

APPLICATION TENANCY





TABLE OF CONTENTS

| | |
|---|---|
| > 1. Introduction | 1 |
| > 2. SaaS and Tenancy | 1 |
| > 3. Provisioning in General | 2 |
| > 4. Multi-Tenancy in the Application Tier | 2 |
| 4.1 Identity Management | 2 |
| 4.2 Integration | 2 |
| 4.3 Communications | 2 |
| > 5. Provisioning the Application Tier | 3 |
| > 6. Database Concepts | 3 |
| 6.1 Database Management System (DBMS) | 3 |
| > 7. Alternate Approaches to Database Multi-Tenancy | 5 |
| 7.1 Single Database Engine and Database for All Tenants | 5 |
| 7.2 Separate Database Engines and Databases for Each Tenant | 6 |
| 7.3 Single Database Engine and Separate Databases for each Tenant | 6 |
| > 8. Summary | 7 |
| > 9. References | 7 |
| > 10. Glossary of Terms | 8 |



1. INTRODUCTION

This paper is one of a series of papers that explore and discuss the technical architectural components of the Software-as-a-Service (SaaS) model.

SaaS shares the distinction of being both a business model and an application delivery model. SaaS enables customers to utilize an application on a pay-as-you-go basis and eliminates the need to install and run the application on the customer's own hardware. Customers generally access the application via a web browser or thin client over the Internet. SaaS is most often subscription based and all ongoing support, maintenance, and upgrades are provided by the software vendor as part of the service. Application customization capabilities, if available at all, are generally provided to all customers in a consistent manner. From the perspective of the software vendor, the SaaS model provides stronger protection of its intellectual property, operational control of the environment running the software, and generally a repeatable revenue stream from the service subscription fees. Software vendors have varying capabilities and applications can come in varying flavors but SaaS applications most typically support many unique customers using a single instance of that application - also known as multi-tenancy.

This paper will define multi-tenancy and then focus on distinguishing and defining database concepts for the purpose of discussing alternative approaches to multi-tenancy.

2. SaaS AND TENANCY

Multi-tenancy is an architectural design concept used to maximize customer and user scalability, as well as to provide maximum operational scalability and lower support costs for the service provider by leveraging common user interface, business logic and databases for all customers and users. Multi-tenancy at all layers of the application is considered an architectural ideal but not required at all layers to realize benefits. The cost savings often associated with multi-tenancy are often considered key to building and maintaining lower cost service offerings.

This is admittedly an architecture-centric definition of multi-tenancy. There are other definitions that remain more at the business, operational or conceptual level^{1,2} but we will leave the reader to explore these separately. From an architectural perspective, multi-tenancy can be discussed at all tiers of the application (see the SaaS Architecture technical paper for reference to the SaaS Architecture Model and its multiple tiers).



3. PROVISIONING IN GENERAL

The concept of multi-tenancy expresses itself right from the start in a SaaS application with the provisioning of the service. Decisions have to be made about how a given tenant will reside in the environment while keeping their data separate from other tenants. As the service is first setup, any customer-specific configuration or customization must be accounted for. There is also the factor of potential metering and billing, and as the service is provisioned the customer needs to be setup so that their specific usage is distinguished from other users and tracked accordingly. These are just a few of the many aspects of provisioning that must be taken into account in a multi-tenancy environment.

4. MULTI-TENANCY IN THE APPLICATION TIER

The application tier can contain numerous servers providing various components of the overall service. We will review some of the common ones that will be impacted by multi-tenancy.

4.1 Identity Management

If a SaaS solution includes an identity management (IdM) component, it needs to be carefully designed so as not to compromise security. IdM can include access control via interaction with federated repositories such as Active Directory or other LDAP stores. The IdM server must manage interaction with various customer repositories and make sure that access to these directories is managed individually for each tenant.

4.2 Integration

Integration servers are involved with getting data from or sending it to other systems, often within corporate environments. Therefore it is critical to be able to handle multiple tenants in a secure way, keeping access credentials robustly separated and protected. It is also critical to make sure any integration APIs provide for authentication into a specific tenant's SaaS account so that data segregation is maintained.

4.3 Communications

As far as communication servers are concerned, multi-tenancy has more of an impact on inbound communications than on outbound. Outbound communications are inherently addressed to specific users, whereas inbound communications generally need to be routed to a specific tenant instance or account.

5. PROVISIONING THE APPLICATION TIER

Depending on the specific architecture of the application tier, it must be provisioned to support a new tenant. In some cases, all that is required is a set of authentication credentials that provide a tenant identifier to allow a customer to access their data set. However, there are other approaches to multi-tenancy at this tier that may involve the setup of a unique application server (or servers, per some of the ideas presented in section 4) for each customer. There are various reasons why a SaaS vendor might take this approach, including scalability and potential geographic distribution to fulfill data privacy requirements, but regardless of the rationale for the architecture proper provisioning must be performed to get a new customer up and running.

6. DATABASE CONCEPTS

The term database encompasses numerous concepts and ironically there is no complete agreement on the names, use and definition of these concepts. In order to facilitate a discussion of the alternative approaches to multi-tenancy in the database tiers of a SaaS Architecture, we will distinguish a model, apply names to the components of this model and define each of these components. This model is presented in Figure 1 below.

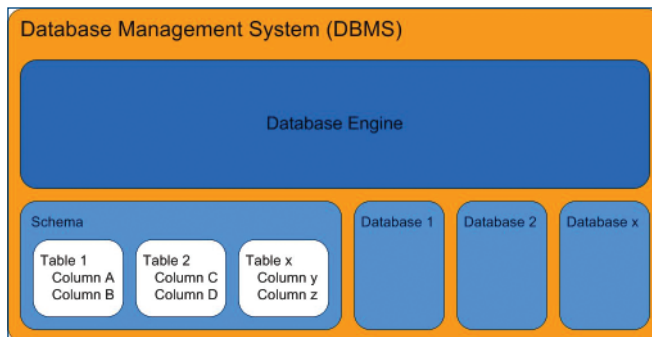


Figure 1 – Database Management System Model

6.1 Database Management System (DBMS)

A database management system (DBMS) is a complex set of software programs that control the organization, storage, management and retrieval of data in a database. For the purposes of this paper we will say that a DBMS is comprised of a Database Engine and one or more Databases. Each of these terms will be defined below. Note that a relational database management system (RDBMS) is a type of DBMS that stores data in related tables. This type is probably the most widely used today but in this paper we will refer to an RDBMS by the more generic term DBMS.



6.1.1 Database Engine

A database engine, sometimes referred to as a Database Manager, is the set of software programs that are used to store, update and retrieve a database. The database engine can be broken down further into components handling connection pools, a structured query language (SQL) parser and various management services and utilities, but these vary per DBMS vendor implementation and are not relevant to this discussion.

6.1.2 Database

We will define a database as comprised of the structure (or schema) and the data stored within this structure. In object oriented programming terms, a schema is like a class that defines the abstract characteristics of an object and a database is like an instance which is the actual object created at runtime.

6.1.2.1 Schema

The schema describes the structure of the database. The major constructs within this structure are tables, and within tables are columns. There are other constructs within a schema including indexes and views but these deal more with the way the tables relate to each other and therefore will not be discussed in the context of this model

6.1.2.2 Table

A table is a set of data elements (values) that is organized using a model of horizontal rows and vertical columns. The columns represent the distinct data elements defined in the table and the rows represent the data stored in these columns

6.1.2.3 Column

A column is a set of data values of a particular simple type, one for each row of the table. The columns provide the structure according to which the rows are composed. The term field is often used interchangeably with column, but it can be less confusing to use field (or field value) to refer specifically to the single item that exists at the intersection between one row and one column

6.1.2.4 Data

The data are the actual values stored in the columns within the tables that comprise the schema. In terms of this overall discussion, data would be the corporate information stored by a customer in a SaaS application.

7. ALTERNATE APPROACHES TO DATABASE MULTI-TENANCY

Now that we have defined a model with terms to describe a DBMS, it will be much easier to discuss alternative approaches to implementing multi-tenancy in the database tiers of a SaaS application. There is no one correct way to handle this implementation as the different alternatives have advantages and disadvantages depending on the nature of the application.

7.1 Single Database Engine and Database for All Tenants

The first alternative involves a single DBMS instance and is shown in Figure 2. This means there is a single database engine and a single, monolithic database containing co-mingled data from all customers. This model is the simplest in terms of configuration and maintenance. It relies heavily on the DBMS vendor to handle scalability and high availability on the database engine level and all the major vendors (i.e., Progress, Oracle, Microsoft, MySQL) support these features. From a data replication and backup perspective, this approach avoids the need to keep track of multiple databases. Data storage is on one hand efficient because reference tables only exist in the single instance. On the other hand, it is less efficient because each table with co-mingled data needs an extra column to distinguish one customer from another and each of these fields needs to be indexed.

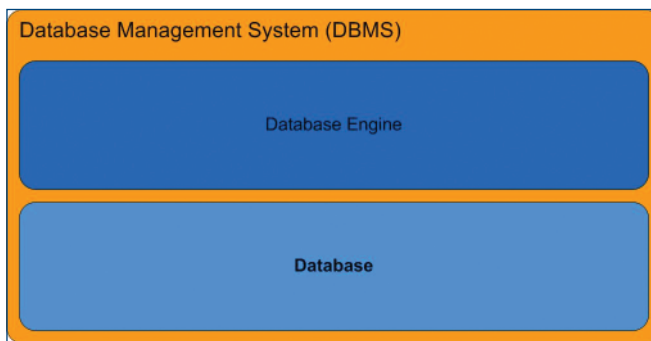


Figure 2 – Alternative 1: Single DBMS

One consideration that must be thought through before utilizing this approach is the perception of security. In reality it is possible to co-mingle data and implement a SaaS application that will correctly only allow a given customer access to its own data. However, sometimes perception trumps reality and customers may insist on physical separation of data, especially if there are regulators looking at all aspects of corporate and IT security controls. Ultimately the SaaS vendor needs to decide the approach that will best help sell the service while at the same time providing operational efficiency.

7.2 Separate Database Engines and Databases for Each Tenant

This approach is practically the exact opposite of the first, with each customer being provided their own DBMS instance (unique database engine and database). See Figure 3 below.

Having separate database engines and databases for each tenant provides complete isolation of individual customer performance at the data tiers, while increasing the cost overhead of operating the environment. There are scalable aspects of the database engine that do not get leveraged in this model, except in the limited case of very large customers.

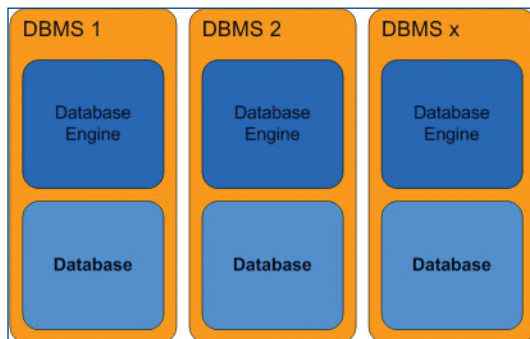


Figure 3 – Alternative 2: Single DBMS for each customer

This model provides the efficiency of managing the database for a given customer without impacting any other customers. For example, in a release that requires modification of the schema, customers can be converted one at a time, keeping downtime per customer limited.

7.3 Single Database Engine and Separate Databases for each Tenant

In this approach, there is a single DBMS and a separate database for each customer. See Figure 4 on next page.

This model provides some of the best aspects of the two models described above in sections 4.1 and 4.2. The scalability of the DBMS vendor's database engine can be leveraged while providing the data segregation that certain customers may demand.

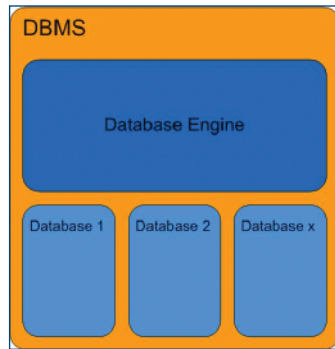


Figure 4 – Alternative 3: Single DBMS, separate database for each customer

One downside of this approach has to do with the release issue raised in the prior section. In order to update the schema of each database, the entire DBMS must be brought down while the scripts are being run. This impacts scalability as the more customers added, the longer the downtime per release for all the customers.

8. SUMMARY

In this technical paper, we covered the topic of application tenancy. We developed a definition that was focused on architecture while acknowledging the business and operational aspects of tenancy. We also noted that multi-tenancy can be addressed in all the architecture tiers.

First we covered the impact of multi-tenancy on service provisioning. Then we explored various aspects of the application tier and how multi-tenancy must be considered.

In order to discuss the impacts of tenancy on the database tiers, we first established a basic database model so specific terms could be distinguished and defined. This provided a basis for presenting four alternative database architectures for handling multi-tenancy. The four approaches were discussed in terms of some of their advantages and disadvantages. In doing so we presented many of the issues that need to be considered when choosing a database architecture for a SaaS implementation.

9. REFERENCES

¹Warfield, Bob (smoothspan), "Multitenancy Can Have a 16:1 Cost Advantage Over Single-Tenant," October 28, 2007 – <http://smoothspan.wordpress.com/2007/10/28/multitenancy-can-have-a-161-cost-advantage-over-single-tenant/>.

²Coffee, Peter, "Busting Myths of On-Demand: Why Multi-Tenancy Matters," Nov 2007 - <http://wiki.apexdevnet.com/images/0/04/MythbustMultiT.PDF>.

10. GLOSSARY OF TERMS

| Term | Description |
|--------------------------|---|
| Customer/Tenant | Customer/Tenant is a term in this document that refers to the company or business that subscribes to the SaaS offering. |
| Customization | In the context of a SaaS-enabled application, customization is defined as changes to the business logic, user interface, and/or database on behalf of a single customer, and requires application code to be recompiled. In practice, most SaaS-enabled applications either do not allow for customization, as it 'breaks' the scalability model by requiring the service provider to support multiple code bases, increasing costs, or customization provided to one customer is made available to all customers and users. |
| Multi-Tenancy | Multi-Tenancy is an architectural design concept used to maximize customer and user scalability. As well as to provide maximum operational scalability and lower support costs for the service provider by leveraging common user interface, business logic, and/or databases for all customers and users. Multi-tenancy at all layers of the application is considered an architectural ideal but not required at all layers to realize benefits. The cost savings often associated with multi-tenancy are often considered key to building and maintaining lower cost service offerings. |
| SaaS Offering | A SaaS Offering in this document is defined as a turnkey service offering, which includes the application license, maintenance, application support, subscription pricing, and hosting and associated delivery infrastructure. |
| SaaS-Enabled Application | A SaaS-Enabled Application is defined as an application that has been designed and built to be deployed and consumed in a service model, and incorporates many of the attributes previously defined above. |
| SQL | Structured Query Language is a standardized programming language for querying and modifying data and managing databases. |
| Users | User in this document is defined as an employee of the customer, as is the ultimate user of the service. |



Worldwide Headquarters

Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA
Tel: +1 781 280-4000 Fax: +1 781 280-4095
www.progress.com

For regional international office locations and contact information, please refer to www.progress.com/worldwide

© Copyright 2008 Progress Software Corporation. All rights reserved. Progress is a registered trademark of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. or other countries. Any other trademarks contained herein are the property of their respective owners.