Designing a database for multi-tenancy on the cloud

Considerations for SaaS vendors

Raul F Chong
Senior DB2 and Big Data Program Manager
IBM

26 January 2012

Learn some considerations that new software as a service (SaaS) vendors need to take into account when developing applications or modifying existing ones to enable them for multi-tenancy on the cloud. The article discusses the considerations only from a database perspective — specifically, from an IBM® DB2® perspective. Six cases or methods are described.

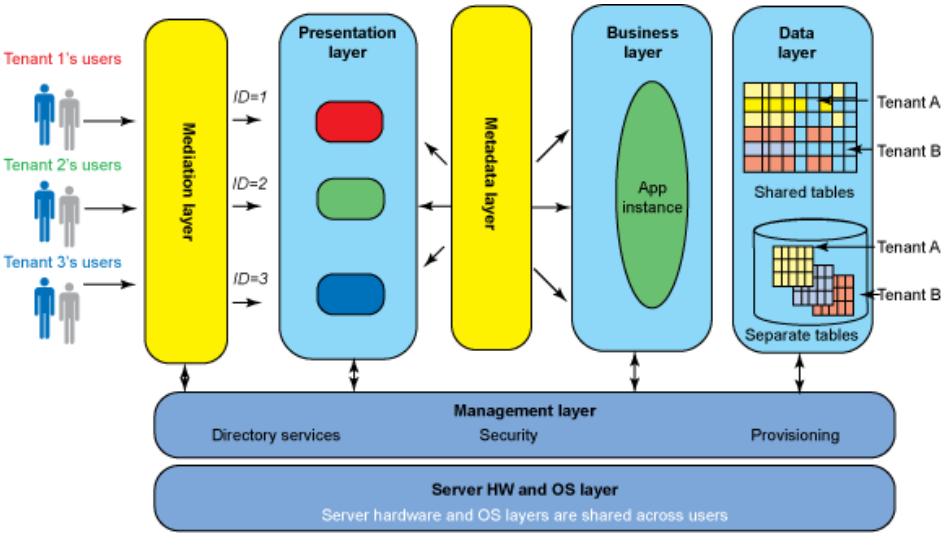
Introduction

Cloud computing is opening markets not previously contemplated by businesses. Some software companies are now thinking of delivering their software as a service instead of the usual method of developing the software and selling it to their clients using typical distribution methods. To become a software as a service (SaaS) vendor, companies need to find the right balance where resources are shared among the various tenants to reduce costs, while ensuring that customer information is kept private from other customers. There couldn't be a more serious problem than being able to see private information of one tenant from the account of another. In addition to tenant information privacy, SaaS vendors need to provide some level of customization for their customers.

In a multi-tenant environment, a SaaS company can reduce costs if it shares or reuses more of its resources. However, the more the company shares resources, the more risks it faces because an outage of a shared resource can potentially affect many customers. The more resources shared also adds to the complexity of the solution.

Figure 1 has been used in several IBM presentations to show an overview of a multi-tenant application environment.

Figure 1. A multi-tenant application environment



For this article, we will only focus on the data layer shown on the right side of Figure 1, and we will use DB2 software. The other layers can be handled by other IBM software, such as WebSphere® Portlet factory, WebSphere Portal Server, Tivoli® Directory Server, Tivoli Directory Integrator, Tivoli Provisioning Manager, Tivoli Monitoring, Tivoli Usage and Accounting Manager, etc.

Multi-tenancy at the data layer using DB2 can be used in various situations as discussed in the next six cases. Also keep in mind that if you are a small company and want to reduce licensing costs, you can consider using the no-cost version of DB2: DB2 Express-C. DB2 Express-C does not have any limits for the database size.

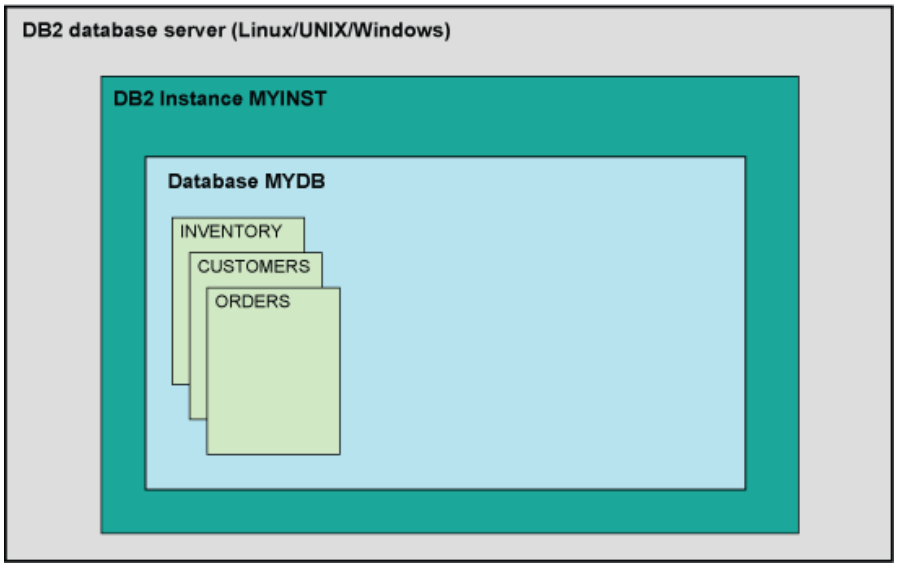
Case 1: Sharing tables

In this case, the following resources are shared:

- The database server
- The DB2 instance
- The database
- One or more table spaces
- One or more tables

Figure 2 shows an overview of the shared resources in this case. The *inventory*, *customers*, and *orders* tables have information from the clients of the various tenants.

Figure 2. Case 1: Sharing tables



The advantages of this case are that it provides the lowest cost, lowest storage, minimum amount of DB2 licensing, and minimum number of cloud instances needed.

The main disadvantage is that if, for example, one table becomes corrupted, it affects all customers. Also, application complexity might also be added in trying to always determine in your queries which records to retrieve for a given tenant.

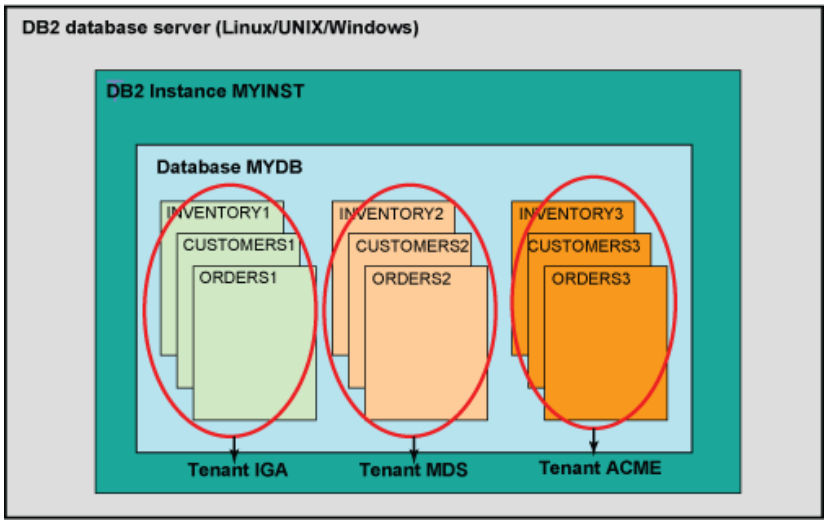
Case 2: Sharing a database

In this case, the following resources are shared:

- The database server
- The DB2 instance
- The database

Figure 3 shows an overview of the shared resources in this case.

Figure 3. Case 2: Sharing a database



In this case, the benefits are that sharing a database is still relatively low in cost in terms of still using one DB2 license and one cloud instance. Data isolation is good since different set of tables is used. Customization from a data perspective is easier since every tenant gets its own set of tables.

The disadvantages are that more storage is required since you need to create one set of the same tables per tenant. So comparing to case 1, you would be using *x* times more of storage, where *x* is the number of tenants. The application complexity is also increased and not as flexible, since now you need to customize your application to handle different table names and potentially different table structure in case there is specific customization for a tenant.

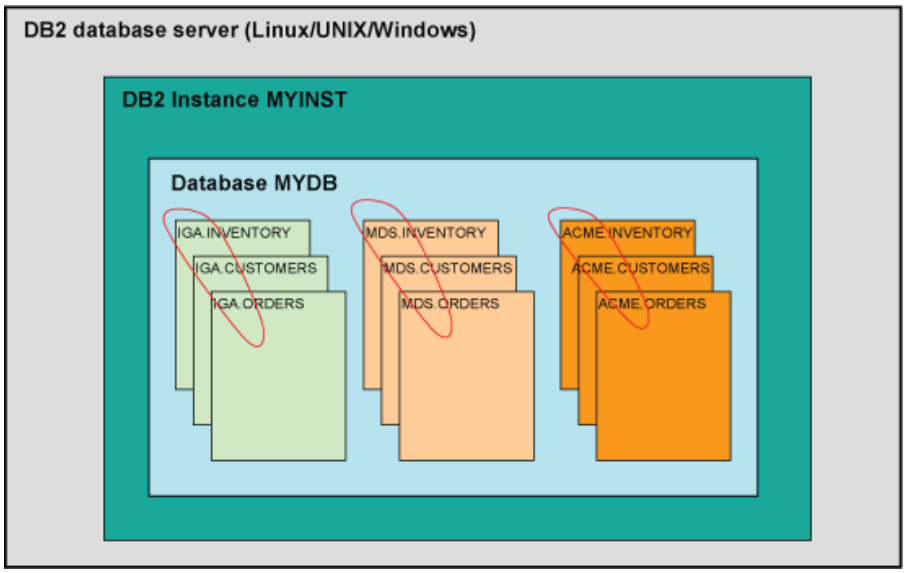
Case 3: Sharing a database and using a different schema name

In this case, the following resources are shared:

- The database server
- The DB2 instance
- The database

Figure 4 shows an overview of the shared resources in this case.

Figure 4. Case 3: Sharing a database and using a different schema name



Under this case, the benefits are that it is still low in cost, almost the same as in case 2. You still need one DB2 license and one cloud instance. Data isolation is good because a separate set of tables is used. Compare to case 2; there is less complexity in the application because the SQL statements used can be exactly the same. Redirecting a query to a given set of tables is done by changing the schema name by using the `SET SCHEMA` command. Customizations of a given table will obviously add complexity to your application.

The disadvantage, as in case 2, is that you must still use more storage because you would be creating one set of tables per tenant.

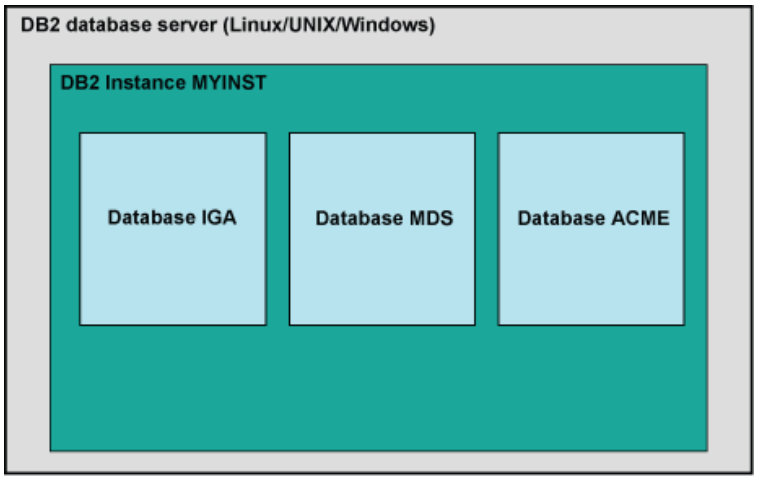
Case 4: Sharing an instance

In this case, the following resources are shared:

- The database server
- The DB2 instance

Figure 5 shows an overview of the shared resources in this case.

Figure 5. Case 4: Sharing an instance



In this case, the benefits are that it is still low in cost, almost the same as in case 2. You would still need one DB2 license and one cloud instance. Data isolation is very good because each tenant gets its own database, which in DB2 is an independent unit. Each database can be configured and maintained independently, providing more flexibility. The application complexity is less than in case 1. The table structure for most tables will be the same in all databases. If customization is required for one tenant, the table definition can be altered, but this adds complexity to the application.

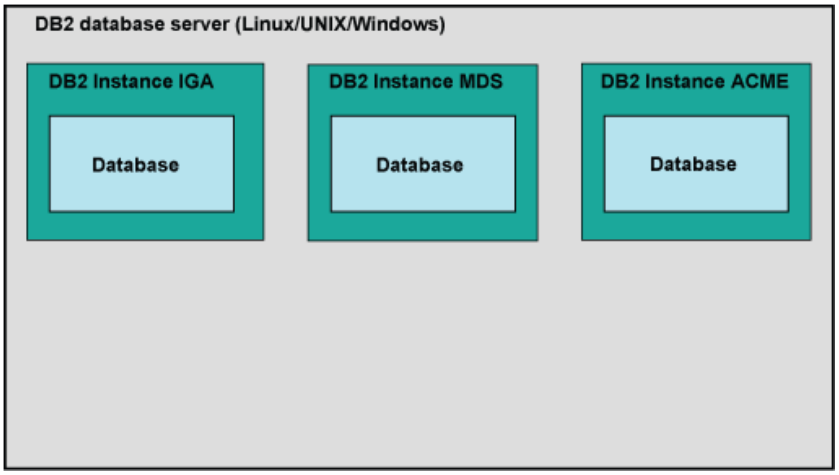
The disadvantage is that you will need more storage. Each database in DB2 creates its own catalog, which in other database products is known as the *dictionary;* therefore, more tables, views, and other database objects from the system must be created. In addition, in the case of DB2, there is a limit of 256 active databases per instance; therefore, under this scenario, only 256 tenants could work concurrently. Another disadvantage is that your memory consumption also increases, which can be troublesome on two fronts:

- You might reach the memory limit of the DB2 edition you are using, and will have to purchase a more expensive DB2 edition.
- You might reach the memory limit in your cloud instance, in which case, you will need to choose a more expensive cloud instance.

Case 5: Sharing a database server

In this case, only the database server resource is shared. Figure 6 shows an overview of the shared resources in this case.

Figure 6. Case 5: Sharing a database server



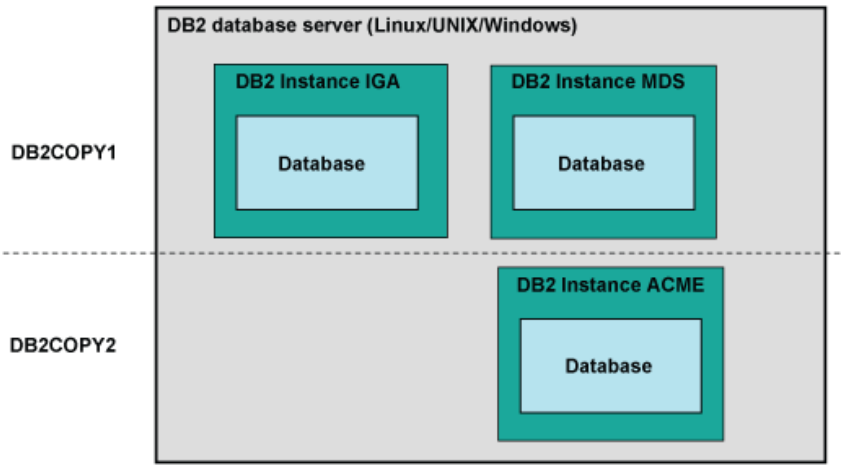
In this scenario, each tenant gets its own DB2 instance. The first benefit is good access control. Application complexity is similar to case 4. However, the system administrator might have to configure connectivity parameters appropriately in all the instances, which can mean more work to do. The table structure for most tables is the same and, as in case 4, for a given tenant, you can customize some tables, but application changes are required. Another benefit is that each instance and database can be maintained independently. If you bring down one instance, it affects only one tenant.

With respect to disadvantages, again, you see more storage required than in other cases, and you can also encounter problems with respect to memory. Although the number of instances in DB2 is limited by the operating system limits, and although starting an instance does not consume a lot of memory, having many instances started with several databases active at the same time might still cause memory problems. As a result, you could be forced to change your DB2 edition to a more expensive one or change your cloud instance to a larger, more expensive one. In addition to these disadvantages, administration complexity will also increase, which might warrant the company hiring more resources and directly affect costs.

Case 6: Sharing a database server with multiple DB2 copies

In this case, only the database server resource is shared. Figure 7 shows an overview of the shared resources in this case.

Figure 7. Case 6: Sharing a database server with multiple DB2 copies



For the purpose of a SaaS, there are really no benefits in using this approach. In terms of disadvantages, there are several:

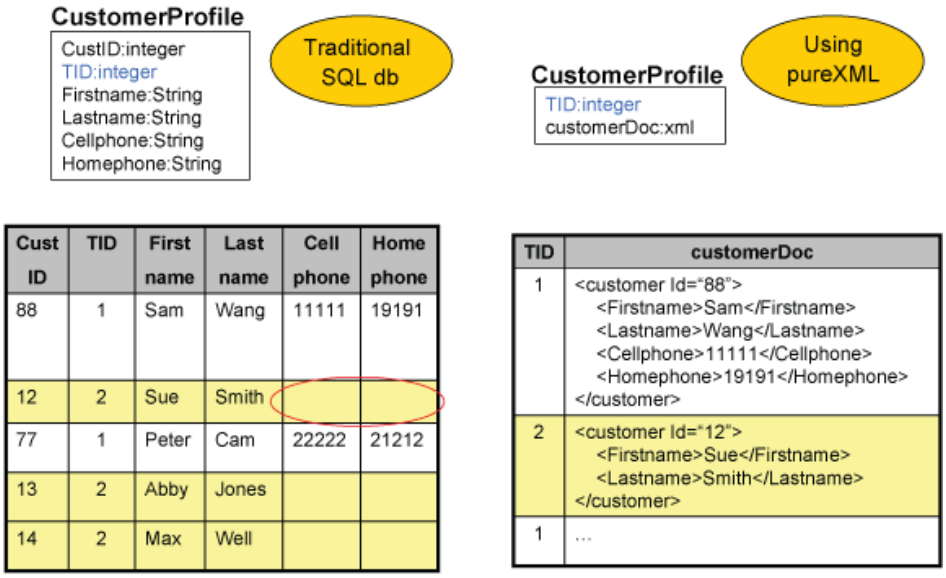
- More copies of the DB2 code must be stored in your cloud instance, taking up space.
- You have to install and configure DB2 for each copy installed; therefore, there is more administration setup time required.
- More application complexity exists, and this type of environment can confuse developers as to which database to connect in which instance of which DB2 copy.
- It has similar issues as case 5 for memory and storage consumption.

In summary, this case has no real benefit, but it has been added in this article for thoroughness.

Customization in the data layer

As mentioned, customization for a given tenant can add application and administration complexity. This section describes how customization can be handled by using XML — specifically, DB2 pureXML® technology, which provides more flexibility. Figure 8 shows an example.

Figure 8. Handling customizations for separate clients of separate tenants



The figure shows that a tenant has many clients, and each client has a different profile. The tenant ID (TID) column in both tables is used to identify the tenant. Rows 1 and 3 in the left table belong to one tenant with a TID of 1; rows 2, 4, and 5 belong to another tenant with TID of 2.

Suppose tenant No. 2 (TID = 2) has a business rule that indicates that clients cannot input telephone information, so storing information about telephone numbers would not be applicable to this tenant. However, in a multi-tenant environment where tables are shared (case 1), the SaaS company needs to consider that other tenants do want to include telephone information. Using a traditional SQL database (left table), the SaaS vendor can create one fixed-column table, which includes columns for all possible cases of telephone numbers (cellphone number, home phone). Even if tenant No. 2 doesn't allow phone numbers in customer profiles, the column is still included. Therefore, there will be a lot of "holes" and dispersed data, as highlighted by the circles in Figure 8.

In addition, say tenant No. 1 (TID = 1) wants to change its requirements so it doesn't store only cell and home numbers but also work numbers. In this situation, you may have to alter the table. However, if you follow the normalization rules for database design, you actually need to create a separate PHONE table. Then you'd have to move the data and change your applications so your SQL queries point to the new table PHONE and use `join` operations. This method is not flexible.

On the right side of Figure 8 is the suggested method of handling customizations. The table in this case has only two columns, where the second column is defined with the XML data type. Using XML, it is a lot more flexible to handle changes in the database schema. In addition, with DB2 pureXML technology, performance is greatly improved. DB2 9.7 also allows for schema evolution, so future changes to a given XML schema can be easily applied. In addition, DB2 allows multiple XML schemas per column, so each tenant can use separate XML schemas.

Summary

In this article, we have talked about the considerations that new SaaS vendors need to take into account when developing new applications or modifying existing ones. The article discusses the considerations or cases only from a database perspective — specifically, from a DB2 perspective. Six cases or methods were described. As in many situations in the IT world, you need to look for balance of costs vs. requirements when choosing a given method over the other. The methods where you share the most allow for better use of resources and less cost. However, if not designed correctly, they can cause a lot of trouble because failure of one resource can affect many tenants. Alternatively, using the methods where you share fewer resources can increase your costs, although the risk to other tenants is less.

As a SaaS vendor, you will likely have to choose the methods where you share resources. If you take the necessary precautions, such as using DB2 HADR, and other high-availability and redundancy controls, you can reduce or prevent problems in a multi-tenant environment. Some of these data resiliency controls are built into the cloud.

Resources

Learn

- Learn more about DB2 pureXML technology.
- Review the article "Get started with Hadoop-based data analytics on IBM SmartCloud enterprise" to learn how to set up a Hadoop cluster in the IBM SmartCloud Enterprise.
- Take the course Hadoop and the IBM SmartCloud Enterprise to grow your skills in this area.
- Learn more about IBM Cloud offerings.
- Learn more about DB2 HADR.
- Visit the developerWorks resource page for DB2 for Linux, UNIX, and Windows to read articles and tutorials and connect to other resources to expand your DB2 skills.
- Learn more about Information Management at the developerWorks Information Management zone. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with developerWorks technical events and webcasts.
- Follow developerWorks on Twitter.

Get products and technologies

- Build your next development project with IBM trial software, available for download directly from developerWorks.
- Now you can use DB2 for free. Download DB2 Express-C, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.

Discuss

- Participate in the discussion forum for this content.
- For current Cloud Computing information, visit this Thoughts on Cloud blog.
- Check out the developerWorks blogs and get involved in the developerWorks community.

About the author

Raul F Chong

Raul F. Chong is a senior program manager working in the Information Management Cloud Computing Center of Competence. He has been working at IBM for 14 years as a database consultant, support specialist, information developer, and technical evangelist. His main areas of expertise are in cloud computing, big data, and databases.