Progress®

WHITEPAPER

PROGRESS® OPENEDGE® MULTI-TENANCY OVERVIEW

Table of Contents

Disclaimer

The information contained in this document represents the current view of Progress Software Corporation on the issues discussed as of the date of publication. Progress reserves the right, in its sole discretion, to modify or abandon without notice any of the plans described herein pertaining to future development and/or business development strategies. Any reference to third-party software and/or features is intended for illustration purposes only. Progress Software Corporation does not endorse or sponsor such third parties or software.

This white paper is for informational purposes only. PROGRESS MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT. No part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Progress Software Corporation.

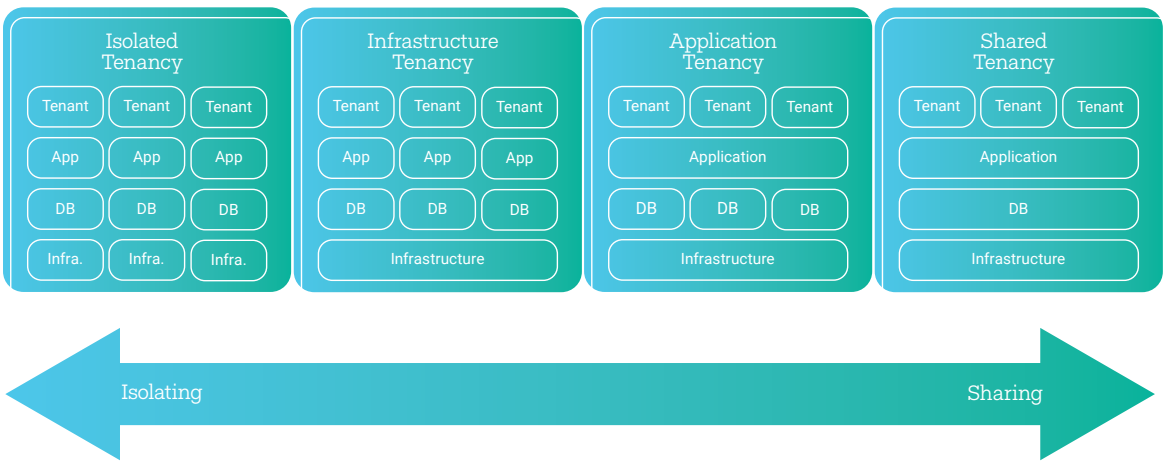
1. Multi-Tenant Databases and the Progress® Openedge® Difference

To understand multi-tenancy as a database concept it is easiest to start with the original derivation of the term, which, of course, comes from the familiar world of real estate. An apartment building provides multi-tenant housing, serving any number of tenants—from individuals who live alone, to couples, groups, and families of various sizes. Within the apartment building, each of these tenants has its own set of requirements for accommodations, storage space, and utilities.

Similarly, a multi-tenant database is one which provides database support to a number of separate and distinct groups of users, also referred to as "tenants." These tenants might be comprised of all of the users from a certain company, or tenancy might be defined more narrowly, with each tenant signifying a department or regional location within one particular company. Ultimately, there are no strict rules as to what constitutes a tenant in a multitenant database, since the grouping and association of users to tenants is specific to each deployment, but from a purely definitional standpoint a tenant is simply any logically defined group of users that requires access to its own set of data.

Given this definition of a tenant, there are different kinds of multi-tenancy that exist in database applications today. It is possible to view these different multi-tenant approaches as occupying a continuum of paradigms, some of which are shown in Figure 1.

Figure 1: The multi-tenancy continuum



With isolated tenancy (seen at the far left of Figure 1) each tenant has its own instance of the application running with its own instance of the database, as well as its own infrastructure to support the deployment. As we move further to the right in Figure 1, the "sharedness" of the tenancy in each paradigm increases: first with the tenants sharing just the infrastructure **(infrastructure tenancy)**, then sharing the application instance along with the infrastructure **(application tenancy)**, and then, ultimately, sharing everything: application instance, database, and infrastructure. This is **shared tenancy**, perhaps the "purest" of multi-tenant approaches. It is important to keep in mind that while the tenants in a shared tenancy all run the same application code and connect to the same database, within the database itself they access separate and distinct storage partitions, allowing each of them to accomplish their own business goals without any danger of infringing upon one another's data.

Each of the multi-tenancy approaches shown in Figure 1 comes with its own set of advantages and disadvantages. For example, if a tenant requires a customized version of the application and/or the database schema or has special infrastructure requirements, then isolated tenancy is theideal approach. However, as the number of isolated tenancy deployments increases, the cost of maintaining them also increases. An application provider maintaining an isolated tenancy deployment for an increasing number of tenants will have to invest considerably more time and effort to deploy code changes or carry out database maintenance for all of those tenants.

Shared tenancy (at the far right of Figure 1) provides a comparative reduction in maintenance costs because there is just one instance of the application, database, and infrastructure that is shared by all of the tenants. This makes code changes and database maintenance more straightforward and less costly than with isolated tenancy, while also adding scalability due to the ease with which additional tenants can be added to the deployment. But shared tenancy also has its disadvantages, making it harder to customize a particular tenant's code, customize schema, or provide any type of specialized service to a particular tenant. Tenants in a shared tenancy configuration may also have increased concerns over data security since their data is stored together on the same database instance.

Multi-tenant applications have taken an increasing foothold in the application marketplace largely due to the emergence of "software as a service" ("SaaS") as a deployment strategy. With SaaS, rather than purchase a license for a vendor's application and install it on-site, users connect to a hosted instance of the application and pay for it on a subscription basis. The subscription may be based on one or more metrics, such as the number of transactions, hours of usage, number of users, server utilization, etc.

The users of a multi-tenant SaaS application are unaware of whether the application is multi-tenant or not and, in fact, experience the application as though it were a totally isolated instance. When they are working with the application, there is no sense of whether other tenants are also connected at the same time, and the user of one tenant never encounters the data belonging to another. (Furthermore, tenants of an SaaS application enjoy a worry-free existence when it comes to application updates or database maintenance tasks since the SaaS provider handles all of these functions for them.)

For the SaaS application provider a multi-tenant deployment strategy provides many benefits, especially when a shared tenancy approach is used. Management of the deployment can take place either at the overall application or database level, affecting all tenants simultaneously, or based on individual tenants. Operational costs are greatly reduced due

to the number of shared resources and because only one instance of the application and database is involved. For example, when application or schema updates are required in a shared tenancy, they need to happen only once for the entire set of tenants.

With Progress® OpenEdge® releases prior to OpenEdge 11.0 one can build a multi-tenant environment using any of the options shown in Figure 1. Within these applications, shared tenancy is achieved by incorporating a schema column that is shared across all tenants to keep track of a tenant's identity. Any table within the database that is intended to be multitenant- enabled must include such a column. Also at any place in the application where a tenant-specific action is required (such as in database queries, creates, finds, and deletes) the code must use the "tenant-ID" schema-column to assure that the action is happening for the correct tenant. (With SQL this is somewhat mitigated through the use of views. However, maintaining the views for each newly added tenant brings its own set of complexities.)

Within an application of significant size, with database access occurring at every turn, the "tenant-ID" approach to shared multi-tenancy can add substantially to the size and complexity of the code. This, in turn, increases the overall cost of software development, QA testing, and maintenance. For the lifetime of the application, code additions or changes must remain "tenant-aware," making sure to incorporate the "tenant-ID" column properly.

An additional consequence of the "tenant-ID" approach to multi-tenancy is that the database records for the different tenants are intermingled within the database, with no partitioning based on tenancy. This, along with other factors, makes it impossible to perform database maintenance tasks that are tenant-specific. Operational activities must always approach the database as a monolithic whole.

OpenEdge 11.0 significantly simplifies the development of multi-tenant applications by reducing and even eliminating the challenges typically presented when implementing a shared tenancy application. This is due to two major factors:

- Tenant support is implemented in the database layer, completely removing the requirement that tenancy be managed by the application.
- Database utilities and tools are tenant-aware. This means that their operational characteristics include an accommodation for multi-tenancy wherever appropriate.

The implication of the first point, implementing tenant support in the database layer, is profoundly significant since it means that in OpenEdge 11.0, once the tenant identity is asserted, the code that tenants run when using a multi-tenant database is **exactly the same as** with the same application running in a non-tenancy configuration. The tenancy (or non-tenancy) aspect is completely transparent.

The reason is that OpenEdge 11.0 builds multi-tenancy right into the database, thus eliminating the need to add a "tenant- ID" schema column to manage tenancy within the application. In fact, in OpenEdge 11.0 no changes at all are required to an application in order to enable a table to be multi-tenant and to access it in a multi-tenant fashion.

Furthermore, as stated in the second point above, with OpenEdge 11.0 operational aspects of the database are also capable of acting in a mode that is tenant-specific. Index maintenance, data dump and load, object moves, and other database functions can take place at the tenant level as well as for the entire database. This is partly due to the fact that the data for each tenant is stored physically separate from the data for other tenants in the database by means of tenantbased partitions.

Stated simply, OpenEdge 11.0 makes the development and deployment of multi-tenant applications faster and easier by taking nearly all of the cost-intensive aspects of multitenancy and making them transparent. Multi-tenancy is made straightforward not just for the user but for the application provider as well. New applications can be brought to market more quickly and can then be maintained in a manner that ideally accommodates the multi-tenant paradigm.

OpenEdge 11.0 supports multi-tenancy with new features and enhancements in several areas of the product:

- The database engine, to support multi-tenant database objects
- The languages (ABL and SQL)
- An ABL API for managing tenants and multi-tenant objects
- Database utilities and tools enhanced to operate on a per-tenant basis
- Database monitoring, enhanced to support multi-tenancy
- A GUI tool for managing the multi-tenant database, provided as an extension of OpenEdge Explorer

The purpose of this document is to provide an overview of the OpenEdge multi-tenant database architecture. In presenting the material it occasionally touches upon the support for multitenancy in other areas of the product, however an in-depth discussion of multi-tenant tooling, ABL and SQL support, and the ABL API are all beyond the scope of this document.

2. The Openedge Multi-Tenant Database Architecture

The multi-tenant database architecture of OpenEdge 11.0 has been designed to make shared tenancy as fast, easy, and transparent as possible. This focus on shared tenancy does not preclude the use of one of the other multi-tenancy approaches shown in Figure 1 (or even others). In fact, with OpenEdge 11.0 there are direct benefits when using other multi-tenant approaches as well.

This section provides a high-level overview of the OpenEdge multi-tenant database architecture. Subsequent sections discuss the features in more detail.

Schema Definition, Data Storage, and Default Field Values

As described in the previous section, in the shared tenancy model, multiple tenants share a single instance of the application, database, and infrastructure. OpenEdge 11.0 introduces multi-tenant support to the database, with implications for schema definitions, data storage, and default field values.

For each multi-tenant table, one schema definition is shared by all tenants, with each tenant's data stored in its own data partition. Once a user has been authenticated as belonging to a particular tenancy, all data access within the database is based on that tenancy. This access model is fully managed by the database.

For a multi-tenant table the data associated with one tenant resides in one storage area while another tenant's data resides in another. What results is physical separation of tenant data by means of a data partition within each table. Such data may be physically laid out in the database in a variety of ways, accommodating various approaches that might be needed for optimal performance or maintenance. For example, each tenant's data partition may be stored in a separate storage area, or in a separate cluster within the same storage area.

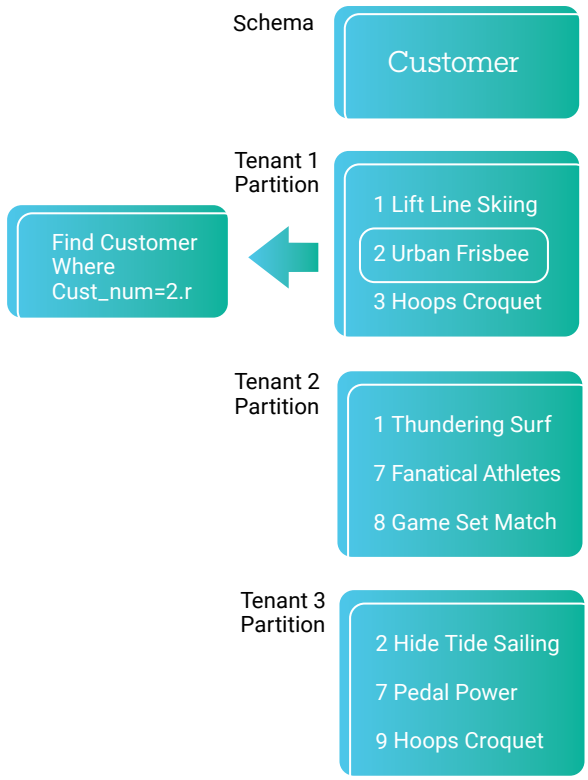
Within a multi-tenant table there is one set of default field values shared by all tenants. In other words, default values within a multi-tenant table cannot be different for different tenants.

Tenant Authentication and Data Access

Once a user is authenticated as being a member of a particular tenancy in the database, the user credentials (which include the domain that the user has authenticated to) are stored in the CLIENT-PRINCIPAL object. SQL has a slightly different mechanism, using an expanded userid to achieve the same result. From that point forward, all access to multi-tenant tables is automatically carried out based on the authenticated tenant identity. For example, the ABL chooses the appropriate tenant local index to use when operating on a multi-tenant table.

Figure 2 below illustrates data access by an authenticated tenant user. When a user for Tenant 1 runs an ABL statement to find the customer record where CUST_NUM = 2, the record returned is from the Tenant 1 partition only. While Tenant 3's partition also contains a customer with CUST_NUM = 2, due to tenant authentication this record is completely omitted from the handling of the FIND for Tenant 1. (However, if Tenant 3 were to run the exact same FIND statement, then the record returned would be the record from Tenant 3's partition where CUST_NUM = 2: "High Tide Sailing".)

Figure 2: Data access by an authenticated tenant user



An important characteristic of the FIND statement used in Figure 2 is that it provides tenant access to data without incorporating any kind of "tenant-ID" column in the syntax. Based on the user having been authenticated as belonging to Tenant 1, the FIND returns only Tenant 1's own data when executed against the multi-tenant "Customer" table.

In addition to the multi-tenant tables in a multi-tenant database, there is the concept of **shared tables**. A shared table is the same type of table that existed in OpenEdge databases prior to OpenEdge 11.0.

A shared table can be used within a multi-tenant database when the data associated with it is the same for all tenants. Examples are tax tables, actuary tables, etc. that are used universally by the application with none of the data requiring anything tenant-specific. If a table is defined as "shared," then authorized users of tenants of the database will share all the data in the table. (A user who is authenticated to the database but who is not authenticated to any tenancy will also be able to access the shared tables.)

So, with tenancy established, a tenant will be able to see (a) its own data stored in its own data partition of the multi-tenant tables, and (b) the shared data of non-multi-tenant tables. It will not be able to see any of the data that other tenants have stored in their own partitions for the multi-tenant tables.

Within OpenEdge 11.0 the existing authorization mechanisms ("can" permissions for ABL and "grant" for SQL) have been extended to be tenant-specific. Thus, one tenant can have different permissions for the same table definition than another tenant in the same database.

Multi-Tenant Indexes And Sequences

One of the other key capabilities included with multi-tenant tables is the support for indexes and sequences at the tenant level.

The definition of an index for a multi-tenant table must be the same for all tenants, but the instances of the index can be unique per tenant partition. In other words, since tenants within the same multi-tenant table each have their own instances of the index, they can each maintain their own unique key entries within that table. An example of this is seen in Figure 2, where two different tenants (Tenant 1 and Tenant 3) each have a customer record where CUST_NUM =2. The customers with key value = 2 are unique within the tenant partitions for each tenant, but are not unique for the entire table. This concept of "multiple unique values" is possible because each multi-tenant partition within the table is considered a separate entity. Database multi-tenancy makes this possible by providing a local index for each multi-tenant table partition instance. (A local index in this context is defined as a b-tree index that only contains entries for a specific tenant's partition of a multi-tenant table.)

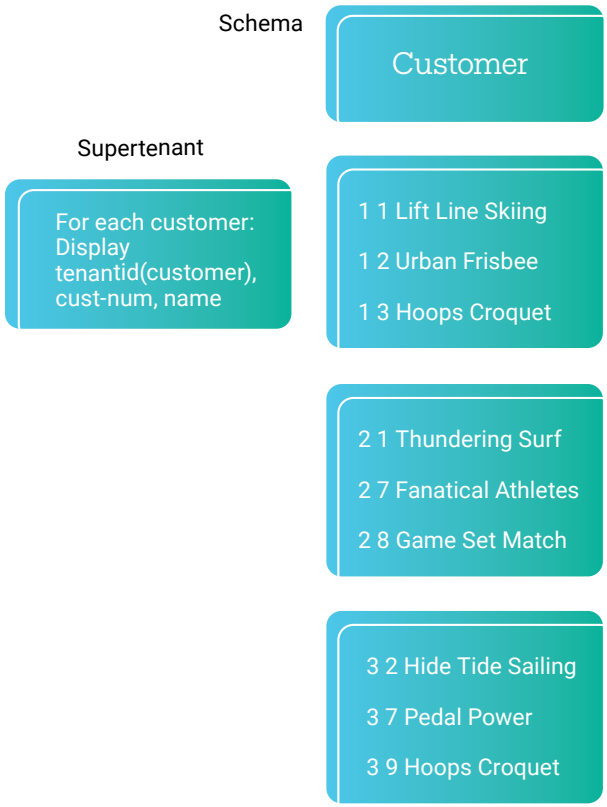
Sequences in multi-tenant tables can have a similar uniqueness per tenant. While each sequence definition is shared by all tenants of the database, the sequences themselves can be identified as tenant-local or shared amongst tenants. In the tenant-local case, when one tenant gets the next value for its instance of a sequence, it does not affect the next value associated with a different tenant in the database. (In the shared sequence case, all tenants see the effects of one tenant increasing the sequence value.)

The Super-Tenant

With database multi-tenancy there exists a special type of tenant called the super-tenant. The super-tenant has the ability to access and maintain data in any partition and for any tenant of the database. There can be multiple users identified as super-tenants for the same database.

A super-tenant has access to all records in a multi-tenant table, regardless of which partition they reside in. This capability makes it possible to run code that needs to access data in more than one tenant partition, for example, when a "home office" super-tenant wishes to run quarterly reports for "branch office" tenants in a database. Such a report requires the ability to access data within the tenant partitions without restriction.

Figure 3: Data access by an authenticated tenant user



With no physical column identifying the tenant ownership of a record in a multi-tenant table, there is still a need for a super-tenant to be able to determine which tenant a particular record belongs to. For this purpose, language constructs have been provided to determine this information at runtime. This concept is illustrated in Figure 3 (page 11.) (Note: ABL syntax shown is for illustrative purposes only and may not be the actual syntax implemented in OpenEdge 11.)

In Figure 3, the super-tenant's FOR EACH statement returns all of the customer records for all of the tenants of the multitenant table, as indicated by the red outline. Furthermore, the super-tenant's DISPLAY statement utilizes a TENANTID( ) function to display the tenant owner of each record returned. This syntax is required since the tenant-ID within an OpenEdge multi-tenant database is a **virtual** object, as opposed to the physical schema column that is used in other multi-tenant implementations.

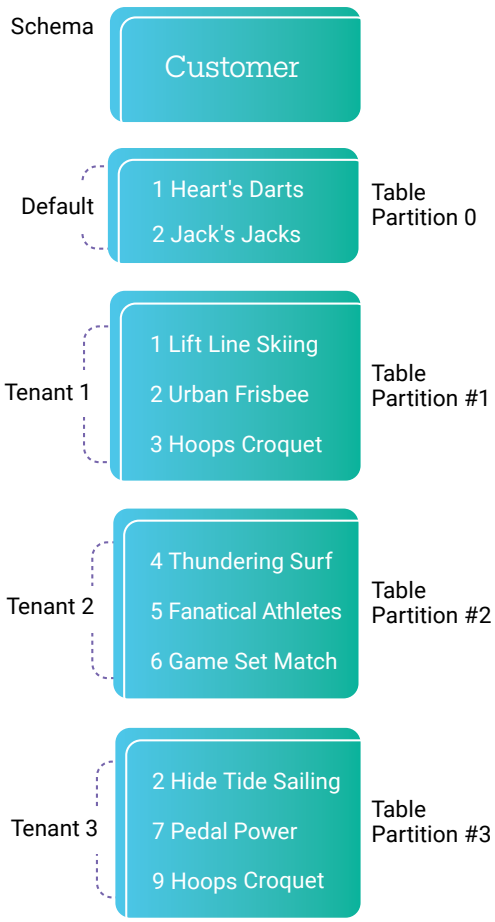
Similarly, a super-tenant can use other ABL constructs to selectively access the data for just one specific tenant or a subset of all the tenants. When these language constructs are used, the data retrieved is in the associated index of the tenant specified. Additional language syntax has been provided to allow a super-tenant to restrict the data retrieved to one or more specific tenant identities through the use of tenant-IDs as well as more user-friendly tenant names.

The Default Partition

Within multi-tenant tables there exists the concept of a default partition. A default partition must always be included when creating a multi-tenant table. Within the multi-tenant table, the default partition is one that does not have a designated "tenant" owner. A user who does not establish tenancy through the CLIENT-PRINCIPAL, _user or through connection in SQL will be allowed to access the default partition of a multi-tenant table based on the table's regular authorization permissions to make such changes. (As stated earlier, the non-tenant authenticated user will also be allowed to access shared tables in the database.)

Figure 4 shows a multi-tenant Customer table with a default partition ("Table Partition 0").

Figure 4: Multi-tenant table with default partition



Tenant Creation

Creating new tenants in a multi-tenant database (also referred to as provisioning tenants) can be accomplished programmatically (using ABL, an ABL API supplied by OpenEdge, or SQL) or by using a web-based GUI interface that is an extension of OpenEdge Explorer. The ABL enhancements provide multi-tenancy support through schema record creation and associated schema traps, while the ABL API allows a DBA or tenant administrator to create and configure new tenants and multi-tenant tables. The web-based tooling for multi-tenancy enhances the existing OpenEdge Explorer to provide configuration and data administration of tenants and multi-tenant data.

When creating a new tenant, the user has the ability to specify a storage area location for each multi-tenant object (table, index, line of business—LOB) partition instance that is created for the new tenant. The user also has the ability to suppress partition creation for the new tenant in any of the existing multi-tenant tables. A decision not to create a partition in a particular table for a new tenant might be useful when a tenant is utilizing only a subset of an application's features and therefore does not require the storage space.

The SQL DDL has also been extended to support the creation of multi-tenant tables and new tenants.

When creating a new multi-tenant table in a multi-tenant database which already has tenants provisioned, a new multi-tenant table partition instance is automatically created for each of the tenants. Again, the storage area location of the new multi-tenant table partition can be specified when creating the table, or it can use the default storage area locations associated with each tenant, or it can avoid instantiation for particular tenants altogether.

SQL Query Support

OpenEdge SQL uses a cost-based optimizer to determine the query plan to execute. A significant determination of the cost associated with a particular query is based on the statistics associated with the table or tables involved in the query.

Since tenant queries are executed on a particular partition of the table, to support multi-tenancy the SQL statistics have been enhanced to be specific to a multi-tenant partition rather than the entire multi-tenant table. This allows the query optimizer to build query plans based on statistics of a specific tenant's data and ensures the best performance for queries executed on multi-tenant tables. The update statistics and automatic statistic generation utilities have been appropriately updated to reflect this.

Database Utilities

As described in more detail in Section 4 of this document, all database utilities that have an object-based frame of reference (such as binary dump/load and database repair) are also tenant-aware. Utilities that have a database-wide frame of reference (such as after-imaging) will continue to be database-wide, and all utilities that require the database to be offline will remain offline.

Additionally, all database monitoring tools have been enhanced to display statistical and status information associated with a user's tenant identity as well as with the tenant ownership of each tenant partition of a multi-tenant table.

Tenant Activation and Deactivation

The database administrator (DBA) has the ability to activate or deactivate data access on a per-tenant basis. Deactivation of a tenant results in the tenant being unable to access the data in any of their partitions of the multi-tenant tables. The deactivated tenant instead receives a runtime error when attempting to retrieve data. This allows the DBA to revoke tenant access without having to delete the tenant data or take the entire database offline. The tenant's data remains intact and can be accessed by the tenant again once they have been activated (or reactivated) in the database. A simple use case for tenant deactivation is in an SaaS environment where a particular customer is far behind on their license payments and has to be temporarily shut off.

This capability is provided in addition to existing approaches through a change of access rights or from within the application itself.

3. Tenant Access Paradigms

Having covered the architectural aspects of multi-tenancy, we can now explore the two tenant access paradigms supported in OpenEdge 11.0: **simple multi-tenancy** and **multi-tenant groups**.

Simple Multi-Tenancy

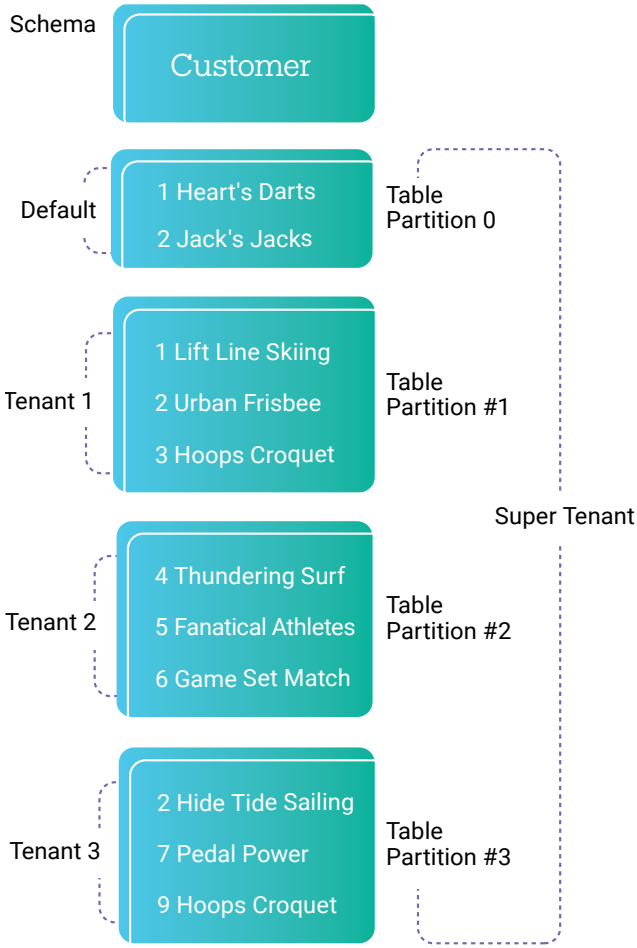
The basic level of support for multi-tenancy, known as **simple multi-tenancy**, is an implementation where a 1-to-1 relationship exists between the tenant and its table partition, with additional support for the super-tenant. With simple tenant access, each tenant has its own tenant partition assigned to it for a multi-tenant table. This is the paradigm that was used to illustrate the multi-tenancy principles discussed in the previous section.

Figure 5 illustrates simple multi-tenancy by depicting a single definition of the multi-tenant Customer table with three distinct tenants of the database. Each of these tenants has its own multi-tenant table partition within the Customer table where its data is stored and maintained. Each tenant can see only the data associated with its own table partition instance.

Figure 5 also includes the default tenant partition ("Table Partition 0"), which was described in Section 2. Any user that does not assert their tenancy when connecting to the database is considered to be a default tenant. Such a tenant has access only to the default tenant partition of a multitenant table, if such a default partition exists.

When any tenant executes code involving the Customer table shown in Figure 5, the only data that can be accessed is shown by that tenant's corresponding bracket. For example, when Tenant 2 runs a "FOR EACH" on the Customer table, the records that can be accessed are only those from Table Partition #2, which is colored green.

Figure 5: Simple multi-tenancy with default partition



Super-tenant access is also depicted in Figure 5. As mentioned in Section 2, the super-tenant is able to access all of the tenant partitions of a multi-tenant table for the individual tenants and for the default tenant. So, when the super-tenant runs a FOR EACH statement against the Customer table, all of the customer records shown in the diagram can be accessed for each and every partition.

When a tenant is deleted from the database (as opposed to merely being disallowed access) its table partition is also deleted.

Multi-Tenant Groups

Multi-tenant group support allows two or more tenants to map to a single table partition for a multi-tenant table, consequently sharing all of the same data for that table. An example might be a chain of stores, where each store location is a tenant in a retail database but with all of these locations/ tenants sharing the same inventory in a warehouse. The table containing the inventory data would be multi-tenant-enabled, and all of the store locations/tenants would access it as a single tenant group. Since the data in the Inventory table is shared by all of the tenants within the group, any insert, update, or delete by one tenant has a direct impact on the records accessed by the other tenants in the group.

Since all of the data within the multi-tenant group's table partition is shared equally by the tenants who are part of the group, there is no concept of row-level tenant ownership for the rows within the group partition. As a result, when part of a multi-tenant group, an individual tenant need not write a special query to access the records in the group partition: an internal partition manager handles record access using a partition ID associated with each tenant's request.

A tenant may belong to only one group for a particular multitenant table. Also, if a tenant group is defined for a specific multi-tenant table, not all tenants need to be part of that group. This leads to the corollary that each multi-tenant table can support zero, one, or multiple groups and may ultimately end up consisting of a mix of partitions for individual tenants and tenant groups.

Additionally, a group is specific only to the multi-tenant table to which it is assigned. So, a tenant who is a member of a group for one multi-tenant table is not required to participate in any other groups for other multi-tenant tables. For example, a chain of stores that shares a common customer list for all of its locations as part of a multi-tenant group might assign each location its own separate employee list in a single-tenant partition for the Employee table.

When created by the DBA or tenant administrator, a tenant group is automatically assigned a group-ID which is unique for the database. The DBA or tenant administrator can also assign a user-friendly name to the tenant group, such as "Warehouse Inventory."

When creating a tenant group, it is possible to assign both pre-existing tenants and newly created tenants as members. If a tenant who is being assigned to a group is pre-existing, meaning it has already been using its own table partition to store the same kind of data that will be stored in the tenant group partition, then as part of group setup it will be necessary to take steps to migrate the pre-existing data into the shared group partition. To avoid this migration requirement, in practice it is better to create a group before creating the tenants that will become part of the group.

Deleting a tenant who participates in a multi-tenant group has no effect on the group's table partition. Even deleting **all** of the tenants who are part of a group leaves the group partition itself, and whatever data it contains, intact. The data associated with the group will be deleted only when the group itself is deleted. The reason for this becomes clear if we consider the example of a regional tax table. If the individual tenants who are part of a tenant group accessing the tax data are all suddenly relocated to a different region, there is no need to delete the tax table information for the region that has been vacated. This is partly due to the fact that other tenants may someday come along who need to access that tax information once again.

As mentioned earlier, one possible use of groups occurs where several tenants of the same parent company share the same Customer table. This is shown in Figure 6 (page 18), where Tenants 1 and 2 share the same multi-tenant Customer table partition. In addition, Tenant 3 has its own Customer table partition separate from the tenant group. Tenant 3 does not have access to Tenant 1 and 2's Customer data, nor do Tenants 1 or 2 have access to Tenant 3's Customer data.

When each tenant executes the exact same "FOR EACH customer: DISPLAY cust-num, name. END" statement, the data returned is shown by the corresponding brackets. Note that Tenant 1 and Tenant 2 both retrieve the same set of customer records, as would be expected since they are sharing the same table partition.

4. Features and Enhancements to Support Multi-Tenancy

Having covered the architectural aspects of multi-tenancy, we can now explore the two tenant access paradigms supported in OpenEdge 11.0: simple multi-tenancy and multi-tenant groups.

Authentication

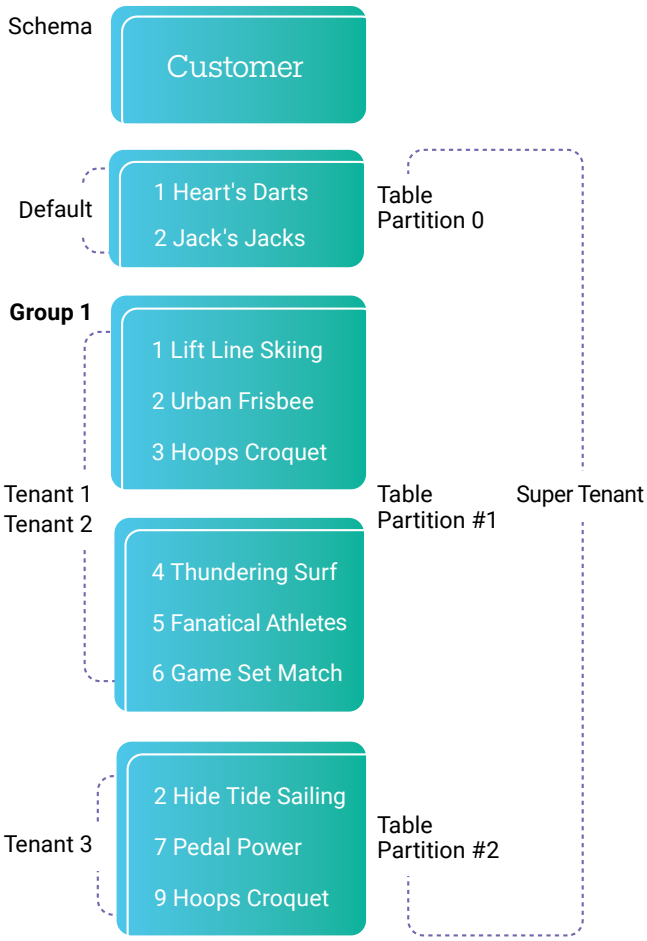
In order to access a tenant partition in a multi-tenant table, the tenant-ID of the current user must be asserted in the CLIENT-PRINCIPAL object associated with the OpenEdge session. There are two ways that the CLIENTPRINCIPAL object can be filled in: (1) the application can programmatically do it, or (2) it can be done automatically by authenticating the user to the _user table.

Whether you choose to use the _user table for authentication and have the CLIENT-PRINCIPAL object automatically filled in, or authenticate the user programmatically and fill in the CLIENT-PRINCIPAL object as part of the application, is a decision that must be made by the application developer. Further consideration of this topic is beyond the scope of this document.

For those developers who are using the _user table for authentication, the _user table has been extended to include tenancy through a <userid>@<domain-name> pairing mechanism. The association of the domain-name to the tenant is maintained in the existing _sec-authenticationdomain table. Single sign-on, _user and application-based authentication are all supported through this mechanism.

With this arrangement, it is possible for two different tenants in the same database to have users with the same user name. This is because of the inclusion of the domain-name in the <userid>@<domain-name> pairing. The domain-name distinguishes which tenant the userid actually belongs to. The same tenant name will support different domain names and, therefore, different security domains for authentication purposes.

Figure 6: Multi-tenant Groups



The CLIENT-PRINCIPAL has also been extended to maintain tenant identity (see below). This includes appropriate accessor/manipulator methods (get and set), making the CLIENT-PRINCIPAL easier to create and maintain.

Authentication via _user or within the ABL application itself must use the CLIENT-PRINCIPAL to identify an authenticated user and assert the tenancy associated with that user.

Authentication for SQL is accomplished using the domainname extension of the userid used during authentication of <userid>@<domain-name> pair.

In addition, both ABL and SQL provide support for the following:

- Recognizing and identifying the current tenant by tenant name and tenant id
- Connection as a tenant to a database at startup and at runtime
- Connection to multiple databases establishing different tenancy
- The ability to switch tenancy at runtime

The ability to specify _user entries that are SQL-specific is also provided. This allows the CREATE USER SQL statement to create users in the _user table for authentication without affecting the existing ABL authentication mechanisms, which bypass the use of the _user table.

Client-Principal (ABL only)

The CLIENT-PRINCIPAL is a built-in ABL object that was introduced prior to OpenEdge 11.0 to establish and securely maintain a user's identity. It can integrate external authentication systems with OpenEdge and can be used to synchronize user identity among application server agents. In OpenEdge 11.0, the CLIENT-PRINCIPAL object has been extended to include tenancy information as part of the identity.

Authorization

In OpenEdge 11.0, authorization has been extended to become tenant-specific. For the ABL this means extending a table's "CAN" permissions to include the domain-name of the tenant, for example, user1@domain1. Within this paradigm it is important to realize that the user names user1, user1@ domain1 and user1@domain2 are handled as three separate and distinct users, each with its own separate and distinct permission settings.

Wild cards for the "CAN" permissions are supported to aid in assigning permissions at the user and tenant levels. This allows for easy and efficient assignment of de facto permissions (for example, *@acme.admins, !*@acme), which can be built by configuring multiple domains for a tenant or an individual user.

SQL grant permissions are extended to be tenant-aware.

Language Support (ABL and SQL)

In OpenEdge 11.0, the ABL and SQL engines retrieve and allow updates only to data that the user has permission to access based on the currently asserted tenant identity and any existing authorization setting. The multi-tenant implementation on the database side provides for so much language transparency that other than asserting the tenant identity, application code changes are not necessary to support this behavior.

The languages also feature a programmatic way of creating new tenants for a multi-tenant database as well as creating new multi-tenant tables and indexes. Through the ABL this is accomplished through the creation of schema records, by means of a straightforward extension of the existing schemadefinition APIs. In SQL this is accomplished through new extensions to its DDL.

The following query capabilities are provided:

- Identify data returned by tenant-name and tenant-id
- Restrict data returned by tenant-name and tenant-id
- Query across tenant partitions for authorized users (supertenants)
- Query capabilities for members of the same tenant group

In addition, the languages provide the ability for an authorized user (super-tenant) to store a record in a particular tenant's table partition.

Multi-Tenant Table Creation/Deletion

When tenants are defined in a database, creating a multitenant table through the SQL DDL, the ABL APIs, or the webbased data administration tool automatically instantiates data partitions for each of those tenants. Each tenant has a default storage area defined, but this definition can be overridden by the DBA per tenant. The DBA also has the capability to specify when a tenant partition for a particular multi-tenant table should not be pre-allocated for specific tenants.

This raises the question of how the runtime engine behaves in cases in which a partition has not been allocated to a tenant, but the tenant tries to access data anyway. An attempt to add data to a tenant partition which has no space allocated to it (that is, which has been defined but not instantiated) returns an error while a query returns no result but generates no warnings or errors. The DBA can later decide to instantiate the tenant partition programmatically through SQL, the ABL or through the web-based multi-tenant data administration tool, and the tenant will immediately be able to use the partition to store data. The default partition associated with a newly created multi-tenant table will be instantiated only if explicitly requested by the DBA.

Existing non-multi-tenant tables can be marked as multitenant tables. In such cases, all the data that currently resides in the table will be considered to be part of the default tenant partition of the multi-tenant table.

When dropping a multi-tenant table from the database, all tenant data partitions, local indexes and LOBs are deleted from the database and the table definition is removed from the schema.

Index and LOB creation and deletion behave in an analogous manner.

Tenant Creation/Deletion

The creation of a new tenant or the deletion of an existing tenant does not affect the operation of other tenants in the same database. In other words, this capability is a fully online operation.

By default, the creation of a new tenant in an existing multitenant database results in the creation of a new tenant partition for each existing multi-tenant table defined in the database. (The DBA can override the default behavior for particular multi-tenant tables, if desired.) The specific storage area location of table, index, and LOB data can be specified, or previously defined default locations for this new tenant are used. If the tenant is to be part of a tenant group for a particular table, then there is no need to create a new tenantspecific data partition for the tables associated with the group since the tenant will use the existing group partition for that specific table.

Dropping a tenant will result in the tenant's table, index and LOB data partitions being deleted from the database, along with tenant-specific sequence data. However, the table, index and LOB definitions maintained in the schema remain intact since dropping a tenant must not affect other tenants of the database.

Auditing

New auditing events, default policies and reports have been added in support of multi-tenancy. Aside from these, no other changes have been made to the existing auditing functionality.

Operational Maintenance Tools

Multi-tenant support from the existing database maintenance tools abides by the following rules:

- Database-wide tools continue to be database-wide (e.g., backup/restore, after-imaging)
- Offline utilities continue to be offline (e.g., db repair, roll forward)
- Object-specific tools are tenant-aware (e.g., table move, dbanalys)

Monitoring Tools

The monitoring tools that have been updated for multitenancy support are promon, VSTs, statement cache and ViewB2. Wherever an object identifier exists, it has been extended to include the object's partition id. Wherever a user identifier exists, it has been extended to include the tenant identifier.

The .lg file, user-level and object-level messages have been extended to be tenant-aware in the same manner. For example, when a user logs into the system, rather than displaying "Usr 24 set name to user1," the message now shows "user1@domain1."

Enablement Tool

The existing feature-enablement utility has been extended to enable and disable multi-tenancy on a database. Multi-tenancy for the database must be enabled explicitly. It is never enabled by default.

The conversion tool used to migrate from OE 10 to OE11 populates all schema that are necessary to support multitenancy, regardless of whether the database will actually be used with multi-tenancy enabled.

ID	Feature	Active	Details
141	Multi-tenancy	Yes	

Multi-Tenant Configuration And Data Administration Tool

A web-based, multi-tenant configuration tool has been created to manage and configure tenants of the database as well as to perform other multi-tenant data administration tasks. This tool is called the Database Administration Console and has been implemented as an extension to the existing OpenEdge Explorer. The Database Administration Console provides the following capabilities:

1. Create new tenants in the database

2. Create new multi-tenant tables in the database

3. Create new multi-tenant sequences

4. Configure storage area locations for each partition of a multi-tenant object (tables, indexes and LOBs) when creating a new tenant

5. Configure storage area locations for each new tenant partition when creating a new multi-tenant object (tables, indexes and LOBs)

6. Support .df load and maintenance

7. Present a visualization of the tenants' access paradigms in conjunction with the associated storage locations of the multi-tenant objects

8. Create new multi-tenant groups and assign tenants to those groups

9. Migrate an existing tenant into a tenant group

10. Perform data administration on multi-tenant data for dump and load

11. Manage security associated with tenants of the database, including creating and deleting domains and maintaining authentication and authorization.

12. Enable and disable a tenant's access to all their data through a tenant activation/deactivation option

13. Manage alternate buffer pool settings on a per-object, pertenant basis

14. Display and report on multi-tenant-specific schema configurations

The existing ADE Data Dictionary has been extended to provide minimal multi-tenant support. Included are support for the .df file, data dump and load, and security configuration.

Both the ADE Data Dictionary and the Progress Developer Studio DB Navigator have been enhanced to provide for the creation and identification of multi-tenant

tables and sequences. They can use only the tenant defaults for the physical location of the tenant partition.

DB Navigator is also enhanced to launch the web-based OpenEdge Explorer tool as a means of providing complete tenant administration.

Application Server Support

Tenant-awareness within the multi-tenant architecture includes the Progress® OpenEdge® Application Server (AppServer) agents and Progress® WebSpeed agents as well, ensuring privacy as well as correctness of data access. The identity of a "session" has been extended through the CLIENTPRINCIPAL to include tenant-specific information as previously described. Thus, continuation of a request from the same user/tenant session can be ascertained by the ABL application code and properly processed. These agents can also detect when a request has come in from a different tenant from before. When switching tenants within an AppServer agent, all open buffers of the previous tenant will be automatically closed, just as they are with any ABL client.

Sequences

Sequences can be configured as tenant-shared or tenantspecific. For tenant-specific sequences, the same sequence definition is available to all tenants while the values of the sequence are specific per tenant.

A specific tenant can manipulate only shared sequences or its own tenant-specific sequences. A specific tenant cannot manipulate the sequence values associated with a different tenant, while a super-tenant can access the specific sequences of any tenant. In order to enable this access to all sequences by the super-tenant, existing ABL statements related to sequences (NEXT-VALUE, CURRENT-VALUE, etc.) have been enhanced to accept a tenant-id following the sequence name.

When a tenant is dropped from the database, the instance of its tenant local sequences is also dropped from the database. The sequence definition itself, however, remains in the database since its definition is shared and presumably being used by other tenants of the database.

Index Support

One index b-tree is supported per tenant for each _ index entry defined on each multi-tenant table. A multi-tenant table is identified by _file-attribute[1] = true.

The exception to this occurs when tenants are configured into tenant groups. In this case, the tenants share the same b-tree associated with the group's one shared multi-tenant table partition.

The current index search mechanisms have not changed in OpenEdge 11.0.\

Openedge Management Support

OpenEdge Management is tenant-aware.

The trend database has been changed to support the collection of tenant-specific statistics.

Monitoring and alerts have been updated to be tenant-aware.

Report and screen displays now include filters to restrict data by tenant.

5. Limits and Restrictions

When using Transparent Data Encryption, template records of multi-tenant tables reside in the schema area and, therefore, are not encrypted in the same manner as their associated tables. These records are considered part of the schema definitions of the tables.

No changes to the template-record location for non-multitenant tables have been made.

This implementation currently applies only to multi-tenant partitions.

Multi-tenant tables and their associated objects are available only in Type II storage areas.

Multi-tenant tables are not available within the temp-table database.

A maximum of 32,767 tenants is supported.

A maximum of 32,767 multi-tenant groups is supported.

6. Compatibility

Multi-tenancy does not in any way affect the limits on versionto- version compatibility of the OpenEdge product. The same rules and restrictions apply. This includes the stipulation that there is no backward compatibility for previous OpenEdge releases to access multi-tenant data, nor can a multi-tenant database be invoked by a previous release of OpenEdge.

Remote client connections must be run with the same OpenEdge version to have access to multi-tenant tables.

About Progress

Progress (NASDAQ: PRGS) is a global leader in application development, empowering the digital transformation organizations need to create and sustain engaging user experiences in today's evolving marketplace. With offerings spanning web, mobile and data for on-premises and cloud environments, Progress powers startups and industry titans worldwide, promoting success one customer at a time. Learn about Progress at www.progress.com or 1-781-280-4000.

Worldwide Headquarters

Progress, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280-4000 Fax: +1 781 280-4095
On the Web at: www.progress.com
Find us on facebook.com/progresssw twitter.com/progresssw youtube.com/progresssw
For regional international office locations and contact information, please go to www.progress.com/worldwide

Rev 16/05 | 111116-0004

Progress®