Progress®



OpenEdge® Getting Started:
Multi-tenancy Overview

OpenEdge®

Table of Contents

Copyright....5

Preface....7

Purpose....7
Audience....7
Organization....8
Using ABL documentation....8
References to ABL compiler and run-time features....8
References to ABL data types....8
Typographical conventions....9
OpenEdge SQL syntax conventions....10
Examples of syntax descriptions....10
Long syntax descriptions split across lines....12
Complex syntax descriptions with both required and optional elements....12
Examples of SQL syntax diagrams....13
Long SQL syntax descriptions split across lines....14
OpenEdge messages....14
Obtaining more information about OpenEdge messages....15

Chapter 1: Introducing multi-tenancy....17

What is multi-tenancy?....17
What are the advantages of multi-tenancy?....18
What is a tenant?....18
How does data access work?....19
Partition allocation....19
Table types....21
What is a multi-tenant group?....23

Chapter 2: Multi-tenant feature capabilities....25

Database Administration Console....25
Data Dictionary support for multi-tenancy....28
Database Administration....28
Database utilities....28
Auditing....30
Progress Developer's Studio for OpenEdge support....31

Chapter 3: Multi-tenancy and ABL....33

ABL Support....33

Running applications in a multi-tenant database environment....34
Coding for super-tenant access....34
Using multi-tenant sequences with shared tables....37
ABL API for multi-tenant maintenance....38
DataAdminService operations....38
Entity interface members....38
Further information....39

Chapter 4: Multi-tenancy and OpenEdge SQL....41
OpenEdge SQL support....42
DBA use cases....42
Define a multi-tenant table....43
Define a multi-tenant sequence....43
Define a multi-tenant Index....43
Alter a regular table to a multi-tenant table....43
Alter a multi-tenant sequence....43
Create a tenant....44
Create a super tenant....44
Alter a tenant....44
Delete a tenant....44
Regular tenant use case (accessing data)....44
Super tenant use cases....45
Access data as a super tenant....45
Join multi-tenant tables....46
Insert rows into a multi-tenant table....46
Move rows between partitions in a multi-tenant table....46
Update rows of a tenant partition....47
Delete rows of a tenant partition....47
View tenant specific information....47
SQL utility use cases....47
Dumping tenant data as a regular tenant....47
Dumping all tenant data as a super tenant or a DBA....48
Loading tenant data as a regular tenant....48
Loading tenant data as a super tenant or a DBA....48
Writing schema definitions for a multi-tenant table....48

Chapter 5: Multi-tenancy documentation map....49
Additional multi-tenant information....49

Glossary....51

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- Examples of SQL syntax diagrams
- OpenEdge messages

Purpose

The purpose of this book is to provide an introduction to OpenEdge® multi-tenancy. It will provide concepts and overview, definitions and terms, and pointers to other more detailed documentation.

Audience

This books is intended for anyone interested in a high-level understanding of OpenEdge multi-tenancy.

Organization

Introducing multi-tenancy on page 17

Introduces multi-tenancy, discusses advantages, and the basics of how OpenEdge multi-tenancy works.

Multi-tenant feature capabilities on page 25

Introduces the feature capabilities and some of the tools created or enhanced for managing multi-tenant databases.

Multi-tenancy and ABL on page 33

Discusses the changes in ABL that support multi-tenancy.

Multi-tenancy and OpenEdge SQL on page 41

Discusses the changes in OpenEdge SQL that support multi-tenancy.

Multi-tenancy documentation map on page 49

Provides a map to the many other locations in the documentation that discuss multi-tenant topics in greater detail.

Using ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is *ABL (Advanced Business Language)*. With few exceptions, all keywords of the language appear in all `UPPERCASE`, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates go to https://docs.progress.com.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the *ABL Virtual Machine (AVM)*. When the documentation refers to ABL source code compilation, it specifies *ABL* or *the compiler* as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies *the AVM* as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever *integer* appears, this is a reference to the INTEGER or INT64 data type.
- Wherever *character* appears, this is a reference to the CHARACTER, LONGCHAR, or CLOB data type.
- Wherever *decimal* appears, this is a reference to the DECIMAL data type.
- Wherever *numeric* appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to built-in class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
Fixed width	A fixed-width font is used in syntax, code examples, system output, and file names.
Fixed-width italics	Fixed-width italics indicate variables in syntax.
Fixed-width bold	Fixed-width bold italic indicates variables in syntax with special emphasis.
UPPERCASE fixed width	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except DO, FOR, FUNCTION, PROCEDURE, and REPEAT end with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon.

Convention	Description
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

OpenEdge SQL syntax conventions

The following syntax conventions are used for OpenEdge SQL:

Convention	Description
`lowercase`	Lowercase words are source language elements that are not SQL keywords.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

In this example, `ACCUM` is a keyword, and *`aggregate`* and *`expression`* are variables:

Syntax

```
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

Syntax

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

```
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

```
{ &argument-name }
```

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

In this example, you must specify MESSAGE and at least one *expression* or SKIP [(*n*)], and any number of additional *expression* or SKIP [(*n*)] is allowed:

Syntax

```
MESSAGE { expression | SKIP [ ( n ) ] } ...
```

In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

Syntax

```
{ include-file
    [ argument | &argument-name = "argument-value" ] ... }
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, WITH is followed by six optional items:

Syntax

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more *field* entries or one *record*. Options available with *field* or *record* are grouped with braces and brackets:

Syntax

```
ASSIGN   { [ FRAME frame ] { field [ = expression ] }
           [ WHEN expression ] } ...
       | { record [ EXCEPT field ... ] }
```

Examples of SQL syntax diagrams

In this example, GRANT, RESOURCE, DBA, and TO are keywords. You must specify RESOURCE, DBA, or both, and at least one *user_name*. Optionally you can specify additional *user_name* items; each subsequent *user_name* must be preceded by a comma:

Syntax

```
GRANT { RESOURCE, DBA } TO user_name[, user_name] ... ;
```

This excerpt from an ODBC application invokes a stored procedure using the ODBC syntax { call *procedure_name (param)* }, where braces and parentheses are part of the language:

Syntax

```
proc1( param, "{ call proc2 (param) }", param);
```

In this example, you must specify a *table_name*, *view_name*, or *synonym*, but you can choose only one. In all SQL syntax, if you specify the optional *owner_name* qualifier, there must not be a space between the period separator and *table_name*, *view_name*, or *synonym*:

Syntax

```
CREATE [ PUBLIC ] SYNONYM synonym
  FOR [ owner_name.] {table_name | view_name | synonym} ;
```

In this example, you must specify *table_name* or *view_name*:

Syntax

```
DELETE FROM [owner_name.] {table_name | view_name}
  [ WHERE search_condition] ;
```

In this example, you must include one expression *(expr)* or column position *(posn)*, and optionally you can specify the sort order as ascending (ASC) or descending (DESC). You can specify additional expressions or column positions for sorting within a sorted result set. The SQL engine orders the rows on the basis of the first *expr* or *posn*. If the values are the same, the second *expr* or *posn* is used in the ordering:

Syntax

```
ORDER BY {expr|posn} [ ASC | DESC ]
  [ , [{expr|posn} [ ASC | DESC ]] . . . ]
```

Long SQL syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, CREATE VIEW is followed by several optional items:

Syntax

```
CREATE VIEW [owner_name.] view_name
  [ ( column_name[, column_name] . . . ) ]
  AS [ ( ] query_expression[ ) ] [ WITH CHECK OPTION ] ;
```

OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help** > **Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help** > **Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the **HELP** key or **F1**.

On UNIX platforms, use the OpenEdge `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   ```
   OpenEdge-install-dir/bin/pro
   ```

2. Press **F3** to access the menu bar, then choose **Help** > **Messages**.
3. Type the message number and press **ENTER**. Details about that message number appear.
4. Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose **File** > **Exit**.

1

Introducing multi-tenancy

This chapter introduces OpenEdge multi-tenancy.

For details, see the following topics:

- What is multi-tenancy?
- What are the advantages of multi-tenancy?
- What is a tenant?
- How does data access work?
- What is a multi-tenant group?

What is multi-tenancy?

Multi-tenancy is a term applied to a multi-tenant application environment. It encompasses multi-tenant databases, tenant-aware tools, and applications.

A multi-tenant database is a database that provides support for a number of separate and distinct sets of users, referred to as tenants. All tenants share the same database schema. However, each set of tenant users accesses its own instance (partition) of a given table (tenant data) unless the table is shared by some or all tenants in the database (shared data). This access is transparent for each tenant. That is, each tenant accesses its own and shared tenant data as if they had access to the entire database, and each tenant accesses the database without necessarily having knowledge of any other tenants and their data. In fact, a tenant cannot access another tenant's data at all.

As an enabling technology, multi-tenant databases provide benefits to a SaaS provider by reducing database administration overhead and increasing resource sharing. Two key advantages of the OpenEdge implementation are that:

- Tenant support is implemented in the database, removing the requirement that tenancy be managed by the application.
- Database utilities and tools are tenant-aware.

Although there are many aspects of multi-tenancy, by implementing tenant support in the database layer, once the tenant identity is asserted, the application code that tenants run when using a multi-tenant database is exactly the same as application code running in a non-tenant configuration. For application users (tenants), once access to the database is authenticated, multi-tenancy is transparent. For application developers, there are very few changes that need to be made to existing applications to support a multi-tenant database. The changes are primarily confined to the areas of security, authorizing users (signing into the application), and database connections. Some applications may have intermingled these types of tasks (security and database connections) into their business logic; however, for most applications, user authentication and security are handled as an independent step when a user initially signs into the application, and the other aspects of an existing application will not need any changes.

For database administrators, many aspects of maintaining an OpenEdge RDBMS are unchanged when a database is enabled for multi-tenancy. Utilities that operate on a database-wide scale, such as PROBKUP, are not changed by multi-tenancy. Database utilities and tools that operate on an object level are tenant aware, that is, the utilities can perform maintenance on tenant-specific data. For example, index maintenance, data dump and load, and object moves, can take place at the tenant level.

Database multi-tenancy is available to customers with an Enterprise RDBMS and an add-on product called OpenEdge Multi-tenant Tables, for deployment. For development, customers with a Development Server license can create and configure multi-tenant databases and develop multi-tenant applications.

What are the advantages of multi-tenancy?

There are many advantages to implementing a multi-tenant application environment. A multi-tenant application can provide savings by reducing development and deployment costs to companies that develop applications. These savings can be passed on to customers – increasing competitive advantages for all parties involved.

Savings created by multi-tenancy come from sharing the same resources with multiple tenants. Sharing resources provides a way for an application vendor to create and maintain resources once for all customers, which can result in significant savings.

For example, one benefit a multi-tenant database provides is that the database schema is shared. One copy of the schema is available to all tenants. Schema changes only need to be made once for all tenants. This allows changes to be rolled out faster. Another benefit of the shared schema is that it allows one version of the r-code to be shared by all tenants. Through centralizing resources, all customers are maintained at the same version, reducing the overhead an application provider would otherwise encounter when maintaining multiple versions of an application across deployments.

What is a tenant?

A tenant is a separate and distinct set of users within a multi-tenant database. For example, a tenant can equate to one company, or a division within one company.

Access to a tenant's data is restricted to users that are authenticated to the tenant via a security domain. For information on security domains, see Security domains on page 52 for a brief discussion, or see *OpenEdge Getting Started: Identity Management* for complete details.

For tables that are defined to be multi-tenant, each tenant has its own instance of the multi-tenant table (unless a partition for that table has explicitly not been allocated for that tenant). Tenants have access to the data in the instances designated for that specific tenant as well as to tables that are shared.

Two special cases of tenant data access are as follows:

- **Super-tenant users** — Super-tenants users (or simply super tenants) can access all of the data in the database, typically for maintenance purposes.
- **Tenant groups** — Groups allow more than one tenant to access the same data for an instance of a table.

How does data access work?

There are two types of users that have access to multi-tenant databases: regular tenants and super tenants.

Regular tenants are further classified as the default tenant and regular named tenants. When a multi-tenant database is created, a default tenant is automatically created. When a user signs-in with a user name that does not contain a domain, the only tables available for data access are those defined with a default tenant partition, and any shared tables.

Regular named tenants are tenants that are explicitly defined and named by the database administrator. After a named tenant is defined, at least one domain and one user for that domain must be defined in order to provide access to tenant data. Users signed in to a regular tenant have access to the tenant's data as well as data in shared tables in the database.

The other type of tenant is a super tenant. Just like regular named tenants, a properly defined super tenant must have a least one domain and one user assigned to it in order to provide data access. Logging in as a super tenant provides access to all of the data in the database.

All data access is restricted by any runtime authorization settings in addition to tenancy. Each regular tenant is a separate organizational entity that is unrelated to the other tenants. When users sign-in, they specify both a user name and a domain name. The concept of a domain name exists in OpenEdge prior to OpenEdge 11, but in this release it takes on increased prominence. If no domain is specified, internally there is a blank domain to which the user is assigned, and the user is connected to the default tenant.

For access to non-multi-tenant data in legacy applications, no modifications are required to specify a domain since those users use the blank domain by default. Users who are logging in to use data for a regular tenant must specify a domain. The format is the user name and domain name separated by an @, for example `joe@AcmeCorp`.

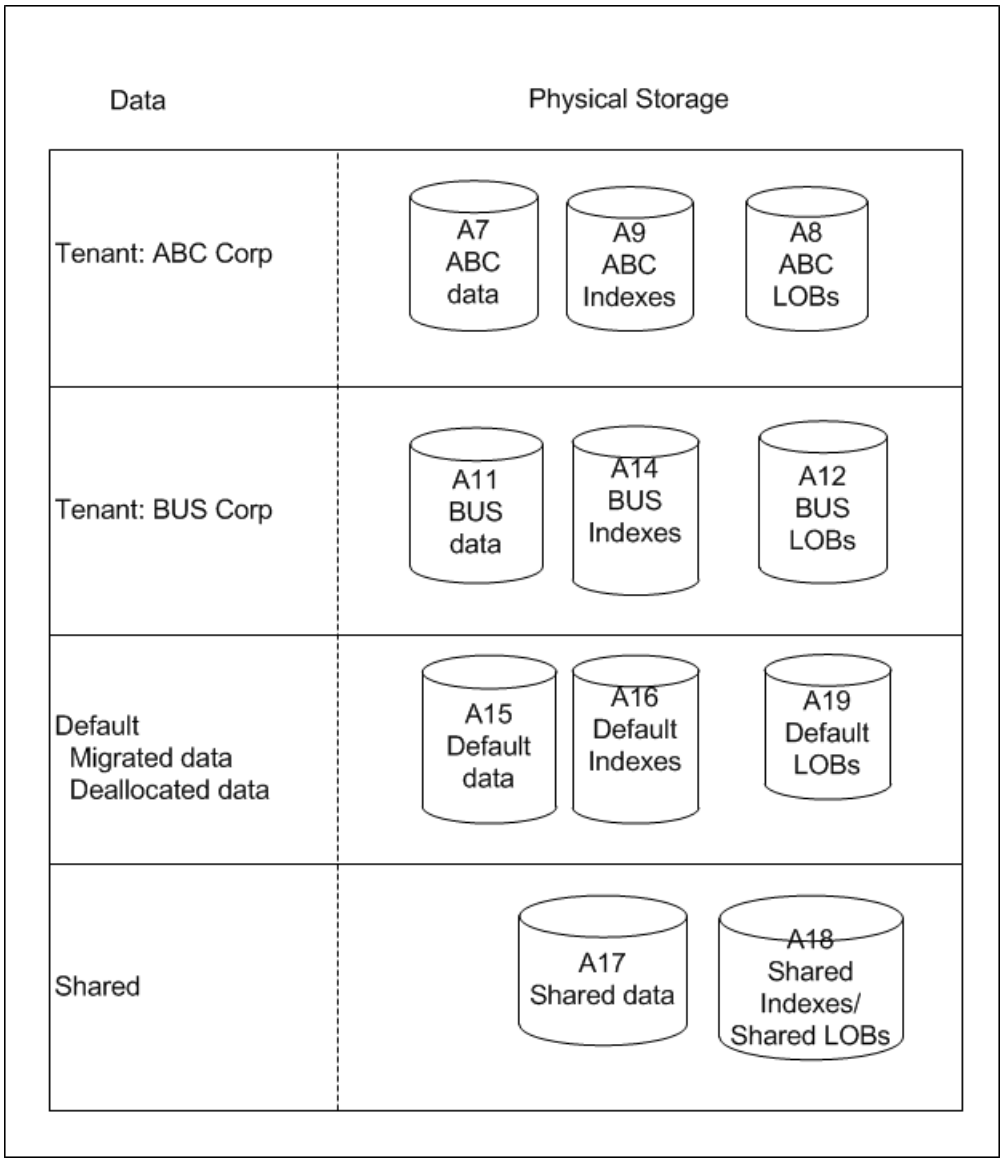
A domain is always associated with one tenant. A user has rights to that tenancy after they are authenticated to the domain, since they then have asserted their tenant identity for that tenancy.

Partition allocation

Partitions keep data physically separate for each tenant. Partitions can exist in the same storage area or different storage areas. All multi-tenant objects are required to reside in a Type II area. If an object has an allocation state that indicates not to allocate data, then the partition is not created. Additionally, access to a partition can be disabled without deleting the data. For more information on Type II areas, see *OpenEdge Data Management: Database Administration.*

The following figure depicts the allocation of data objects across storage areas in a multi-tenant database with two defined tenants, ABC Corp and BUS Corp.

Figure 1: Physical storage in an multi-tenant database



The left-hand side of the figure shows the logical representation of the database, and shows two regular tenants and the default tenant. The Shared section represents data that all tenants can access. The Shared section contains all non-multi-tenant data as well as the database schema area.

The right-hand side of the figure shows a representation of the physical storage of the database. In this example, each tenant has a distinct area for each default area.

Note: Multi-tenant partitions **must** be stored in Type II areas. Shared data can be stored in either Type I or Type II areas. The schema area is **always** a Type I area.

The following figure depicts data access. Users for a specific domain can access the data for the tenant associated with the domain, and the shared data. Users who do not specify a domain are assigned to the default tenant, and can only access the default tenant and shared data.

Figure 2: Data access

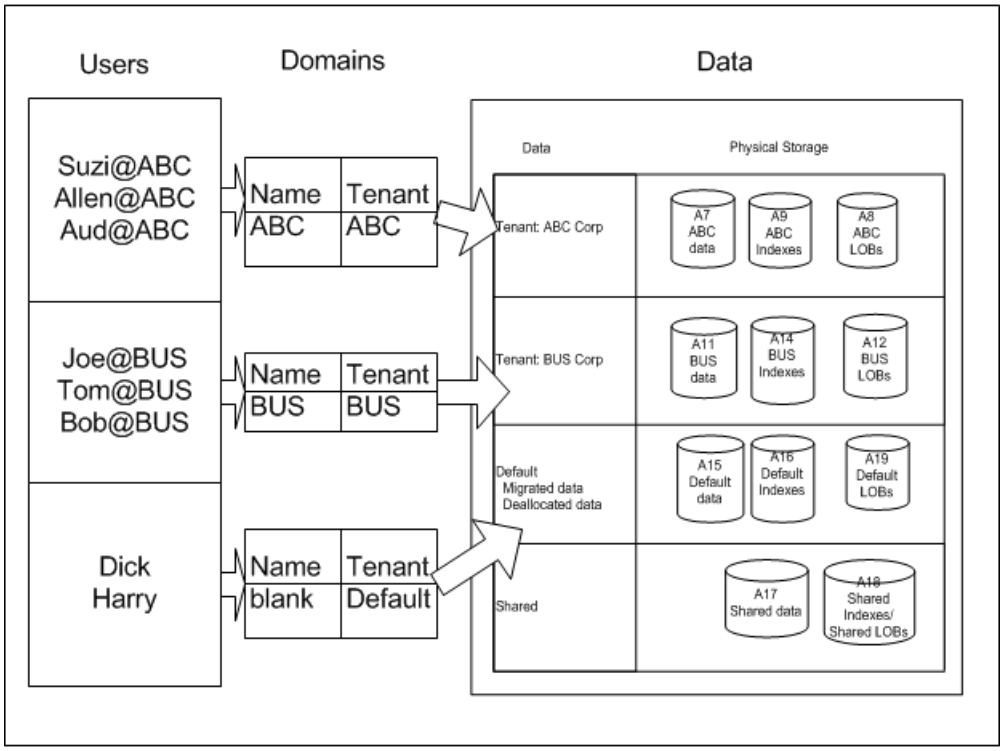


Table types

A multi-tenant database can contain the following two types of tables:

- **Shared tables** — Define tables that are not multi-tenant. Every non-multi-tenant table is accessible by all users, regardless of tenancy. Shared tables are just like the tables in all OpenEdge databases prior to OpenEdge 11, and defined data access privileges on shared tables still apply.
- **Multi-tenant tables** — Define tables where each tenant has its own set of data. Regular tenant users cannot access the data of other tenants. Multi-tenant tables are accessible only to users in domains defined for a given tenant and by super tenants.

For a table to be multi-tenant, a database administrator has to enable multi-tenancy for the table either through the Data Dictionary or the Database Administration Console. Data for multi-tenant tables is stored in partitions. Each tenant, including the default tenant, can optionally have a the data partition for each multi-tenant table allocated or not.

Multi-tenant objects

An *object* in a multi-tenant context is either a table, or an index, or a LOB.

An *object type* is the type of a specific object: table, index or LOB.

An *object instance* exists when space is allocated.

Object allocation

Allocation is done for each multi-tenant database object for each tenant, except when (optionally) set to not allocate. A storage object definition is created even if space for an object is not allocated. The space allocated for an object is referred to as a *partition*.

There are three types of multi-tenant database objects: tables, indexes defined for multi-tenant tables, and LOBs (CLOBs and BLOBs) defined for multi-tenant tables. As a unit, the database objects defined for a table are known as a *table instance*.

Allocation states

The allocation state for an object is either allocated or not allocated. For a table instance either all objects that are part of the table instance are allocated, or none of the objects are allocated.

ABL supports three allocation states:

- **Immediate allocation** — Space is allocated at the time an object is created.
- **Delayed allocation** — Space is not allocated. Delayed allocation indicates that customizations to the storage area are required before allocation can occur, and that allocation will happen in a separate step. For example, delayed allocation is required if the tenant will be included in a group.
- **Do not allocate** — Space is not allocated. Do not allocate indicates that allocation is not required or not planned for this object.

OpenEdge SQL supports two allocation states:

- **Allocated** (default) — Allocated implies immediate allocation using the default storage area.
- **Not allocated** — SQL syntax `NO SPACE.`

A database administrator can change the allocation state from not allocated to allocated at a future point, but once space is allocated it is not possible to change the allocation state (without first deleting the object and the data for the object).

Multi-tenant table instance

A multi-tenant table instance summarizes many of the previously defined terms.

Each multi-tenant table can have a multi-tenant table instance for each tenant. A table instance consists of object instances defined in the schema for a particular table definition.

The number of objects can grow very quickly. For example if you have 10 tenants, and have 10 multi-tenant tables defined, and each table has 3 indexes and 3 LOB fields, you will have 700 objects. That is:

```
10 tenants * (10 tables * (7 objects per table) ) = 700 objects.
```

Configuring and maintaining the objects of a multi-tenant database is simplified with the Database Administration Console of OpenEdge Management and OpenEdge Explorer. For more information on the Database Administration Console, see *OpenEdge Management and OpenEdge Explorer: Getting Started with Multi-tenancy*.

Data partitions

A data *partition* is space in a database area to hold the data for a database object. Data partitions exist for each allocated table and each index and LOB for a table. Data partitions for a multi-tenant table instance can be stored in the same or separate Type II storage areas. Individual partitions cannot span a storage area.

Default storage area

Each tenant defines a default storage area for each database object type; there is a default data area, a default index area, and a default LOB area.

Partitions are created and allocated for objects in the default storage areas for each tenant based on object type, unless the object's definition is changed to be allocated in a non-default area at creation time.

Multi-tenant sequences

The purpose of sequences is to provide a unique value, most often used as a value for a unique key on an index.

When working with multi-tenant databases, there are two types of sequences available:

- **Shared global sequences** — Shared sequences are non-multi-tenant sequences and existed prior to OpenEdge 11. Shared sequences are database-wide sequences. They provide one current value regardless of the tenant.
- **Multi-tenant sequences** — Multi-tenant sequences are sequences that also have a database wide definition, but which have current values that are maintained on a per-tenant basis. That is, each tenant has its own current value for each multi-tenant sequence.

A shared sequence is useful for shared tables, where all tenants can see the table, and for tenant groups, where multiple tenants each see the same records in one partition.

When exporting values for sequences, both regular and super tenants can export values for shared sequences. However, for multi-tenant sequences, a regular-tenant user can export sequence values only for the tenant to which the user belongs; a super-tenant user can export sequence values for any tenant.

What is a multi-tenant group?

A multi-tenant group, or simply a group, is a mechanism for allowing multiple tenants to access the same data in a single multi-tenant table. Each group is defined for one specific table. There can be multiple groups defined for the same table, and multiple tables that are accessed by different multi-tenant groups, but a specific tenant group applies only to one multi-tenant table.

The data for a multi-tenant group is stored in the same way as it is for any multi-tenant table instance. A group has a partition for each object of the multi-tenant table instance, for the table data, indexes, and LOBs associated with that table definition.

When data is added to a tenant group, that data is stored in the group's partitions, and shared by all tenants who are members of the group. This is different from the way tenant data is handled when there is no group, since the data is shared among the tenants of the group rather than being private to an individual tenant. The stored data is owned by the group, and not by any individual tenant or tenants.

Consequently, even if there are no tenants associated with a group, any data that had been stored in the group's table partitions continues to exist in the database until the group itself is removed.

While a tenant can belong to only one group for any given table, a table can have multiple groups defined simultaneously. A tenant can belong to zero or one group for each table.

A tenant can be added to a group with the Database Administration Console, provided that the tenant does not have its own data currently allocated for the table where it intends to become a member of a group. If tenant partitions have been allocated for the table, then the partitions must be deleted before the tenant can join a group.

Removing a tenant from a group automatically results in the tenant having its own partitions for that table, based on the defaults that are set in the tenant record and allocation setting.

2

Multi-tenant feature capabilities

This chapter highlights the capabilities of the tools and processes supporting multi-tenancy.

Note: Programming capabilities associated with multi-tenancy are discussed in later chapters. For ABL, see Multi-tenancy and ABL on page 33 and for OpenEdge SQL, see Multi-tenancy and OpenEdge SQL on page 41.

For details, see the following topics:

- Database Administration Console
- Data Dictionary support for multi-tenancy
- Database Administration
- Database utilities
- Auditing
- Progress Developer's Studio for OpenEdge support

Database Administration Console

As part of OpenEdge Management and OpenEdge Explorer, the Database Administration Console provides a browser-based interface for managing all aspects of your multi-tenant database.

The Database Administration console user interface provides efficient and easy-to-use navigation for managing tenants and multi-tenant data. From the management console, you can:

- **Convert a non-multi-tenant database to one configured for multi-tenancy** — You can quickly enable multi-tenancy for a database.
- **Establish connections to multi-tenant databases** — You can set up local and remote connections to managed databases or scripted databases. These databases are listed automatically in the list of database resources in the management console.

  You can also set up unmanaged local or remote database connections. Depending on the type of connection you define, you can also edit, copy, and/or delete it.
- **Create new database tenants, delete tenants, or generate a tenant creation program** — You can create two types of tenants: a regular tenant or a super tenant. You can also delete a tenant. Alternatively, you can generate a tenant program based on preferred settings for either an existing tenant or a new tenant. You can use the resultant generated program to later create a tenant quickly, without needing to resupply the settings saved in the program. You can also create a tenant from an existing tenant.
- **Enable a table for multi-tenancy** — You can enable one or more tables for multi-tenancy and then allocate the table's partitions.
- **Create a tenant template** — You can create, view, and edit a template that includes tenant-specific details, such as default storage areas and per-object tenant mapping, so you can reuse the template to create another tenant.
- **Create a tenant group** — You can create, view, and edit a tenant group, and add one or more tenants to it. A group has a partition for a table and partitions for any of the LOB fields and indexes in the table. Each tenant in the group has shared access to the partition and its data.

  You can also generate a group creation program based on preferred settings, and later use the program to create a group quickly or that you can use to transfer a group from one database to another database.
- **Work with existing tenants and multi-tenant database objects** — You can work easily with existing tenants to accomplish various tasks related to multi-tenancy configuration, such as:
  - View an alphabetical list of all tenants in the currently selected database.
  - Access utilities by selecting a tenant.
  - View a tenant's layout, including the database object schema, tenant object areas, and object properties. Color coding in the partition layout allows you to see easily each object's allocation state.
  - You can also view tenant partition details and choose how the data is displayed: within area panels or by columns in a grid format. The grid format allows you to choose the information you want to see (partition name, area, allocation state, and/or object type) and sort the information in ascending or descending alphabetical order.
  - Decide when storage space is to be allocated for a tenant's data: immediately, delayed (for now), or not at all.
  - Specify storage area assignments for a tenant's data (tables, indexes, and LOB fields). If you choose not to allocate storage space immediately when you create a tenant, you can reorganize the layout of the multi-tenant database objects, such as moving a table from one area to another area.
  - Filter the tenant list so you can focus on specific tenants. For example, you can set up the list so that you see only regular tenants or tenants categorized by allocation rule.
  - Add one or more tenant domains by choosing from a predefined domain list.
  - Add one or more users to a tenant.
  - Search for a particular tenant.

  - Select which of a tenant's areas you want to see when you look at the tenant's area layout. By choosing the areas you want to see, known as the area working set, you can focus only on areas of current interest to you.
  - Filter the partition view by allocation state for a tenant or a tenant template.
  - Enable or disable access to a regular tenant's data.

- **Load and preview a data definition file** — You can upload a data definition file (`.df`) and preview its contents (tables, LOB fields, indexes, and sequences) before you commit the changes.
- **Load database contents** — You can load table and group contents into an active database. The database must have table definitions before you can load table or group contents. The console uses the `.d` files to load table contents from a directory you specify or to load group contents from the subdirectory for groups you specify.

Note: You must have administrator privileges to load table and group contents.

- **Dump database contents** — You can dump table or group data in a text format from the console. The console creates a text file that contains the dump data of each table or group you specify. This file has a `.d` extension and is stored in the work directory you specify. You can dump data either from a shared table, a multi-tenant table, or both. You can also dump data from a group of tenants sharing a table. You can use these `.d` files to load data into a database. The console creates a separate contents file for each table or group you dump.

  Note: You must have administrator privileges to dump table and group contents.

- **Commit and monitor database updates** — You can commit updates to the database, either from an uploaded data definition file or as a result of other changes you make.

  As changes are being saved to the database, the My Dashboard page opens to allow you to monitor the updates' progress. The page presents a task viewlet that shows the activity as the updates are being applied.

- **Filter database connections by AdminServer** — You can view all database connections for all AdminServers, or you can choose which AdminServer's connections you want to view.
- **Manage domains** — You can create and modify a domain for each tenant.
- **Manage authentication systems** — You can view, create, or delete authentication systems.
- **Establish and manage user and table security settings** — While you can create and work with users for a particular tenant, you can also create, edit, and delete users at the database level. You can define and review table and field security for the database users, and you can review a summary that provides details about the database's security. You can also configure one or more security administrators who have control over authentication and authorization.
- **Manage sequence definitions and values** — You can manage sequences and sequence values by:
  - Viewing a list of all sequences in a database. This includes both the sequences enabled for multi-tenancy and those sequences not enabled for multi-tenancy.
  - Viewing and editing the current values for multi-tenant-enabled sequences for a specific tenant.
  - Viewing and editing the current values for all tenants of a sequence.
  - Exporting sequence current values for one tenant, multiple tenants, or all tenants in a multi-tenant database.

Note: Full access to a tenant's sequences functionality requires a database connection for a user associated with either a super tenant or that specific tenant.

- **Improve database performance** — The performance of a multi-tenant database may be improved by using multi-tenant governors: Login Governor (`-nGovernor`) and Lock Governor (`-LGovernor`). These governors limit the amount of each individual tenant's allocation of shared database resources, so that one tenant does not fully control these resources at the expense of all of the other tenants. These governors apply to the default and regular tenants in a multi-tenant database. They do not apply to super-tenants. For more information on managing these governors, see *OpenEdge Management and OpenEdge Explorer: Configuring Multi-tenancy*.

Data Dictionary support for multi-tenancy

Within the Data Dictionary user interface, you can create, modify, and report on multi-tenant aspects of the database. More detailed manipulation of multi-tenant aspects of database administration is available through the Data Administration Console. For more information, see Database Administration Console on page 25

Database Administration

The multi-tenancy support in Data Administration ensures the following:

- Security Domains are defined as belonging to a Tenant in a multi-tenant database, and this information is included in dump and load.
- The built-in Security Authentication System and Authentication System Domain records are not included in data dump and the delete action and the fields that are not editable are disabled for these records in the UI.
- Multi-tenant information is included when dumping and loading data definitions.
- A super tenant can select and set their effective tenancy when dumping and loading data and sequence current values and that the dump and load utilities have defaults that make it easy to dump and load tenant data to and from subdirectories named after the tenant.
- The maintenance of alternate buffer pools does not include multi-tenant tables.
- All reports and lists include multi-tenant information where applicable.

Database utilities

Database utility changes in support of multi-tenancy fall into three categories: unchanged, indirectly changed, and multi-tenant aware.

Unchanged database utilities continue to operate across the database, and are not impacted by multi-tenancy. These utilities include:

- Backup and restore

- After-imaging and Replication
- Failover clusters
- Prostrct
- Transparent Data Encryption

Auditing is a utility that is indirectly changed. The mechanics of how auditing is managed remain unchanged, but auditing has the following changes:

- New audit policies and auditing events are added and extended to support multi-tenancy. An incremental audit policy, `multi-tenant.xml`, is included in the `%DLC%/auditing` directory. The full auditing policy will also have the incremental one appended to it.
- Audit event triggers are included in the implementation of the new enable and disable multi-tenant feature commands in PROUTIL.
- Existing database record auditing event triggers are updated to include the tenant-name information.

Tenant aware utilities allow the database administrator to execute the utilities on a per-tenant, per-group, or database-wide basis. Database utilities that can manipulate and display data based on multi-tenant characteristics are added or updated. The list of impacted utilities follows:

- New PROUTIL commands
  - ENABLEMULTITENANCY
  - DISABLEMULTITENANCY
  - DEALLOCATE
  - IDXDEACTIVATE
- Modified PROUTIL commands:
  - BULKLOAD
  - DBANALYS/IDXANALYS/TABANALYS
  - DESCRIBE
  - DISPTOSSCREATELIMITS
  - DUMP/DUMPSPECIFIED
  - IDXACTIVATE
  - IDXBUILD/IDXCHECK/IDXFIX
  - IDXDEALLOCATE
  - IDXMOVE
  - LOAD
  - Create and Toss limits (SETAREACREATELIMIT, SETAREATOSSLIMIT, SETLOBCREATELIMIT,SETLOBTOSSLIMIT,SETTABLECREATELIMIT,SETTABLETOSSLIMIT)
  - TABLEMOVE
  - VIEWB2

- DBTOOL
- SQLDUMP/SQLLOAD/SQLSCHEMA
- PROMON

The OpenEdge RDBMS and utilities fully support the authentication and authorization enhancements that are part of the Identity Management enhancements to OpenEdge Release 11.0 and higher. The utilities accept the -U/-P parameters. If no user name and password are included, the utilities trust the underlying operating system authentication and authorization.

Auditing

For multi-tenant databases, certain new and existing audit policy events add a tenant or group name prefix to the information in the _Event-context field of the _aud-audit-data table.

For these multi-tenant events, where no database record operation is involved, the multi-tenant format of the _Event-context field is as follows:

```
{ T:tenant-name | G:group-name } .owner.table-name
```

To identify database records for audited database operations, the multi-tenant format is as follows:

```
{ T:tenant-name | G:group-name } .owner.table-name
CHR(6)id-fld-val [ CHR(7)id-fld-val-n . . . ]
```

`T:tenant-name`

Identifies the name of database tenant on which the audited operation occurred.

`G:group-name`

Identifies the name of tenant group on which the audited operation occurred.

`owner`

The owner of the database table on which the audited operation occurred. For ABL, this is always PUB. Otherwise, it is the table owner as defined by SQL.

`table-name`

The name of the database table on which the audited operation occurred.

`CHR(6)`

`CHR(7)`

Non-printing ASCII characters with the value of 6 to delimit an initial identifying field value, and the value of 7 to delimit each additional identifying field value (respectively).

`id-fld-val`

An initial field value that identifies the database record on which the audited operation occurred.

`id-fld-val-n`

One of zero to any number of additional field values required to identify the database record on which the audited operation occurred.

For details on the audit policy events that contain this new format, see *OpenEdge Getting Started: Core Business Services - Security and Auditing.*

Progress Developer's Studio for OpenEdge support

DB Navigator is enhanced to support the multi-tenant features in OpenEdge databases. DB Navigator now enables:

- Serving multiple users with a single OpenEdge database instance.
- Creating, editing, and identifying multi-tenant tables and sequences. Categorizing and filtering DB tables.
- Launching OpenEdge Explorer from the DB navigator perspective for configuring the database.
- Sharing data definition files (schema changes) among developers.

3

Multi-tenancy and ABL

Support for multi-tenancy with ABL is discussed in this chapter.

For details, see the following topics:

- ABL Support
- ABL API for multi-tenant maintenance

ABL Support

As noted previously (see the What is Multi-tenancy? on page 1), after converting a non-multi-tenant database to a multi-tenant database, most, if not all, of the code in an existing ABL application will work without change. When executing against a multi-tenant database, all existing ABL code that accesses database objects is implicitly tenant aware, which means that it accesses database objects that are shared (non-multi-tenant) or that belong to the tenancy of the connected user (if you have made any database objects multi-tenant).

Super-tenant users run a converted ABL application without any real tenancy of their own, which means they can access shared data and, potentially, the data for one or more **effective** regular tenants. Initially, OpenEdge sets the effective tenancy of all super-tenant users to the default tenant, whose data super tenants can access by default. To enable super-tenant access to data regardless of tenancy, ABL provides additional features that you can use in both existing and new applications.

If you plan to create tenant groups for multi-tenant tables, or use any shared (tables, and are also considering the use of multi-tenant sequences, you need to plan carefully because of the way multi-tenant sequences interact with any tables that are shared by multiple tenants.

The following sections provide a general overview of all the ABL features that multi-tenancy enables:

- Running applications in a multi-tenant database environment on page 34
- Coding for super-tenant access on page 34
- Using multi-tenant sequences with shared tables on page 37

For more information on coding for super tenancy and other issues with coding for multi-tenant databases, see the sections on multi-tenant ABL in *OpenEdge Development: Programming Interfaces.*

Running applications in a multi-tenant database environment

The primary requirements to run an application against a multi-tenant database is to configure one or more tenants in the database, and to configure at least one domain per tenant to which application users can authenticate when they log into your application. For a converted multi-tenant database, any existing application only requires the default tenant in order to execute as originally designed, and you typically add additional regular tenants to expand the application to run with totally separate sets of users, again, with little or no change to the application code.

Note: Because super tenants have no real tenancy of their own, OpenEdge initially provides effective super-tenant access to default tenant data. However, a super tenant can assume the effective tenancy of any and all regular tenants of a database (including the default tenant) at run time. For more information, see Coding for super-tenant access on page 34.

If you already have domains defined in a multi-tenant database in order to sign on users using a client-principal object, and you configure each of these existing domains with a tenant, your application might work without any further changes depending on how you initialize attributes of the client-principal. Assuming these attributes are set correctly, existing calls to the `SET-DB-CLIENT` function or the `SET-CLIENT( )` method (on the `SECURITY-POLICY` system handle) will set the tenant identity along with the user identity of any multi-tenant database connection.

If your user login procedure relies entirely on the User ID (`-U`) connection parameter to authenticate database access (at startup or using the `CONNECT` statement), as long as you ensure that users provide a user ID that includes their assigned user name and domain name in the form, *`user-name@domain-name`*, and the domain is configured for an authentication system (such as the `_User` table accounts) to which OpenEdge can authenticate database connections, your application might also work without any further changes. The main thing is to ensure that your application sets database connection identity in a manner that authenticates users with a tenant identity defined in your database.

You can design a number of different approaches (authentication models) to manage user access to OpenEdge databases. For more information on authenticating users with or without a tenant identity, see *OpenEdge Getting Started: Identity Management.*

Coding for super-tenant access

A super-tenant user has access to all the data of a multi-tenant database, including all tenant and shared data, according to the data authorization provided by the user's permission settings. Super-tenant access is enabled by configuring the user's security domain with a super-tenant identity. Data authorization settings work the same as any multi-tenant database user, with permissions for individual tables and fields set based on the user and domain name (see *OpenEdge Getting Started: Identity Management*).

The initial tenancy of any super tenant provides no access to tenant data. This means that when initially logged in, a super tenant user has implicit access only to shared database objects. To support super-tenant access to multi-tenant database objects, ABL provides features to allow a super tenant user to query and update database objects that belong to one or more regular tenants. In addition, ABL allows a super tenant user to identify the tenant and tenant group identity (if any) to which any given multi-tenant table record belongs.

These features, taken together, allow a super tenant user to perform any run-time action on database objects, regardless of tenancy, and limited only by the data authorization permissions that are defined for the given super tenant. For example, if the super-tenant user identity has only can-read permissions on a given multi-tenant table, the super tenant user can read instances of the table owned by all tenants, but cannot update any instances of the table, including the default tenant instance.

The following sections provide an overview of OpenEdge super tenant capabilities in ABL:

- Super-tenant effective tenant on page 35
- Super-tenant applications on page 35
- Super-tenant queries on page 35
- Super-tenant data updates on page 36
- Tenancy identification on page 36
- Super-tenant sequence access on page 37

Super-tenant effective tenant

A super tenant can change effective tenancy for a given database connection with the ABL built-in function, `SET-EFFECTIVE-TENANT`. This function allows the super tenant to implicitly query data for the effective tenant without having to set a new user identity for the connection. All queries with this effective tenancy return only shared data and data owned by the effective tenant.

Once a super tenant has set an effective tenancy, the super tenant has all the same behaviors and access as the regular tenant for actions such as creating, updating, and deleting data.

Super-tenant applications

Applications written prior to OpenEdge 11 have no concept of tenancy as defined by OpenEdge. Virtually all code written for them works with whatever tenancy is defined for the current database connection identity. To take advantage of super-tenancy, you can write new applications or new modules for existing applications that require super-tenant access, or use the `SET-EFFECTIVE-TENANT` built-in function to identify a tenant prior to executing an existing application.

Typically, these are applications used to maintain or monitor multi-tenant data across tenants. One example might be in a Software as a Service (SaaS) environment, where the SaaS administrator needs to collect billing information for all application tenants, and to add or update billing notices for applicable tenants.

Super-tenant queries

With can-read access, a super tenant can query data for any one or a combination of tenants using the following basic ABL query mechanisms:

- **`TENANT-WHERE` option of the record phrase** — Adds tenancy criteria to the data in a query independent of the current effective tenancy. Given the specified multi-tenant tables and joins in a query, this option can be used to qualify the tenancy of each individual record buffer, including record instances from multiple tenants. If multiple tenants are specified in a tenant group, the associated `SKIP-GROUP-DUPLICATES` option skips all but the first tenant record in any group.

- **REPOSITION statement and REPOSITION-TO-ROWID( ) method on a query handle** — Provides an option to reposition to a query row owned by a specified tenant.

Super-tenant data updates

ABL allows a super tenant to create (with can-create access) and update (with can-write or can-delete access, as appropriate) database records for specified tenants using the following elements:

- **CREATE statement and BUFFER-CREATE( ) method on a buffer handle** — For a multi-tenant table, these ABL elements provide an option for a super tenant to create a database record for a specified tenant, regardless of the effective tenancy. Without this option, these ABL elements create database records owned by the effective tenant.
- **All record update statements, such as ASSIGN, SET, UPDATE, DELETE , etc.** — Regardless of the current effective tenant, ABL remembers the tenancy of all records created for a multi-tenant database as well as the records retrieved from a multi-tenant database. When a super tenant updates database records created for multiple tenants or queried for multiple tenants, ABL ensures that all these record instances are written to or deleted from the table instances owned by the corresponding tenants.

Tenancy identification

Tenancy is identified by:

- A name that you assign when you configure the tenant in a multi-tenant database.
- A database-unique, integer ID that OpenEdge assigns when you first create the tenant. The default tenant ID is zero (`0`), all other regular-tenant IDs are positive integers, and all super-tenant IDs are negative integers.

A super tenant can return information about database connection tenancy using these ABL built-in functions:

- **IS-DB-MULTI-TENANT function** — Returns `TRUE` if a connected database is multi-tenant enabled.
- **TENANT-NAME and TENANT-ID function** — Returns the tenant identity associated with a current database connection identity.
- **TENANT-NAME-TO-ID function** — Returns the tenant ID, given the tenant name associated a current database connection identity.
- **GET-EFFECTIVE-TENANT-NAME and GET-EFFECTIVE-TENANT-ID function** — Returns the current effective tenant identity for a connected database.

Note, again, that the `GET-EFFECTIVE-TENANT-*` functions return the most recent **effective** tenancy for a database connection, where as the corresponding `TENANT-*` functions return the tenancy of the current database **connection identity**. The effective tenancy for a database connection is set **only** using the `SET-EFFECTIVE-TENANT` function, where as the connection identity can be set using any connection authentication operation. These two tenancies can be different, with the most recently set tenancy overriding the other for purposes of effective tenancy.

A super tenant can return information about the tenancy of a database record using these ABL built-in elements:

- **IS-MULTI-TENANT attribute on a buffer handle** — Returns `TRUE` if the current record referenced by the specified buffer handle is multi-tenant.
- **BUFFER-TENANT-NAME and BUFFER-TENANT-ID function** — Returns the tenant identity that owns the current record in the specified buffer.
- **BUFFER-TENANT-NAME and BUFFER-TENANT-ID attribute on a buffer handle** — Returns the tenant identity that owns the current record referenced by the specified buffer handle.

Super-tenant sequence access

Similar to multi-tenant tables, a super tenant can access a multi-tenant sequence owned by a particular tenant using the following functions:

- **`CURRENT-VALUE` and `DYNAMIC-CURRENT-VALUE` function** — Provides an option to return the current value of the multi-tenant sequence owned by a specified tenant.
- **`NEXT-VALUE` and `DYNAMIC-NEXT-VALUE` function** — Provides an option to return the next value of the multi-tenant sequence owned by a specified tenant.

However, note that if you configure, or ever intend to configure, tables with tenant groups or shared tables in a multi-tenant database, certain uses of a multi-tenant sequences can have unintended consequences, both for regular tenants and super tenant. The following section describes these potential issues.

Using multi-tenant sequences with shared tables

Tenant group tables and shared tables can be useful, but using them with sequences requires careful consideration. The challenge comes from the fact that shared tables and tables with tenant groups are shared by some or all tenants of the database, where as multi-tenant sequences have no groups and are never shared, but are always separately owned by each regular tenant in the database. This means that for each multi-tenant sequence that you define, every tenant that executes a sequence function generates and returns the same set of values. The tenant can then assign these values to a group or shared table, but with certain restrictions:

- If the field requires a unique value, as for a primary index, the AVM will raise an error when any tenant tries to assign an existing value previously assigned by another tenant.
- If the field represents a foreign key, queries on that foreign key with the same value will return multiple records from different tenants. However, information about which tenant owns which record is lost within the group or shared index on that foreign key.

Clearly, the only solution to assign sequence values to a primary key is to use a shared sequence. The best practice to avoid record ambiguity with foreign keys in group or shared tables is to always use shared sequences to assign values to foreign keys in these types of tables.

If you plan to include multiple tables with groups in a join, you might want to ensure that the groups in each table have the same tenant membership.

Groups also have identity, similar to tenants, using a name or integer ID. You might chose to implement separate shared sequences to assign key values for each group in a table. In the `CREATE` trigger, where you would typically assign the key values to a record, you can test the group identity for the record in order to determine which sequence to use to assign a key value. You can return this group identity using the following ABL elements:

- **`BUFFER-GROUP-NAME` and `BUFFER-GROUP-ID` function** — Returns the tenant group identity that owns the current record in the specified buffer.
- **`BUFFER-GROUP-NAME` and `BUFFER-GROUP-ID` attribute on a buffer handle** — Returns the tenant group identity that owns the current record referenced by the specified buffer handle.

ABL API for multi-tenant maintenance

OpenEdge provides several tools for maintaining the multi-tenant configuration of a database, including the Data Administration tool, the Database Administration Console in OpenEdge Management or OpenEdge Explorer, and the ABL API for multi-tenant maintenance. OpenEdge provides the first two tools with all the features required to configure multi-tenancy for many use cases.

If you want to build custom multi-tenant configuration tools entirely in ABL, perhaps to automate multi-tenant configurations, OpenEdge provides the ABL API. This API includes a set of ABL interfaces. These interfaces provide access to the following configuration entities for a database:

- Tenants
- Tenant groups
- Authentication systems
- Domains
- Users (in the `_User` table only)
- Areas
- Tables
- Sequences
- Partitions

For example, an `OpenEdge.DataAdmin.ITenant` instance represents a tenant.

The operations to create, retrieve, update, and delete (as appropriate) these entities are supported by a single service class: `OpenEdge.DataAdmin.DataAdminService`. Not all operations are supported for all entities. For example, the `DataAdminService` class does not create tables, but updates table configuration information, for example, to convert a shared table into a multi-tenant table.

DataAdminService operations

The configuration operations supported by the `DataAdminService` class are provided as methods. You create one instance of this class for each database you want to configure. The methods for a given instance operate on the multi-tenant configuration for the corresponding database.

These methods include factory methods to create class instances for each configuration entity you need to create, which you reference as the corresponding interface type. The remaining `DataAdminService` methods perform operations on these entity instances, whether newly created or existing instances that are returned from the database you are configuring.

Entity interface members

The entity interfaces provide properties to access elements of the entity configuration, such as the name of a tenant. The interfaces also provide methods, which perform common operations on entities, such as adding or retrieving them from entity collections, or importing or exporting entities (or entity collections) to the database as JSON objects.

Further information

For more information on the ABL API for multi-tenant maintenance, see *OpenEdge Development: Programming Interfaces.*

4

Multi-tenancy and OpenEdge SQL

OpenEdge SQL delivers a range of features supporting and enabling multi-tenancy, including:

- Multi-tenant specific SQL statements to create, alter, and drop tenants of an existing database table.
- Extended support for multi-tenancy in the OpenEdge SQL Data Definition Language (DDL) to create, alter, and drop multi-tenant enabled tables, indexes, and sequence.
- Extended support for multi-tenancy in the OpenEdge SQL Data Manipulation Language (DML) to select, insert, update, and delete to/from a multi-tenant table.

This chapter is intended for existing OpenEdge SQL users who are familiar with the OpenEdge SQL syntax and provides a brief overview of multi-tenant support, and provides use cases for OpenEdge SQL users getting started with multi-tenancy.

For more information on SQL syntax, see *OpenEdge Data Management: SQL Reference.*

For details, see the following topics:

- OpenEdge SQL support
- DBA use cases
- Regular tenant use case (accessing data)
- Super tenant use cases
- SQL utility use cases

OpenEdge SQL support

As noted previously (see What is multi-tenancy? on page 17), after converting a non-multi-tenant database to a multi-tenant database, most, if not all, of the code in an existing application will work without change. When executing against a converted multi-tenant database, all existing code that accesses database objects is implicitly tenant aware, which means that it accesses database objects that are shared (the default) or that belong to the tenancy of the connected user (if you have made any database objects multi-tenant).

Super-tenant users are allowed access to all the data in a multi-tenant database regardless of tenant ownership.

The primary requirements to run an application against a multi-tenant database is to configure one or more tenants, and to configure at least one domain per tenant to which application users can authenticate when they login to your application. For a converted multi-tenant database, any existing application only requires the default tenant in order to execute as originally designed, and you typically add additional regular tenants to expand the application to run with totally separate sets of users, again, with little or no change to the application code.

DBA use cases

The following use cases require you to be connected to an OpenEdge database as a DBA.

Connect to an OpenEdge database as a DBA, `dbauser`, using the SQL explorer tool:

```
sqlexp  -user dbauser -password abcd -db sports3000 -S  1234  -H  localhost
```

After connecting as a DBA, you can:

- Define a multi-tenant table on page 43
- Define a multi-tenant sequence on page 43
- Define a multi-tenant Index on page 43
- Alter a regular table to a multi-tenant table on page 43
- Alter a multi-tenant sequence on page 43
- Create a tenant on page 44
- Create a super tenant on page 44
- Alter a tenant on page 44
- Delete a tenant on page 44

Define a multi-tenant table

To define a table, `mtcustomer`, as a multi-tenant table:

```
CREATE TABLE pub.mtcustomer (cust_num    integer, hire_date   date)
MULTI_TENANT;
```

Define a multi-tenant sequence

To define a sequence, `mtcustomer_sequence`, as a multi-tenant sequence:

```
CREATE SEQUENCE pub.mtcustomer_sequence
     START WITH 100,
     INCREMENT BY 1,
     NOCYCLE
     MULTI_TENANT
     ;
```

Define a multi-tenant Index

An index on a multi-tenant table is automatically a multi-tenant index. To define an index, `mtcustomer_index`, as a multi-tenant index:

```
CREATE INDEX mtcustomer_index ON mtcustomer (cust_num)
       FOR TENANT DEFAULT USING INDEX AREA "Misc_Index_Area"
;
```

Alter a regular table to a multi-tenant table

To alter a regular table, `customer`, to a multi-tenant table:

```
ALTER TABLE pub.customer SET MULTI_TENANT;
```

Alter a multi-tenant sequence

To alter the sequence, `mtcustomer_sequence`, of a tenant, `mtAsiaPacificCust`:

```
ALTER SEQUENCE pub.customer_sequence SET CURRVAL 521 FOR TENANT mtAsiaPacificCust;
```

Create a tenant

To create a tenant, `regular_tenant`, in a multi-tenant table:

```
CREATE TENANT  regular_tenant TABLE AREA  "MMM Data area"
                              INDEX AREA  "MMM index area"
                              LOB AREA  "MMM pix area"
FOR TABLE pub.mtcustomer USING NO SPACE;
```

Create a super tenant

To create a super tenant, `super_tenant`, in a multi-tenant table:

```
CREATE SUPER_TENANT super_tenant;
```

Alter a tenant

To alter a tenant, `regular_tenant`, in a multi-tenant table:

```
ALTER TENANT regular_tenant
    FOR TABLE pub.finance_accts USING TABLE DEFAULT AREA
    FOR TABLE pub.new_customers USING TABLE AREA "MMN Data area"
                                USING INDEX AREA "MMN_Index_Area";
```

Delete a tenant

To drop a tenant, `regular_tenant`, from a multi-tenant table:

```
DROP TENANT regular_tenant;
```

Regular tenant use case (accessing data)

After connecting to an OpenEdge database as a regular tenant, accessing data is unchanged.

Connect to an OpenEdge database as a regular tenant, `regular_tenant`, using the SQL explorer tool:

```
sqlexp  -user regular_tenant@DomainOne  -password abcd -db sports3000 -S  1234  -H
localhost
```

Access data from a multi-tenant table, `mtcustomer`, as shown:

```
SELECT * FROM Pub.mtcustomer;
```

The above select statement returns the column values only from the rows in the `regular_tenant` partition.

Super tenant use cases

The following use cases require you to be connected to an OpenEdge database as a super tenant.

Connect to an OpenEdge database as a super tenant, `super_tenant`, using the SQL explorer tool:

```
sqlexp  -user super_tenant@DomainOne -password abcd -db sports3000 -S  1234  -H localhost
```

After connecting as a super tenant, you can:

- Access data as a super tenant on page 45
- Join multi-tenant tables on page 46
- Insert rows into a multi-tenant table on page 46
- Move rows between partitions in a multi-tenant table on page 46
- Update rows of a tenant partition on page 47
- Delete rows of a tenant partition on page 47
- Delete rows of a tenant partition on page 47

Access data as a super tenant

To access data from a multi-tenant table, `mtcustomer`:

```
SELECT * FROM Pub.mtcustomer;
```

The above select statement returns the column values from the rows in the all the tenant partitions.

Join multi-tenant tables

To join three multi-tenant tables, `pub.mtcustomer, pub.mtorder,` and `pub.mtorderline`:

```
SELECT
    tenantName_tbl (c) as ten_name,
    c.name as c_name,
    COUNT (*)
FROM
    Pub.mtcustomer AS c
    INNER JOIN pub.mtorder AS o
    ON tenantId_tbl (c) = tenantId_tbl (o)
    AND c.custnum = o.custnum
    INNER JOIN pub.mtorderline AS ol
    ON tenantId_tbl (o) = tenantId_tbl (ol)
    AND o.ordernum = ol.ordernum
GROUP BY
    ten_name, c_name;
```

Insert rows into a multi-tenant table

To insert a new row in a tenant partition of the tenant, `SNCSoftware`, in the multi-tenant table, `mtcustomer`:

```
INSERT INTO pub.mtcustomer TENANT SNCSoftware
(custnum, name)
VALUES
(9999, 'West Side Sports');
```

Move rows between partitions in a multi-tenant table

To move the rows from the DEFAULT partition of a multi-tenant table, `mtcustomer`, and distribute it to the tenants, `SNCSoftware` and `OEDProducts`:

```
INSERT INTO pub.mtcustomer TENANT SNCSoftware
SELECT *
FROM pub.mtcustomer AS mtc
WHERE tenantName_tbl (mtc) = 'default'
AND MOD (custnum, 2) = 0;

INSERT INTO pub.mtcustomer TENANT OEDProducts
SELECT *
FROM pub.mtcustomer AS mtc
WHERE tenantName_tbl (mtc) = 'default'
AND MOD (custnum, 2) <> 0;

DELETE
FROM pub.mtcustomer AS mtc
WHERE tenantName_tbl (mtc) = 'default';
```

Update rows of a tenant partition

To update rows of a tenant, `SNCSoftware`, in a multi tenant table, `mtcustomer`:

```
UPDATE pub.mtcustomer
SET postalcode = '99999'
WHERE custnum = 1428 AND tenantName_tbl (pub.mtcustomer) = 'SNCSoftware';
```

Delete rows of a tenant partition

To delete rows of a tenant, `SNCSoftware`, in a multi-tenant table, `mtcustomer`:

```
DELETE FROM mtcustomer
WHERE custnum = 1428 AND tenantName_tbl (pub.mtcustomer) = 'SNCSoftware';
```

View tenant specific information

To view tenant-specific information for all the tenants in a multi-tenant table, `mtcustomer`:

```
SHOW ALL TENANT FOR TABLE pub.mtcustomer;
```

SQL utility use cases

You can use the SQL utilities, SQLDUMP, SQLLOAD, and SQLSCHEMA on a multi-tenant database, as demonstrated in the following sections:

- Dumping tenant data as a regular tenant on page 47
- Dumping all tenant data as a super tenant or a DBA on page 48
- Loading tenant data as a regular tenant on page 48
- Loading tenant data as a super tenant or a DBA on page 48
- Writing schema definitions for a multi-tenant table on page 48

Dumping tenant data as a regular tenant

To dump the data to an SQL dump file for a tenant partition of a regular tenant, `regTenantUser`:

```
sqldump -u regTenantUser@OpenEdgeA -a regTenant -t mttab1
progress:T:localhost:9999:sports3000
```

Dumping all tenant data as a super tenant or a DBA

To dump the data to an SQL dump file for all the tenant partitions of a multi-tenant table:

```
sqldump -u dbaUser -a dba -t mttab1 progress:T:localhost:9999:sports3000
```

For this operation, the user, `dbaUser`, must be mapped to a DBA or a super tenant.

Loading tenant data as a regular tenant

To load the data from a formatted file into a tenant partition of a regular tenant, `regTenantUser`:

```
sqlload -u regTenantUser@OpenEdgeA -a regTenant -t mttab1 progress:T:localhost:9999:mtdb
```

Loading tenant data as a super tenant or a DBA

To load the data from a formatted file into all the tenant partitions of a multi-tenant table:

```
sqlload -u dbaUser -a dba -t mttab1 progress:T:localhost:9999:mtdb
```

For this operation, the user, `dbaUser`, must be mapped to a DBA or a super tenant.

Writing schema definitions for a multi-tenant table

To write schema definitions for a multi-tenant table:

```
sqlschema -u mtuser@mtdomain -a mtuser -t pub.acct_payable
progress:T:localhost:2222:sports3000
```

5

Multi-tenancy documentation map

This chapter provides a documentation map that directs you to additional sources of more detailed information regarding multi-tenancy in OpenEdge.

For details, see the following topics:

- Additional multi-tenant information

Additional multi-tenant information

The following table provides pointers to the many other locations in the OpenEdge documentation where you can find specific details regarding multi-tenancy enhancements.

Table 1: Additional information

For additional information on...	See...
Database administration	*OpenEdge Data Management: Database Administration*
Configuring multi-tenancy with the Database Administration console in OpenEdge Management and OpenEdge Explorer	*OpenEdge Management and OpenEdge Explorer: Getting Started with Multi-tenancy* and the OpenEdge Management and OpenEdge Explorer online Help

For additional information on...	See...
Enhancements in ABL to support multi-tenancy and multi-tenant maintenance	• *OpenEdge Development: ABL Reference* • *OpenEdge Development: Programming Interfaces*
Enhancements in OpenEdge SQL to support multi-tenancy	• *OpenEdge Data Management: SQL Development* • *OpenEdge Data Management: SQL Reference*
Identity management and domain configurations	*OpenEdge Getting Started: Identity Management*
Support in Progress DB Navigator for multi-tenancy	The Progress DB Navigator Guide online Help

Glossary

Data partitions

A data *partition* is space in a database area that holds the data for a database object. Data partitions exist for each allocated table and each index and LOB for a table. Data partitions for a multi-tenant table instance can be stored in the same or separate Type II storage areas. Individual partitions cannot span a storage area.

Default storage area

Each tenant defines a default storage area for each database object type, meaning there is a default data area, default index area, and default LOB area.

Partitions are created and allocated for objects in the default storage areas for each tenant based on object type, unless the object's definition is changed to be allocated in a non-default area at creation time.

Domain access code

The domain contains an access code that is used to establish trust relationships between members of a security domain. An access code is assigned when a domain is created. The access code for a security domain is required in order to authenticate a user to a domain. The access code is the same for all users in a given domain.

Group/multi-tenant group

A multi-tenant group, or simply a group, is a mechanism for allowing multiple tenants to access the same data in a multi-tenant table. Each group is defined for one specific table. There can be multiple groups defined for the same table, but a group cannot be defined for more than one table.

The data for a multi-tenant group is stored in the same way as it is for any multi-tenant table instance. A group has a partition for each object of the multi-tenant table instance that is for the table data and each index and LOB associated with that table.

When data is added to a tenant group, that data is stored in the group's partitions, and shared by all tenants who are members of the group. This is different than the way tenant data is handled when there isn't a group, since the data is shared among the tenants of the group rather than being private to an individual tenant. The stored data is 'owned' by the group, and not by any individual tenant or tenants. This is an important distinction, because even if there are no tenants associated with the group, any data that had been stored in the tenant group's partition continues to exist in the database until the group itself is removed.

While a tenant can only belong to one group for any given table, a table can have multiple groups defined simultaneously. A tenant can belong to zero or one group for each table.

A tenant can be added to a group with the Database Administration Console provided that the tenant does not have data currently allocated for the table where it is trying to become a member of a group. If tenant partitions have been allocated for the table, then the partitions must be deleted before the tenant can join a group.

Removing a tenant from a group causes the tenant to have its own partitions for that table. Tenant partitions are created based on the defaults that are set in the tenant record and allocation setting.

Multi-tenant objects

An *object* in a multi-tenant context is either a table, or an index, or a LOB.

A *object type* is the type of a specific object: table, index or LOB.

An *object instance* exists when space is allocated creating a data segment for an individual object.

Multi-tenant sequence

A multi-tenant sequence is a sequence that has a database-wide definition, but which has current values that are maintained on a per-tenant bases. That is, each tenant has its own current value for each multi-tenant sequence.

Object allocation/allocation state

Allocation is done for each multi-tenant database object for each tenant, except when set to not allocate. A storage object is created even if space for an object is not allocated. When space is allocated for an object it is known as a *partition*.

There are three types of multi-tenant database objects: tables, the indexes defined for tables, and the LOBs (CLOBs and BLOBs) defined for that table. As a group, the database objects defined for a table are known as a *table instance*.

Allocation states

The allocation state for an object is either allocated or not allocated. For a table instance either all objects that are part of the table instance are allocated, or none of the objects are allocated.

OpenEdge SQL supports two allocation states:

- **Allocated** (default) — Allocated implies immediate allocation using the default storage area.
- **Not allocated** — SQL syntax `NO SPACE`.

ABL supports three allocation states:

- **Immediate allocation** — Space is allocated at the time an object is created. This maps to the allocated state.
- **Delayed allocation** — Space is not allocated. Indicates that customizations to the storage area are required before allocation can occur, and that allocation will happen in a separate step. For example, delayed allocation is required if the tenant will be included in a group.
- **Do not allocate** — Space is not allocated. Indicates that allocation is not required or not planned for this object.

A database administrator can change the allocation state from not allocated to allocated at a future point, but once space is allocated it is not possible to change the allocation state (without deleting the object and the data for the object).

Security domains

Security domains are OpenEdge domains configured in an OpenEdge database. They identify how a user's identity must be authenticated and what tenant data they can access.

Each domain is configured with a domain name and other domain information, the name of an authentication system, and (in a multi-tenant database) the name of a tenant. The authentication system identifies the mechanism supported to authenticate users who are members of the domain, and whether OpenEdge or an ABL application performs the authentication. The tenant name identifies the tenant data that a user authenticated to a domain can access in the multi-tenant database. To be used for authentication, a configured domain must also be enabled (the default).

The membership of a user in a domain depends on the authentication system that is configured for the domain. If the configured authentication system can authenticate the user's identity, the user is a member of that domain. So, any domain to which a user belongs must be defined in every OpenEdge database that the user accesses, and must directly, or indirectly, identify an authentication system capable of authenticating the user's identity.

When users provides their user credentials for authentication to an application, either OpenEdge (including SQL) or the ABL application performs the authentication, depending on the specified domain and whether that domain is enabled. If authentication is successful, either OpenEdge or the ABL application assigns the user's identity to the database connection. For a multi-tenant database, setting the connection identity also sets the tenant identity specified by the database domain configuration, and the user can access only the specified tenant and shared data according to the user's access permissions.

Users can be defined inside or outside of the database, or both, depending on the authentication systems for which the database domains are configured. Users can also authenticate to domains defined in both non-multi-tenant and multi-tenant databases as long as the domains are enabled. If a user does not specify a domain in their user credentials, OpenEdge assumes the blank (`""`) domain, which by default is configured to authenticate users defined in the `_User` table accounts, and to provide access to data for the default tenant of a multi-tenant database. Note that security best practices recommend that all users in a production environment be a member of a named domain.

Table types

In a multi-tenant database there are two types of tables, *shared tables* and *multi-tenant tables*:

- **Shared tables** — define tables that are not multi-tenant. Every non-multi-tenant table is accessible to all users, regardless of tenancy. Shared tables are just like the tables in all OpenEdge databases prior to OpenEdge 11, and defined data access privileges on shared tables still apply.
- **Multi-tenant tables** — define tables where each tenant has its own version of data and regular tenant users cannot access the data of other tenants. Multi-tenant tables are accessible only to users in domains defined for a given tenant and by super tenants.

For a table to be multi-tenant, a database administrator has to mark a table to be multi-tenant. Data for multi-tenant tables is stored in partitions. Each tenant, including the default tenant, can optionally have a private data partition for each multi-tenant table.

Tenants

A *tenant* is a separate organizational entity within a multi-tenant database. Tenant data can only be accessed by users who are authorized to the tenant in a security domain, where a user acquires tenancy.

For tables that are defined to be multi-tenant, each tenant has its own instance of the multi-tenant table. Tenants have access only to the data in the instances designated for that specific tenant as well as to tables that are shared (non-multi-tenant).

Tenants are classified into the following types:

- Regular tenant
- Default tenant
- Super tenant

  Super tenants exist to manage multi-tenant databases; they are able to see all data.

There are many administrative tasks that are done by a super tenant. For example, if a table was migrated to be multi-tenant, the existing data is placed into the default tenant's partition. To move the data from the default tenant to another regular named tenant requires data access to more than one tenant. In other words you must sign-in as a super tenant to move data between these tenants.

ABL and SQL clients can establish a new effective tenancy for a super-tenant identity programmatically.

Tenant ID

A *Tenant ID* is a value set by the system when a tenant record is created. Tenant ID values are not guaranteed to always be the same value for the same named tenant in two different databases. For example, if you dump and load, tenants may have different IDs after being reloaded. However, a tenant ID will never change during a session.

Developers should only use tenant IDs in their code when they are certain that it refers to the correct tenant. In situations where code needs to find a tenant record or that is verifying that the correct tenant is available, it is safer to use the tenant name, since the tenant name will not change.

Tenant records are stored in the _Tenant table. Tenant IDs have the following characteristics:

- The tenant ID of the default tenant is always 0 (zero). There can be only one default tenant (an unnamed tenant).
- Tenant IDs of regular tenants (named tenants) are always a positive number.
- Tenant IDs of super tenants are always a negative number.

Tenancy

Tenancy is implicit and controls access to data. Users in a multi-tenant database always have tenancy. For regular tenant users and default tenant users, tenancy restricts the user's view of multi-tenant data to partitions for a particular tenant. Users connecting to a database without establishing tenancy are connected to the default tenant. This happens when a domain is not specified (uses the blank domain).

Tenant identity

Tenant identity is set based on a user id and domain. Once signed in the tenant identity does not change. Tenants cannot access any data for which the tenant identity is not authorized. Tenant identity determines tenancy for a user.