International Journal of Grid and Distributed Computing
Vol.6, No.5 (2013), pp.29-38
http://dx.doi.org/10.14257/ijgdc.2013.6.5.03

Implementing a Storage Pattern in the OR Mapping Framework

Muhammad Naeem Ahmed Khan, Arsalan Shahid and Sarah Shafqat

Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST), Islamabad, Pakistan

mnak2010@gmail.com, drarsalan@gmail.com, sarah.shafqat@gmail.com

Abstract

Heading towards the next decade, a major paradigm shift has been observed in the way the software services are being provided to the enterprises and corporate sector. Corporations and enterprises are switching to www host applications being offered as a service by software vendors and on-premises LOB (Line of Business) applications are taking a toll back. SaaS (Software as a Service) is the new concept. Adapting of SaaS, however, requires that the applications which are being provided as a service should be generalized for users or groups of users and it would require a vast space to be allocated to user or user group. The users or user groups ordinarily correspond to a company or group of companies/businesses and are termed as tenants. In this regard, the architecture of SaaS applications needs to be customized to support certain characteristics — e.g., configurability, maintainability and scalability — to support high storage for hosting resources made available to diverse number of users. This paper, firstly, analyzes new trends in the present day business environment alongside the hardware and software industry that led to the development of SaaS model; and then looks into the characteristics and features that a storage pattern for multi tenant system in SaaS needs to possess in order to put this concept into practice.

Keywords: *Cloud Computing, SaaS, Storage Pattern, Single Instance Multi-tenancy, Data Customization*

1. Introduction

The SaaS has altered the way people build, sell, buy and use software as it is provided as a hosted service by the sellers and is bought by the clients on subscription basis. This is a major paradigm shift in the software industry because the companies do not need to buy and maintain their own IT infrastructure as the service providers maintain the software service and infrastructure at their ends. The users who utilize SaaS are called tenants; and since the same software service can be subscribed by multiple users assigned a certain space, therefore, the concept of multi-tenancy is at the core of SaaS. With configurability, efficiency and scalability being three main attributes of SaaS in addition to multi-tenancy [1], storage space that is being allocated to the tenant comes foremost. Multi-tenancy necessitates novel architectural change in order to create software that is capable to support multiple users or tenants simultaneously. To understand the multi-tenant storage pattern being adopted by SaaS firstly, there is need to understand Multi-tenant application. Multi-tenant application should be differentiated from a multi-user application [3] as applications' customization for each user is an integral part of the former technique to avoid sharing of data among the tenants. Likewise, it also differs from the multi-instance applications since all the tenants use a single instance of the application while a separate application instance for each user is required in the multi-instance application model. Multi-tenancy, therefore, requires a strict separation of data among the tenants to accommodate diverse set of requirements of different tenants over the base functionality of the same application. As a result, security, extensibility, configurability and scalability attributes bear paramount importance in SaaS architecture, particularly, if a large user base is required to be supported. The two main benefits of multi-tenant applications include supporting a single application instance and maximizing the rate of hardware utilization.

Traditionally, the users purchase a license of the software product to install its copy on their system and are bound to adhere to the terms and conditions spelled out in the SLA or license agreement. The recent advancement in hardware, software, virtualization and Internet technologies have led to the foundation and formation of an environment through which software services can be provided to the clients at a much lower cost of ownership on subscription basis.

As end users are much privy to their corporate information and business secrets, therefore, there is a prevalent general reluctance in the adoption of SaaS — as the companies hesitate to store their corporate and business related information with a third party. This situation demands for developing such policies for SaaS platform that not only ensure safety and security of the clients' data but also guarantee its privacy to restore confidence/trust of clients. For this purpose, applications need to be designed in such a way that data of one tenant is not at all visible to any other tenant.

1.1. Types of Multi-Tenant Storage Pattern

There are three main types of multi-tenancy which primarily depend on the way how data is isolated for individual tenants; it is more of a continuum between totally isolated and shared data [2, 9] where data storage architecture is essentially dependent on the technical and business considerations. In a broader spectrum, the three types of multi-tenant storage based on data architectures are: (i) separate databases for each tenant, (ii) same database but separate schema for each tenant, and (iii) same database, schema and tables for all the tenants with separate tenant ID (pure multi-tenancy).

a. *Separate database approach*: In this approach, code and the user interface are the same but each tenant has its own database with distinctive structure. This the simplest form of multi-tenancy and can be used to shape an application into a multi-tenant application, however, it bears a higher operational cost.

b. *Shared Database*, *Separate Schemas*: In this approach, all the tenants share the same database but have different schemas. The schema creation, with its accompanying set of tables, needs to be dynamic and is created when a tenant subscribes to the service. This approach is relative easy to implement and data model extension is straightforward, but the backup and restore procedures become complicated since data of all the tenants needs to be backed-up and restored in case of system failure.

c. *Shared Database, Shared Schema (Pure Multi-tenancy)*: This approach, also called pure multi-tenancy, entails using the same schema to support multiple tenants [2, 10, 11] and data of each tenant is tagged with a particular tenant ID. This approach endures the least hardware cost as it uses the least amount of resources, but on the other hand, it bears higher level of complexity and higher cost of development. The rest of the paper only discusses this approach as other approaches are beyond the scope of this study.

1.2. Key characteristics of Multi-Tenant Storage

Multi-tenancy provides a much better hardware sharing than virtualization. The main characteristics of multi-tenancy include hardware resource sharing and high degree of configurability. The hardware resource sharing characteristic facilitates a lower TCO (Total Cost of Ownership) as the multi-tenant applications allow for placing multiple tenants on the same server, thus letting tenants to share the hardware and software resources.

The more appropriate approach to facilitate configurability is to allow mechanisms built into the application to achieve customization for separation of data of multiple tenants — for instance, user interface customizations, workflow creation and custom field creation.

a. Enabling Multi-Tenancy at Data Layer: The main challenge in multi-tenancy is to add tenant information into the data in a way that every piece of information can be precisely traced back to an individual tenant. Once this has been achieved, there are certain patterns and practices related to security, resource isolation, extensibility and scalability that need to be followed to enable the key characteristics of multi-tenant enabled applications [2, 5]. Security patterns in multi-tenancy fundamentally pertain to filtering, permission and access control and encryption [2, 11]. Filtering ensures that the data specific to a particular tenant is visible only to that tenant and it is accomplished by adding a layer between the actual data and the data source. User permissions and privileges are managed through maintaining Access Control List (ACL) that characterizes users' accesses to data. In multi-tenancy context, this is achieved using two fundamental techniques — impersonation and trusted subsystem — in which the former technique require that only the actual tenant's credentials are used to access data and the latter technique only uses the tenant's credentials for authentication purposes and necessitates a separate account to access the system. Additionally, encryption is applied to the sensitive data to further enhance data security measures.

b. Extensibility Patterns: The ability to store per tenant information by extending the schema is a valuable attribute for any multi-tenant application. For this purpose, the following three patterns are mostly used.

   i) Pre-Allocated Fields: This implies introducing fixed number of columns in a table to store information of tenants [2, 13]. Usually, string data is stored in these fields and necessary metadata tables are created accordingly that contain pertinent information about processing these fields. There could be single metadata table that encompasses details of all the custom fields or there could be a separate metadata table for each custom field [2, 10].

   ii) Name Value Pairs: This approach requires two additional tables along with the primary data table. First table contains information of the name value pair field and the second holds definition of the custom field. Whenever a new field is created, its metadata is written into the metadata table. When the tenant stores data into the relevant fields, then firstly the primary data is stored into the tables and an ID is generated for the corresponding primary item. This approach introduces an arbitrary number of extra fields into the data model without changing the shared schema and retains the cost effectiveness of the shared database, but along the side, it adds a layer of complexity to the data retrieval process.

iii) XML Based Extension Model: XML support, when integrated with the relational database, adjoins data extension capabilities at the client side without changing the schema of the tables [7, 12]. Some database management systems, such as Oracle, allow defining explicit XML fields and embedding XPath queries in the SQL's "select" statement. The key advantage of this approach is that it does not require changing the database schema while adding new fields/columns into the tables.

c. Scalability Patterns: The scalability patterns facilitate expanding the number of tenants, users and data volumes in the multi-tenant application. Scaling can pertain either to moving databases to a more powerful server (scale up) or increasing the total number of servers and sharing the load between these servers using load balancing techniques (scale out). For scale out mode, either replication or partitioning techniques could be used. Replication, as the name suggests, sets up mirror image of the same database server onto another server; whereas, partitioning means dividing data of a single table and coping it into separate tables in the same or different database. The tenant based horizontal partitioning (*i.e.*, dividing tables by rows) could be useful in these situations where many users try to access the data or larger data size is causing delays in fetching query results, or operational maintenance tasks start affecting the availability of data. The simplest approach to horizontally partition the database is to apply the tenant ID.

2. Literature Review

To enable multi-tenant storage pattern in SOA, a considerable amount of research has been carried out and reported in the literature. Most of these approaches have focused on defining architecture at the data storage level. Jacobs and Aulbach [9] studied the currently available relational databases and concluded that there was no intrinsic support available for multi-tenancy. Grund et al. [10] suggest that the RDBMS (initially devised by Codd [16]) and its ISO implementation do not provide the semantics to support multi-tenancy natively. The proposed solution to this could be to introduce a tenant context in a multi-tenant system and employ partitioning strategies based on this context.

Jacbos and Aulbach [9] discuss mapping of the tenant context or tenant identifier into the existing patterns of database semantics to provide data isolation on per tenant basis. This essentially necessitates introducing an administrative framework into the database schema that should maintain customer related metadata. Additionally, the framework should allow for the user management, access control, disk quota management that duly supports the execution of bulk administrative operations.

Chong and Wolter [1, 2] discussed various patterns that can be used at the database level to implement multi-tenancy. The drawback of this approach is the requirement to write code for all the tables that the tenants use to share information.

Bezmer [3, 4] discussed the various types of multi-tenancy and the role that multi-tenancy could play in providing affordable solutions to the small to medium level client sides. The identified multi-tenancy challenges include: performance, scalability, security, zero down-time and maintenance. Data isolation is also identified as the basic requirements for multi-tenancy and researchers proposed using a layer between the business logic and the database layer that should be responsible for creation of new tenants, query adaptation and load balancing. However, such architecture bears a down side that it requires manual creation of layer that allows a single tenant application to be converted to a multi-tenant application.

Guo [6] proposed a framework to support multi-tenancy and advocated introducing a "multi-tenancy enablement" layer between the business logic and the data layer to make it transparent for the developers to write multi-tenant applications. Data management and information isolation are identified as one of the key characteristics of the framework. For data isolation, SQL 'where' command can be applied to filter the data on a per client basis.

While discussing multi-tenancy reengineering patterns, Bezmer [17] identifies database as one of the three areas which need to reengineer in order to enable multi-tenancy in any application. According to Bezmer [17], commercial off-the-shelf databases are not intrinsically equipped to handle multi-tenancy. Therefore, this logic must be introduced in a multi-tenant application by introducing an additional layer between the business and the data layers. The main task of this layer, in terms of data isolation, should be query adaptation. This layer should make sure that all queries are articulated in a manner that each tenant should only be able to access its own data.

Du *et al.,* [7] discussed harnessing the native XML support in the current RDBMS systems in scheme extension on a per tenant basis as well as the implementation of read and write functionality using these patterns. It is concluded that the XML based model are more efficient than the custom fields and pre-allocated fields patterns. Nonetheless, such an approach requires that code on a per table basis be written to make this functionality work.

3. Problem Statement

While considering the challenges of implementing multi-tenant storage pattern for applications, the first requirement that is central to the architecture is data isolation on a per tenant basis. In this regard, a number of valid and efficient patterns have been defined in the literature that facilitate proper data isolation. These patterns rely on filtering data on a per tenant basis. The problem with these approaches is that they introduce considerable complexity in both the business and the data layers. These approaches require that the developer should write code bearing considerable complexity to make sure that data isolation is maintained.

Real world business entities are represented as objects of classes in the present day object oriented systems. In fact, this is how domain models get converted to physical design. The next step is the information persistence. In this step, the information in the business objects is stored into the database. What is required is that the information should be serialized to the database. Traditionally, the process followed in the development of writing a complete data access layer is to store the information into the database by executing SQL queries or stored-procedures on the RDBMS. For this purpose, a separate query needs to be written for each of the CRUD (Create, Read, Update and Delete) operation. In short, one can summarize the problem into the following points:

- There is a need to restructure the complete Data Access Layer to endure data to the databases.
- Data filtration is required to support multi-tenancy, which adds considerable complexity to the queries resulting in additional work for the developers.
- This layer needs to be written for each and every database table and view that represent multi-tenant data.

Keeping into consideration the aforementioned identified problems, we propose a Storage Pattern in O/R mapping framework in the next section that offers a potential solution to these problems.

4. Proposed Approach

One of the foremost objectives of this study is targeted towards freeing the developer of the complexity of writing multi-tenant aware data layer. To simplify this process, a number of Object-Relational Mapping (O/R Mapping) frameworks have been proposed in the literature. These frameworks simplify the process of data layer development by providing alternate, simpler mechanisms to specify information that needs to be persevered into the database management systems. These frameworks do rely on the creation of queries to fetch and update data in the database systems and all such operations are performed internally. These frameworks do not require the developer to write such queries; however, they do require that some rules should be followed while specifying the information that needs to be preserved. Some of these frameworks like Hibernate (Java) and NHibernate (.Net) provide the extension mechanism that allows the query generation pipeline to be extended so that custom processing could be done on the queries before they are a executed on the database. Our proposed storage pattern for O/R mapping framework is shown in Figure 1.

The proposed pattern could be extended to make it multi-tenant aware. The following modifications would be required in the existing storage pattern in framework to make it multi-tenant aware.

- Internal creation and management of tables to store the tenant information.
- Creation of a tenant context object that contains information about the tenant to whom the current user belongs to.
- Extension of the frameworks execution pathway to modify the queries that are tenant specific.

The key components and features of the proposed pattern in the framework are described in the subsequent paragraphs.

4.1. Creation and management of tenant specific tables

In the first place, the data layer in the framework will initialize the database tables which are required to store tenant specific information including groups and users. The framework will also initialize these tables with a special type of group known as super user group and any user belonging to this group will be used to create and manage other tenants, groups and users. However, super user group will itself be treated as read-only by the framework so that there is always one user available to manage other users of the framework.

4.2. Tenant Context

The storage pattern embedded on the data layer provides a special programmatic structure that contains the current tenant ID and information. It also contains methods that allow users and group management. The current tenant information is determined by the currently logged-in user and will be used to fetch tenant information from the database. This object contains a single static factory method that can be invoked to create a new tenant context object so that the information is fetched only on demand.

4.3. Extension to Query Execution Pathway

The storage pattern uses the extension mechanism defined in the underlying framework. It intercepts the currently executing query and adds the tenant specific filter to every executing query so that only data integrity is maintained. This extension could be due to a direct change in the code of an existing framework or it could use an existing extension mechanism provided by the framework.

Figure 1 demonstrates a typical multi-tier application utilizing an O/R storage pattern on the framework. The application layer consists of the application business logic. For an object oriented application, this consists of classes representing the business objects in the transient state. The transient objects are sent to the O/R mapping framework, which does the job of storing and retrieving them from the database.

The query generation engine is at the heart of any such O/R framework. This engine is responsible for determining the schema of the tables that store a certain object. Some frameworks, like Hibernate, provide an extension mechanism through events that allow the modification of the query which is executed on the tables. For the frameworks that do not support extension mechanisms, the actual source code will need to be modified to make the O/R mapping engine tenant-aware.

The data retrieval process is at the heart of our proposed data layer in the O/R framework. All data retrieval requests are routed to the O/R mapping engine which generates the data retrieval query. This query is then routed to the extension mechanism which requests the "Tenant Context" object for the relevant Tenant ID that is determined using the current logged in user-id. The returned tenant ID is used to modify the query to add a SQL "*where*" clause so that data is filtered according to the current tenant. This mechanism allows seamless data isolation and requires no additional effort on the part of the developer.

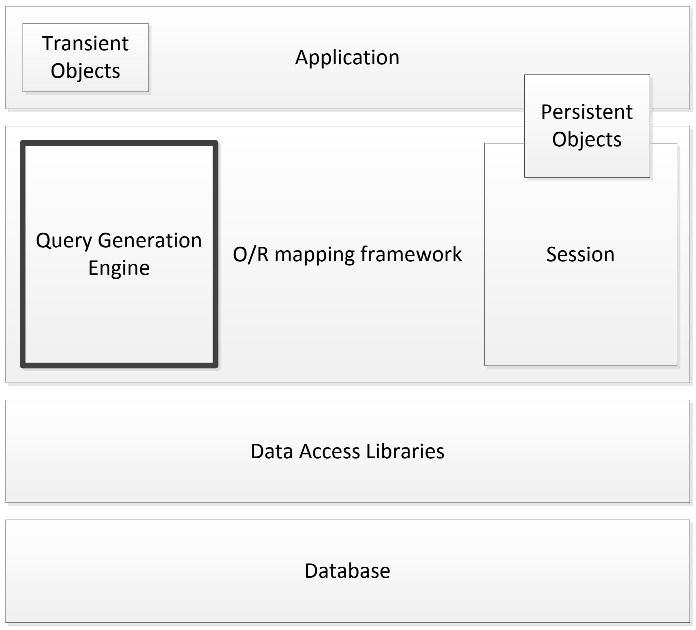

Figure 1. The Data Layer in O/R mapping framework stack

5. Discussion

We have used comparison of our proposed pattern with the existing multi-tenancy techniques for validation. A comparative analysis of our model and the existing models is shown in Table 1.

The proposed system simplifies pure multi-tenant application development by introducing intrinsic support for multi-tenancy. The benefit of using this storage pattern in the framework is two folds. It allows the use of an existing O/R mapping framework thus making the development of the data layer redundant. Moreover, it has intrinsic mechanism to enforce data isolation thus eliminating the need to write any specific code that is needed to make sure that the data is properly isolated on a per tenant basis.

This framework would act like a replacement to the data layer for any existing application and will be the first step towards making any existing application multi-tenant. For those applications that are already using the unmodified version of this O/R mapping framework, our framework could act like a drop-in replacement in making the application multi-tenant aware.

A number of frameworks have been proposed to enable multi-tenancy; though some of them do exhibit real premise, but none of them attempts to use an O/R mapping framework to achieve the goal of multi-tenancy. All of these frameworks require that their stipulations be applied and recreated on a per application basis to make them multi-tenant. Our framework is an endeavor to propose a *write once use anywhere* logic that can be used and enhanced in future multi-tenant aware applications.

Table 1. Comparative Summary of Existing Multi-tenancy and our Proposed OR Mapping Technique

Existing Systems	Snags	Our Proposed System	Expected Results
Multi-User System	No facility to support multiple tenants using pure multi-tenancy.	A system with pure multi-tenancy support.	A system that supports multiple tenants.
Multi-Instance Multi-Tenant Applications	Poor usage and sharing of hardware resources.	Promotes use of pure multi-tenancy.	Better usage and sharing of hardware resources.
Pure multi-tenant systems utilizing O/R mapping framework.	Extra instrumentation required at the object level to tag them with the tenant ID.	Auto tagging and filtering of tenant data with no additional development effort.	No need for additional instrumentation to make the system multi-tenant.
Pure Multi-Tenant systems	Need to write data layer and complex queries to enable data isolation.	Pure multi-tenant system with modified O/R mapping data management layer.	No need to write data layer. Seamless data isolation.

6. Conclusion

This paper is an endeavor to introduce multi-tenancy storage pattern to an existing O/R mapping framework. This research can have different off-shoots as there are a lot of areas that can be explored. This framework applies tenant specific filtration to the data and can be extended to provide configurability on per tenant basis. The framework could be adapted to use any of the techniques discussed earlier like pre-allocated fields, name-value pairs and XML-based extensions to allow configurability to the solutions.

One of the possible areas of further research could be implementation of one of the storage techniques and patterns which have been identified in the previous work. The framework could be directed to use one of the storage techniques like Pre-allocated fields, key value pairs or the XML fields to store tenant specific information. Implementation of any such technique will require considerable changes in any O/R mapping framework since this would require the data serialization mechanism to be rewritten.

Another future direction could be integrating the storage mechanism to the user interface of a multi-tenant application to provide configurability with the application.

References

[1] F. Chong, “Architecture Strategies for Catching the Long Tail”, MSDN, **(2006)**.
[2] F. Chong and R. Wolter, “Multi-Tenant Data Architecture”, MSDN, **(2006)**.
[3] C. P. Bezemer and A. Zaidman, “Multi-tenant SaaS applications: maintenance dream or nightmare?”, Proceedings of the 4th International Joint ERCIM/IWPSE Symposium on Software Evolution (IWPSE-EVOL), ACM, **(2010)**.
[4] C. P. Bezemer and A. Zaidman, “Challenges of Reengineering into Multi-Tenant SaaS Applications?”, The Software Engineering Research Group Technical Reports, **(2010)**.
[5] Z. H. Wang, C. JieGuo, B. Gao, W. Sun, Z. Zhang and W. H. An, “A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing”, IEEE International Conference on e-Business Engineering, **(2008)**.
[6] C. JieGuo, W. Sun, Y. Huang, Z. H. Wang and B. Gao, “A Framework for Native Multi-Tenancy Application Development and Management”, 9th IEEE International Conference on E-Commerce, **(2007)**.
[7] J. Du, H. Wen and Z. Yang, “Research on Data Layer Structure of Multi-tenant E-commerce System”, IEEE, **(2010)**.
[8] M. Hui, D. Jiang, G. Li and Y. Zhou, “Supporting Database Applications as a Service”, IEEE International Conference on Data Engineering, **(2009)**.
[9] D. Jacobs and S. Aulbach, “Ruminations on Multi-Tenant Databases”, In Business Technologie und Web (BTW 2007), **(2007)**.
[10] M. Grund, M. Schapranow, J. Krueger, J. Schaffner and A. Bog, “Shared Table Access Pattern Analysis for Multi-Tenant Applications”, IEEE Symposium on Advanced Management of Information for Globalized Enterprises, 2008, AMIGE, **(2008)**.
[11] W. Li, Z. Zhang, S. Wu and Z. Wu, “An implementation of the SaaS Level-3 maturity model for An Educational Credit Bank Information System”, In International Conference on Service Sciences, **(2010)**.
[12] Nitu, “Configurability in SaaS (Software as a Service) Applications”, In ISEC '09 Proceedings of the 2nd India software engineering conference, **(2009)**.
[13] The Force.com Multitenant Architecture.
[14] F. S. Foping, I. M. Dokas, J. Feehan and S. Imran, “A New Hybrid Schema-Sharing Technique for Multitenant Applications”, IEEE International Conference on Digital Information Management, **(2009)**.
[15] E. F. Codd, “Relational database: A practical foundation for productivity”, Commun. ACM, vol. 25, no. 2, **(1982)**, pp. 109–117.
[16] C. P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans and A. Hart, “Enabling Multi-Tenancy: An Industrial Experience Report”, The Software Engineering Research Group Technical Reports, **(2010)**.