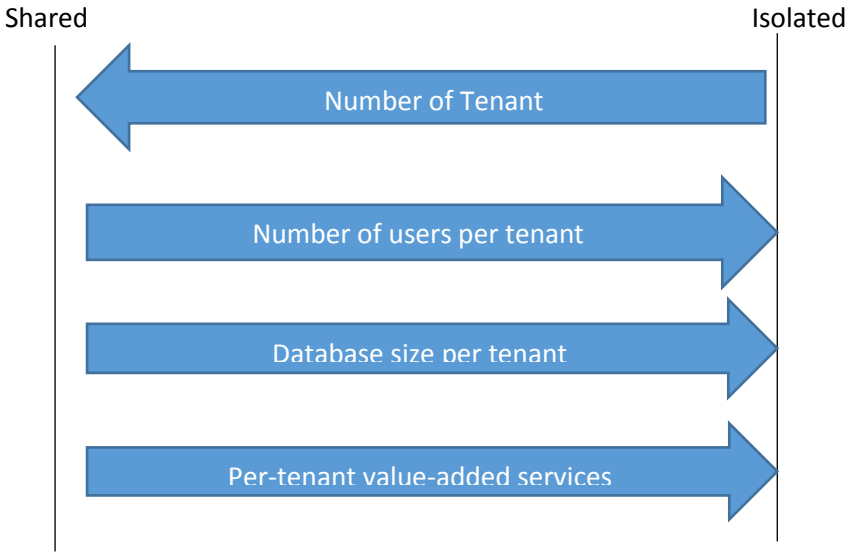

Figure 5 - Tenant-related factors and how they affect "isolated versus shared" data architecture decisions.

Tenancy growth was also mentioned by Aulbach et al (2008 p1196) that a tenant denotes an organisation with multiple users, up to 10 for a small to mid-sized business. A simple web application like business email, a single blade server will support up to 10,000 tenants. This shows that the magnitude of growth is high in terms of tenant number. Therefore, if there are 10 concurrent users in each tenant, this will amount to 100,000 users in the database using the system simultaneously. Chong et al mention per-tenant value-added services. These services include backup, restore, customization reconfiguration etc. which are services that are easier to provide in the isolated approach. These services can still be provided in shared environment but not as easy as in the isolated approach. In mid-sized enterprise application

like CRM, a blade server can support 100 tenants while a large cluster database can support up to 10,000 tenants (Aulbach et al 2008 p1196).

Most multi-tenant database systems have challenges in coping with the increase in number of tenants using the same logical schema. Zhou et al (2011 p335) came up with DB2MMT system, a DB2 massive multi-tenant database platform which provide case for multi-tenant application for long tail of tenants i.e very large number of small tenants using the same logical database schema. This also caters for consolidation of application or database for long tail of application with different schemas. This feature is an advantage for service providers who want to have very large number of tenant that would not require much memory space.

4.3 Security Consideration

In multi-tenant database system, one of the major concerns is the risk of data being exposed to the third parties. And because every database is design to store sensitive data, the prospective tenant will have very high security expectations. Every service provider will always want to operate to much higher security standard but sometimes this might not be to a hundred per cent. Therefore the service level agreement (SLAs) will have to provide very strong data security guarantees.

Some of the security issues related to multi-tenancy database include data isolation among the tenants. Gao et al (2011 p324) mention some of the challenges facing the ISVs (Independent Service Vendor/Provider) in delivering their service these include the data security isolation among tenant, the different tenants having different SLA demand, customization requirement and effective database scale out mechanism as the number of tenant increases. Hui et al (2009 p832) itemizes the problems faced in providing database as a service which includes security, contention for shared resources and extensibility. Hui et al (2009) also mentioned scalability as one of the problem which was defined as the ability to serve an increasing number of tenants without too much query performance degradation. In spite of the increase in the tenancy and query request, the system should still be able maintain its performance level. This tenancy growth should not impact the service availability of other tenants.

Another major security issue is the flexibility of the system. A Multi-tenant database must be able to serve hundreds and thousands of tenants through one instance. Aulbach et al (2009

p881) expand this on by saying that a multi-tenant database must be flexible by been able to extend the base schema to support multiple specialised version of application and to dynamically evolve the base schema and its extension while the database is on line. This means that the evolution of one tenant-owned extension should be independent on the service provider. The tenant should be able to achieve this without the service provider. The creation of multiple tables by individual tenant must be guaranteed. This is not most times possible because each tenant has its own system requirement such as objects attribute business logic etc (Gao et al 2009 p325)

Guo et al (2007 p2) also highlights two major challenges of native multi-tenancy design, firstly the system should support application level isolation among different tenants. This is quite in line with other scholars' positions. The multi-tenancy database should support more tenants and efforts are made to prevent the quality of service of one tenant from being affected by other tenants. While the processes and operation request of one tenant increases should not be a deterrent to other tenant on the platform. Secondly, Guo et al (2007) bring to notice that in this type of environment, the customization for one tenant should not impact on the other tenant during the runtime. This was also pointed by Vashistha and Ahmed (2012 p49) that Isolation should be carefully considered in almost all part of architecture design from both functional and non-functional level such as security, performance, availability, administration and also support tenant customization of their own services in runtime without impacting on others. Customization effort for one tenant which might require code modification and applications re-deployment, this sometimes affects other tenant. Once this becomes frequent occurrence will bring about the issue called service availability issues (Guo et al 2007 p2). The ISP will definitely have scores to settle with the customers (tenants) and will automatically results in termination of contract and withdrawer of service from such ISV.

Gao et al (2011 p325) gave a major requirement of multi-tenant database architecture as a system which is able to scale very large to support very large customer volume. This also Gao et al called incremental scalability, scale-out without impacting the service availability of other tenants. This particular feature is very difficult to achieve in most model types. This is also subject to the level of isolation in the approach adopted. Some service providers of multi-tenant database will have to put a lot of expertise and resources into making sure its infrastructure is as secure as possible against any form of data exposure, which would be very bad for its reputation.

One other security issue with this concept is that when one tenant has a particular application, it should be possible to deploy this application to other tenant in such system. This is a challenge mentioned by Zhou et al (2011 p335) as the cloud developer can easily deploy the existing application on to multi-tenant database without a large code change. When there is much code changes it affect other tenants' processes and subsequently lead to system down time.

Pippal et al (2011 p213) list out some concerns about third party offering multi-tenant service. One of which is the provision of a secure framework for authentication and authorization within the system. Pippal et al developed a protocol called Kerberos protocol. This incorporates a standard framework to implement authentication and authorization in the use of a multi-tenant database. This confirms who can access the database and verifies what you are allowed to do. This is to ensure what type of operation you are permitted to carry on the system. Authorization occurs after a successful authentication process. Every connection, login, query and update on a database must go through these two major security processes. These two security measures help find out why connection attempts are either accepted or denied. Authentication is important part of identity management while authorization has to do with access management. These are major mechanisms that must be enforced in a multi-tenant database for security purposes.

4.4 Regulatory Consideration.

The database of every organisation is very vital to the growth and progress of the organisation. Enormous time and resources are devoted to the safety and protection of databases even as information is very important to organisations' processes. There are laws and regulations put in place by different governments that serve as protection to databases of different entities that operate in that geographical location. Companies, organisation and governments are often subject to regulatory laws that affect their security and even record storage needs. The knowledge of these different laws and regulations is also to be considered when taking the decision about multi-tenant database. Chong et al (2006) argue that the investigation of regulatory environments that your prospective customers occupy in the market in which you expect to operate is important. This shows that you have to be conversant with the laws that operate in that area or country. It is important to find out whether there are any aspect of the law that present any condition that will influence your decision toward given your database service to a third party.

In 1996, the European Union (EU) finally adopted the EU Database Directive (Directive). The Directive created a two-tier protection scheme for electronic and non-electronic databases. Member states are required to protect databases by copyright as intellectual creations, or to provide a novel *sui generis* right to prevent unauthorized extraction or re-utilization of the contents of a database (Grosheide 2002 p39). The *sui generis* database right refer to as an intellectual property right that provides protection for the contents of a database (Schellekens 2011 p620). The difference between the two is that copyright infringement implies copying the structure, while the *sui generis* right infringement implies copying the contents themselves, irrespective of their copyright ability (Grosheide 2002 p39). This shows that provision is made for the protection of the content and also the structure of a particular database. The most important aspect of this law is the sui generis law which protect the content of the database.

Davison (2003 p10) explains these provisions of database protection with three different models. These models are; 1) Copyright protection is provided at a low level of originality, this means that a database user cannot take a substantial amount of the data contained within the database (Davison 2003 p10). There are prohibitions supported by law and regulation to those who might want to copy part of a database. 2) Copyright protection is provided if there is some creativity in the selection or arrangement of the database material which is back up with *sui generis* right (Davison 2003 p10). This is to say that prohibitions are placed on unauthorised extraction or re-utilisation of a substantial part of the data. This actually confers the exclusive property right to the database owner. 3) Copyright protection is provided for the creativity in the selection or arrangement of the database material (Davison 2003 p10). These models help in the protection of database in terms of security. Even when an unauthorised someone gained access, these protections put in place by EU helps protect the database to some extent.

There are challenges when tenants are from different regulatory authority. Most of the countries have different laws which sometimes are not having same conditions and to now manage these conflicts will pose a great technical and economical challenge to the ISVs. One of the reason given by Johnson and Grandison (2007 p256) for privacy breaches and identity theft is the fact that there is weak and ineffective enforcement of the data protection laws as well as discrepancies and conflicts in the legal protections.

5. CONCLUSION

In this paper, we show that multi-tenant database helps reduce the cumulative cost that will be incurred by organizations on the development and deployment of an on premises database system. We show that economic factors can constrain the development attempt which obviously can influence the choice of appropriate approach of multi-tenant database models. One need to take into consideration the growth rate of your database which include the number of tenants, the number of users per tenant, size of each tenant database, growth rate of tenants and lastly the growth rate of tenant database. We have also show different security issues related to multi-tenant database which include data isolation, scalability, flexibility and customization. Effort must be made to look into the regulatory provisions for prospective tenants. The regulatory laws of all the tenants on a particular multi-tenanted database must be the same in order to have harmony in the laws that govern the system.

REFERENCES

1 Aulbach, S., Grust, T., Jacobs, D., Kemper, A. and Rittinger, J. (2008) Multi-tenant databases for software as a service: schema-mapping techniques *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* [online]. Vancouver, Canada New York, NY, USA: ACM, pp.1195-1206.

2. Aulbach, S., Jacobs, D., Kemper, A. and Seibold, M. (2009) A comparison of flexible schemas for software as a service *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data.* [online]. Providence, Rhode Island, USA New York, NY, USA: ACM, pp.881-888.
3. Aulbach, S., Seibold, M., Jacobs, D. and Kemper, A. (2011) Extensibility and Data Sharing in evolving multi-tenant databases *Data Engineering (ICDE), 2011 IEEE 27th International Conference on.* [online]. pp.99-110.
4. Aswathanarayana, H. (2011) Multi-tenant enabling a single-tenant application: *Wipro technologies* [online]. Bangalore: [Accessed 14 March 2013]. Available at http://www.wipro.com
5. Banville, R. and Holzel, R. (2012) Openedge multi-tenancy overview: *Progress software* [online]. Bedford: [Accessed 23 January 2013]. Available at < http://www.progress.com>
6. Bezemer, C., Zaidman, A., Platzbeecker, B., Hurkmans, T. and Hart, A. (2010) Enabling multi-tenancy: An industrial experience report *Software Maintenance (ICSM), 2010 IEEE International Conference on.* [online]. pp.1-8.
7. Chong, R., Carraro, G., Wolter, R. (2006) *Multi-tenant Data Architecture* [online]. [Accessed 23 August 2013]. Available at :<http://www.msdn.microsoft.com/en-us/library/aa479086.aspx#mlttntda_topic3>
8. Das, S., Nishimura, S., Agrawal, D. and El Abbadi, A. (2010) *Live database migration for elasticity in a multitenant database for cloud platforms* [online].

9. Davison, M. (2003) The legal protection of databases, Cambridge University Press, Cambridge.
10. Elmore, A. J., Das, S., Agrawal, D. and Abbadi, A. (2011) Towards an elastic and autonomic multitenant database *Proc. of NetDB Workshop.* [online].
11. Elmore, A. J., Das, S., Agrawal, D. and El Abbadi, A. (2011) Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data.* [online]. Athens, Greece New York, NY, USA: ACM, pp.301-312.
12. Fang, S. and Tong, Q. (2011) A comparison of multi-tenant data storage solutions for Software-as-a-Service *Computer Science and Education (ICCSE), 2011 6th International Conference on.* [online]. pp.95-98.
13. Gao, B., An, W., Sun, X., Wang, W., Fan, L., Guo, C et al. (2011) A Non-intrusive Multi-tenant Database Software for Large Scale SaaS Application *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on.* [online]. pp.324-328.
14. Grosheide, F.W. (2002) Database Protection--The European Way. *Wash.UJL and Pol'y* [online]**, 8**pp. 39 .
15. Grund, M., Schapranow, M., Krueger, J., Schaffner, J. and Bog, A. (2008) Shared Table Access Pattern Analysis for Multi-Tenant Applications *Advanced Management of Information for Globalized Enterprises, 2008. AMIGE 2008. IEEE Symposium on.* [online]. pp.1-5.
16. Guo, C., Sun, W., Huang, Y., Wang, Z. and Bo Gao. (2007) A Framework for Native Multi-Tenancy Application Development and Management *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on.* [online]. pp.551-558.
17. Hui, M., Jiang, D., Li, G. and Zhou, Y. (2009) Supporting Database Applications as a Service *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on.* [online]. pp.832-843.
18. Jacobs, D. and Aulbach, S. (2007) Ruminations on multi-tenant databases. *BTW Proceedings* [online]**, 103**pp. 514-521 .
19. Johnson, C. and Grandison, T. (2007) *Compliance with data protection laws using Hippocratic Database active enforcement and auditing* [online].
20. Keemti, P. (2010) *Multi-tenant Database Architecture* [online]. [Accessed 23 August 2013]. Available at <http://www.msdn.microsoft.com/eus/library/aa479086.aspx#mlttntda_topic1>
21. Lang, W., Shankar, S., Patel, J. M. and Kalhan, A. (2012) Towards Multi-tenant Performance SLOs *Data Engineering (ICDE), 2012 IEEE 28th International Conference on.* [online]. pp.702-713.
22. Ni, J., Li, G., Zhang, J., Li, L. and Feng, J. (2012) Adapt: adaptive database schema design for multi-tenant applications *Proceedings of the 21st ACM international conference on Information and knowledge management.* [online]. Maui, Hawaii, USA New York, NY, USA: ACM, pp.2199-2203.
23. Pippal, S., Sharma, V., Mishra, S. and Kushwaha, D. (2011) Secure and efficient multitenant database for an ad hoc cloud *Securing Services on the Cloud (IWSSC), 2011 1st International Workshop on.* [online]. pp.46-50.
24. Schaffner, J. (2013), Multi tenancy for cloud-based in-memory column databases: workload management and data placement, Springer, Wien

25. Schaffner, J., Jacobs, D., Kraska, T. and Plattner, H. (2012) The Multi-Tenant Data Placement Problem *DBKDA 2012, The Fourth International Conference on Advances in Databases, Knowledge, and Data Applications.* [online]. pp.157-162.
26. Schellekens, M. (2011) A database right in search results? – An intellectual property right reconsidered in respect of computer generated databases. *Computer Law and Security Review* [online]**,** **27**(6), pp. 620-629 Available at:<http://www.sciencedirect.com>
27. Schiller, O., Schiller, B., Brodt, A. and Mitschang, B. (2011) Native support of multi-tenancy in RDBMS for software as a service *Proceedings of the 14th International Conference on Extending Database Technology.* [online]. Uppsala, Sweden New York, NY, USA: ACM, pp.117-128.
28. Vashistha, A and Ahmed, P. (2012)"SaaS Multi-Tenancy Isolation Testing-Challenges and Issues", *International Journal of Soft Computing and Engineering*,[online] **2**(5), pp. 49-50 pp. 49-50 Available at:<http://wlv.summon.serialssolutions.com>
29. Walraven, S., Van Landuyt, D., Truyen, E., Handekyn, K. and Joosen, W.(2014) Efficient customization of multi-tenant Software-as-a-Service applications with service lines. *Journal of Systems and Software* [online] (0), Available at:<http://www.sciencedirect.com/science>.
30. Wang, Z., Guo, C., Gao, B., Sun, W., Zhang, Z and An, W. (2008) A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing *e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on.* [online]. pp.94-101.
31. Ying, H., Wang, Q., Wang, Z., and Wang, N. (2011) DB2MMT: A Massive Multi-tenant Database Platform for Cloud Computing *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on.* [online]. pp.335-340.
32. Zhou, Y., Wang, Q., Wang, Z and Wang, N. (2011) DB2MMT: A Massive Multi-tenant Database Platform for Cloud Computing *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on.* [online]. pp.335-340.