

SaaS Data Architecture

An Oracle White Paper
Oct 2008

Introduction	3
DATA ARCHITECTURE APPROACHES	3
Separate Databases	4
Shared Database, Separate Schemas	4
Shared Database, Shared Schema.....	6
Choosing an Approach.....	8
Security	8
Performance & Scalability	9
Manageability	9
Conclusion.....	9

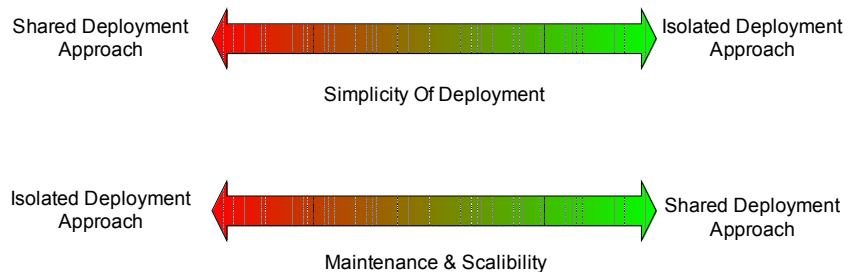
INTRODUCTION

The adoption of Software as a Service (SaaS) or On-Demand presents several technical and business challenges for software providers. These challenges include providing service at a low cost, addressing security concerns of customers and meeting service-level agreements. Data architecture for SaaS application can have a profound impact on a provider's ability to address these challenges. In this paper, we will look at different approaches of data architecture for SaaS applications. We will show you how Oracle Database can be very easily leveraged to create a highly scalable, secure and performing SaaS deployment, irrespective of the approach chosen, by making use of unique features and options available in the Oracle Database. We'll also explore some of the technical and business factors to consider when deciding which approach to use.

DATA ARCHITECTURE APPROACHES

The biggest decision while designing data architecture for SaaS is support for Multi tenancy. Multitenancy refers to a principle in software architecture where a single instance of the software runs on a software-as-a-service (SaaS) vendor's servers, serving multiple client organizations (tenants). Multitenancy is often contrasted with a multi-instance architecture where separate software instances (or hardware systems) are set up for different client organizations.

Data architecture is an area in which the optimal degree of isolation for a SaaS application can vary significantly depending on technical and business considerations. The distinction between shared data (multitenancy) and isolated data architecture isn't binary. Instead, it's more of a continuum, with many variations that are possible between the two extremes.



We shall examine three broad approaches, each of which lies at a different location in the continuum between isolation and sharing.

Separate Databases

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

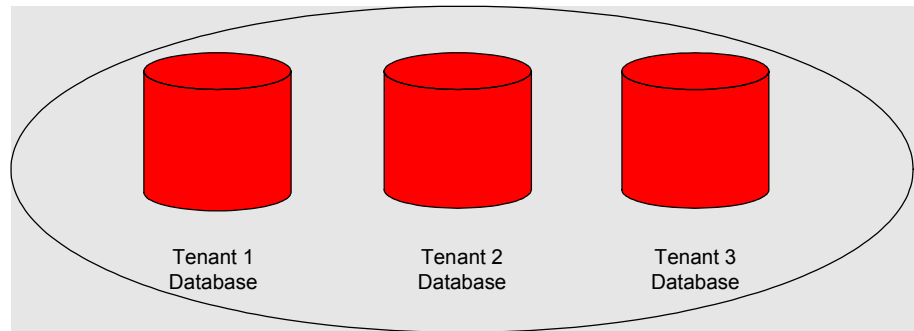


Figure 1 Hosting Environment with three separate Database Servers ensures complete Data & Access Isolation

In this case, each tenant gets an individual database computing resource and has a choice of either an individual application container or a shared one. The biggest benefit of this deployment approach is that the data remains physically isolated for each tenant. Giving each tenant their own database server makes it very easy to extend the application's data model to meet the tenants' individual needs. Although this approach is very secure, scalable and highly performing option, one should consider the cost of maintaining data and hardware availability.

This type of deployment is the "premium" approach, and the relatively high hardware and maintenance costs makes it appropriate for customers that are willing to pay extra for added security and customizability. For example, customers in industry such as Financial Services or Content Management often have very strong data isolation requirements, and may not even consider an application that does not provide each tenant with its own individual database server for maximum availability, future scalability and security compliance.

Shared Database, Separate Schemas

This deployment approach involves creating multiple tenant schemas within one database server, with each tenant having access to their own set of tables that are grouped into individual schemas created specifically for the tenant

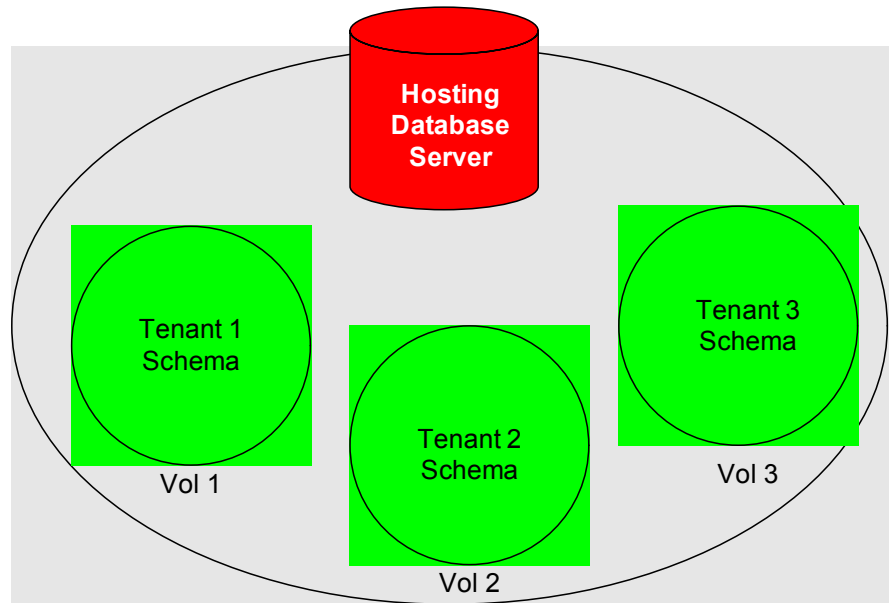


Figure 2 Each tenants' schema housed in a single database server across different data volumes

When a tenant first subscribes to the service, the provisioning subsystem creates a discrete set of table spaces for this new tenant schema and populates it with appropriate set of default application tables and objects for the tenant. This ensures data separation from other tenants' data.

You can use the SQL CREATE command to create the user and authorize a user account to access it. For example, in an Oracle Database:

```
CREATE tablespace tenant1_app_data tablespace datafile .....
```

```
CREATE user tenant1shema identified by password ...default tablespace
tenant1_app_data,...
```

The provisioning subsystem can then create tables within the tenant's schema as required by the application:

```
CREATE TABLE tenant1schema.Resumes (EmployeeID number(6) primary key,
Resume clob...)
```

You can also use a logical layer of synonyms to encapsulate the underlying schema. This way, a single set of SQL statements can be reused for all tenants:

```
SELECT * FROM resumes...
```

Like the isolated approach, the separate schema approach is relatively easy to implement, and tenants can extend the data model as easily as with the separate-database approach (Tables are created from a default script, but once they are created they no longer need to conform to the default set, and tenants may add or modify columns and even tables as desired.). This approach offers a high degree of data isolation for security-conscious tenants, though not as performing as a completely isolated system, and can support a virtually unlimited number of tenants per database server.

By deploying SaaS using this method, one has total flexibility of backing up individual tenants' tablespaces based on their volatility or Service Level Agreements (SLAs). A slight drawback of the separate schema approach is that tenant data is relatively harder to restore in the event of a failure where an incomplete recovery is required. But this situation can be addressed by using the Tablespace Point In time Recovery (TSPITR) and Flashback technology available in Oracle Database.

This approach can typically accommodate more tenants per server than the separate-database approach can, so you can offer the application at a lower cost, as long as your customers will accept having their data logically co-located with that of other tenants.

Shared Database, Shared Schema

This approach involves using one database and one schema to host multiple tenants' data. A given table can include records from multiple tenants stored in any order. A tenant identifier column associates every record with the appropriate tenant. The table is then List Partitioned or Range Partitioned by tenant identifier thereby creating an isolated set of tablespaces per tenant. Oracle database provides a robust and mature set of partitioning methods which will allow for physical data separation of each tenants' data across physical devices while providing simplification of maintenance due to shared table definitions.

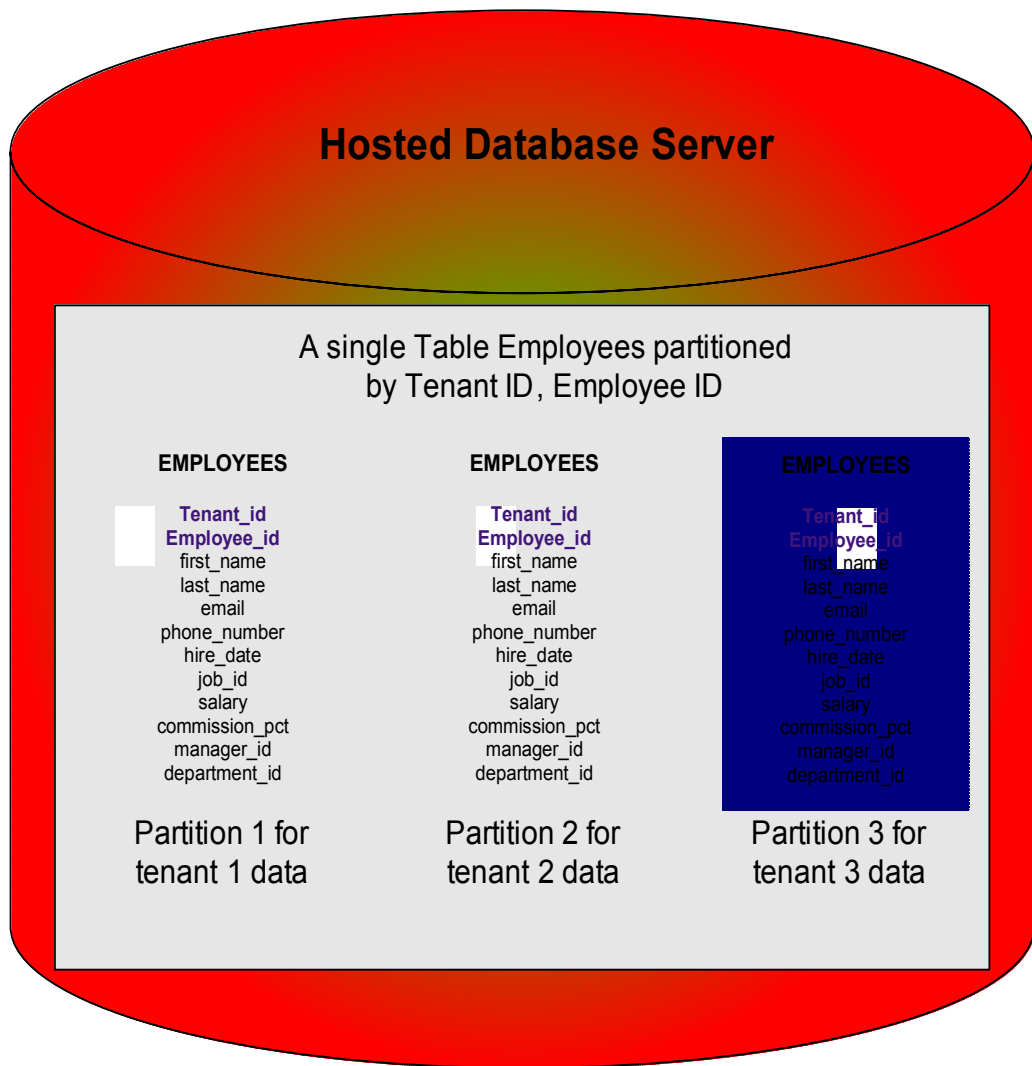


Figure 3 A single hosting database with one Schema holding all tenants' data.

The shared schema approach has the biggest barrier to entry for an existing software provider as it involves investment in re-designing the data layer to include tenant identifier column in various tables. However, it provides lowest hardware and backup costs as it allows you to serve the largest number of tenants per database server. With the use of the Partitioning option of Oracle database, it becomes very easy to create individual tenants scoped backup strategy based on their data volatility.

Oracle provides a very powerful feature called Virtual Private Database (VPD), which allows one to create Fine Grained Access (FGA) controls for database tables. Using VPD policies, tenants can be forced to automatically get access to only their data no matter how they happen to log into the database. VPD will

allow transparent tenant based access policy in a shared schema architecture without any code within the application. Separate policies can be specified for different interactions with the data (Select, Insert, Update, Delete).

Log Miner Tool or Flashback Transaction feature in Oracle database can be used in case a certain data change made by a tenant needs to be undone due to logical data corruption. However, care must be taken when recovering from physical failures since Mean Time to Repair (MTTR) and downtime usually will apply to all tenants .

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

CHOOSING AN APPROACH

Each of the three approaches described above offers its own set of benefits and tradeoffs that makes it an appropriate model to follow in some cases and not in others, as determined by a number of business and technical considerations. Some of these considerations are listed below.

Security

Security is one of the top customer concerns, which SaaS vendors have to address. As your application will store sensitive tenant data outside the customer's firewall, prospective customers will have high expectations about security. Your service level agreements (SLAs) will need to provide strong data safety guarantees. Oracle Advanced Security provides industry leading security features ranging from Transparent Data Encryption within the database, encryption of the networks, as well as, strong Password Authentication using Hardware based keys.

A common misconception is that only physical isolation can provide an appropriate level of security. This might force you to offer separate databases to some of your customers, particularly in heavily regulated industries. However, data stored using a shared database approach can also provide strong data safety, but requires the use of more sophisticated design schemes. Proper use of following database features can allow you to address security concerns in such an environment –

- Oracle's Virtual Private Database (VPD), which uses the concept of Fine Grain Access control, allows for policy-based access to all tables within the shared database. This ensures that each tenant can only access his or her own data irrespective of the access method.
- Oracle partitioning allows data separation of tenant data across multiple physical volumes.

- Oracle Database Vault allows separation of super user duties (Database Administrators, Application Developers, etc) on tenant basis.
- Oracle Audit Vault automates the audit collection, monitoring and reporting process, turning audit data into a key security resource for detecting unauthorized activity.

Performance & Scalability

Theoretically, greater performance and scalability can be achieved in a separate database approach allowing you to meet Service Level Agreements (SLAs). However, Oracle Real Application Clusters (RAC) allows you to address these concerns in any environment. Simply adding RAC nodes into the cluster as required can derive scalability and performance.

Intended resource consumption targets can easily be achieved in a shared database environment by defining and deploying Resource consumption profiles using Oracle Resource Manager thereby confirming to each tenants' SLA. All SaaS deployment types can greatly benefit by using Oracle Automatic Storage Management (ASM) for a free out of the box storage management solution for Oracle Databases.

Manageability

The Management cost can go up in case you are supporting multiple databases, increasing your total cost of ownership (TCO) for the SaaS infrastructure. However, Oracle VM and Oracle Enterprise Manager (OEM) Grid Control can be used as virtualization and management tools in such scenarios to lower your TCO.

OEM Grid Control allows for automated deployment, monitoring and troubleshooting of all components of your SaaS deployment architecture namely the ASM Disk Groups, databases, application servers, operating systems, and applications. With Oracle VM, you can virtualize any of these components and reduce the hardware cost.

CONCLUSION

The design approaches and considerations we've discussed in this article should help you create the foundation data layer that's vital to the success of your SaaS application. Designing a SaaS data architecture that reconciles the competing benefits and demands of sharing and isolation isn't a trivial task, but this paper should help you identify the approach best suited to your business. Further, the Oracle database technologies discussed here will help you meet the security, total cost of ownership and service level management requirements irrespective of the approach you choose.



Oracle SaaS Deployment Architecture
Oct 2009
Author: Rick Pandya
Contributing Authors: Shivanshu Upadhyay

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2006, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.