Building Multi-tenant Applications with Actian PSQL

Actian PSQL White Paper

May 2013

This white paper is the first in a series of papers designed to show how easily Actian PSQL can provide all the requirements for Software-as-a-Service. Other papers will explore SaaS subscription license delivery and the use of cache engine and VMware SQLFire to horizontally scale PSQL. The entire series is available on the Actian PSQL website.

Multi-tenancy and Actian PSQL: A Perfect Fit

The components of cloud computing—and its subtype, Software-as-a-Service (SaaS)—vary depending on the analyst, but in all cases, multi-tenancy is on the list.

Multi-tenancy refers to software architecture in which a single instance of software runs on a server, but multiple tenants are accommodated by that instance. It is similar to a boarding house in which each room houses a different resident, and although all residents share common facilities, each resident has privacy.

Because multi-tenancy is a relatively new concept, database software companies are scrambling to establish the best way to share an application between tenants while isolating each tenant's data. PSQL is uniquely constructed to provide multi-tenancy without the re-architecting required by other databases. Moreover, because multi-tenancy has always been part of PSQL's architecture, ensuring data isolation is not the costly undertaking with PSQL that it is with other databases. PSQL can make preparing for SaaS both less expensive and easier than you anticipated.

Common Approaches to Multi-tenancy

For Software-as-a-Service, approaches to multi-tenancy fall into two general categories: a reliance on dedicated resources within a shared infrastructure and the use of metadata to maneuver within a shared schema. While PSQL supports both of these approaches, it also provides methods that avoid several of their drawbacks.

Dedicated Resources

The dedicated resource approach is the use of separate databases, physical or virtual, one per tenant. This limits the access of each tenant to only specific resources. While tenants share computing resources and application code, each tenant's data is logically isolated and connected to the tenant through metadata. Limitations include higher hardware costs, because each server can support only a given number of tenants. There are also higher costs for maintenance and backup.

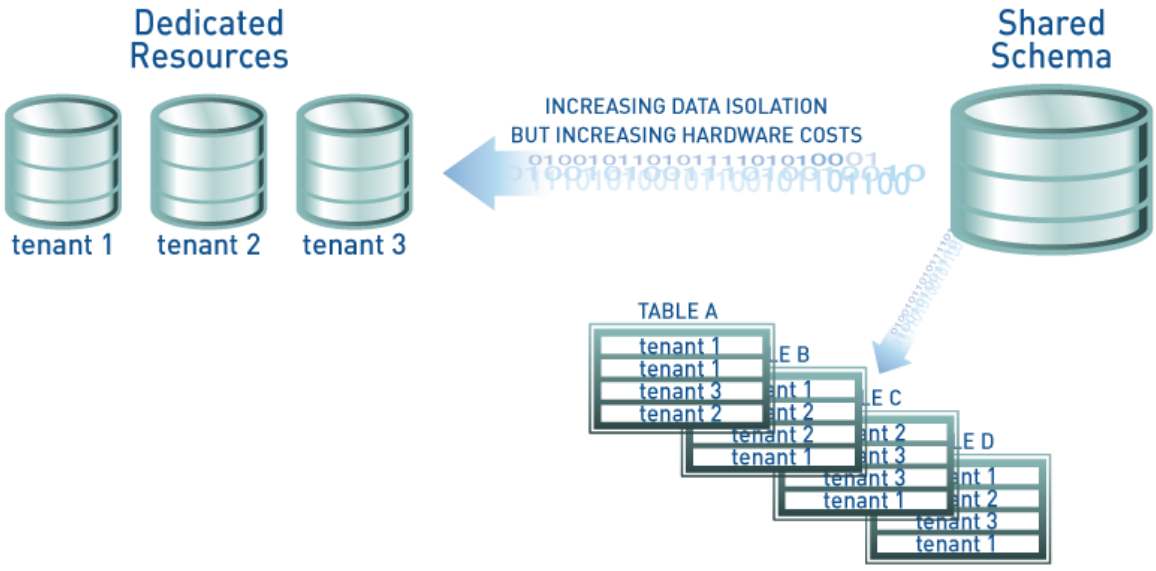
Shared Schema

The shared schema approach, on the other hand, uses a single database for all tenants. There are several varieties of this method. Each tenant can be assigned a discrete set of tables in the same database schema, or all tenants can share the same tables. In the latter case, the tables require an

additional column to identify the owner of each record. The shared schema method, with its reliance on tables, is obviously a product of the relational database model, and for database engines that rely on that model, it can be the most cost-effective approach because it keeps hardware costs low. However, it requires re-architecting the application from single-tenancy to multi-tenancy. It also reduces data isolation and security and introduces maintenance complexities: in the event of a failure, a process to restore only some tables or only some records can be time consuming and complicated.

The shared schema method can be deployed in different ways, depending on the number of tenants and the size of each tenant's data. One database can be insufficient to support all tenants, leading to the replication of the schema on additional databases. A logical extension to replication on additional databases is individual configuration of individual databases, moving toward a middle ground between the shared schema method and the dedicated resource method.

Methods of Multi-tenancy



The PSQL Approach to Multi-tenancy

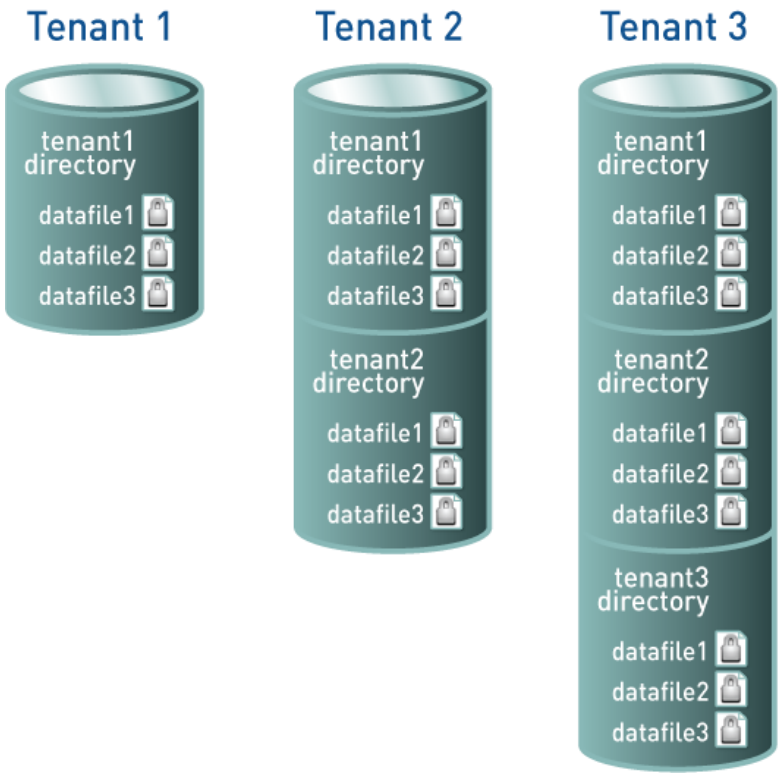
PSQL provides other options because it makes available interfaces not just to a relational database but also to a transactional, NoSQL storage engine, the MicroKernel Database Engine (MKDE). The architecture of the MKDE uniquely equips it for multi-tenancy without sacrificing data isolation or incurring increased hardware costs, even for application developers who prefer to work in a relational database.

The MKDE organizes data into files, which are stored in a directory on the server. Accommodating multiple tenants requires multiple directories rather than either multiple tables or multiple tenant identifications in the same tables. The file record layout in the initial directory can be duplicated and placed within a different directory for each additional tenant. Application developers who prefer accessing data through SQL rather than through the Btrieve API can construct individual secure relational databases for each tenant.

This approach provides the following advantages:

- Avoids any cost for re-architecting the application. Each tenant uses the original record layout.
- Ensures data isolation. Each tenant's data resides in a separate directory rather than a shared schema.
- Guarantees privacy for each tenant either through the assignment of a unique Btrieve owner name and files stamped with that name or through the establishment of database security on a relational level.
- Simplifies backup and restoration processes. Each directory can be backed up and restored individually and completely, without any effect on other tenants' data or operations.
- Provides application developers using the Btrieve API with the faster and more direct data manipulation provided by direct access to Btrieve.

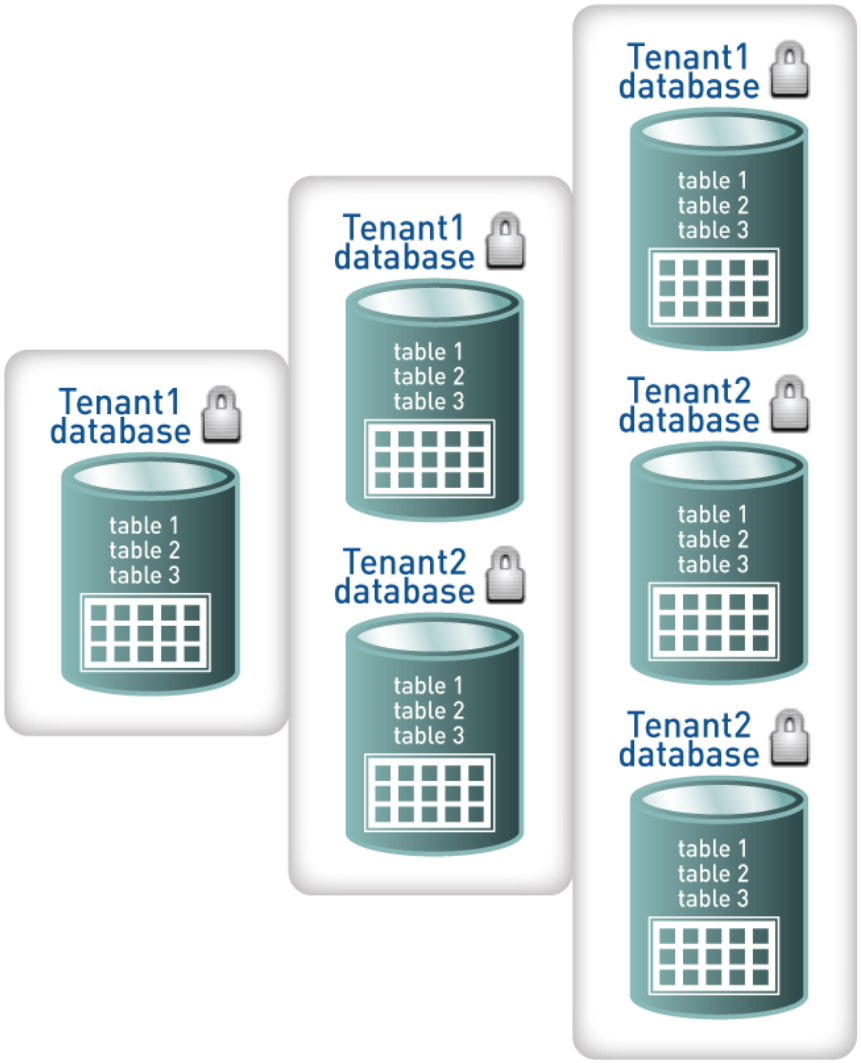
PSQL Approach to Multi-tenancy: Transactional



As the preceding graphic illustrates, the data for tenant 1 resides on the server in the form of a directory of data files. Each data file is stamped with the unique Btrieve name for tenant 1. When tenant 1 accesses and uses the SaaS provider's software application, it manipulates the data in the tenant1 directory. When tenant 2 subscribes to the same application, the provider establishes a separate directory for tenant 2's data. Tenant 2 is assigned a unique Btrieve name, and the data files in tenant 2's directory are stamped with tenant 2's unique Btrieve name. The process continues as succeeding tenants subscribe to the same service. Each tenant is assigned a separate directory and a unique Btrieve name, which is stamped on all files in that tenant's directory.

For application developers who prefer accessing data through SQL, the following graphic illustrates that process.

PSQL Approach to Multi-tenancy: Relational



When tenant 1 accesses and uses the SaaS provider's software application, it manipulates the data in the tenant1 database. A secure login process ensures that the entire database is accessible only by tenant 1. Tenant 2 is provided with a separate database, also locked at the database level. The process continues for each additional tenant.

Conclusion

For ISVs and OEMs eager to profit from the SaaS movement but hesitant because of either the cost of re-architecting their software or their client' need for data isolation, PSQL provides a solution that is both economical and secure. If your application segregates data for different divisions or locations of a company, it probably already provides multi-tenancy–making you more ready for SaaS than you thought. Because it offers access to the storage engine level of its architecture, it bypasses the problems of sharing a relational schema. Because of its inherent directory structure, it ensure data isolation without incurring the costs involved with providing and maintaining multiple databases or re-architecting the record layout.