Progress OpenEdge Multi-tenant Database

Workshop

Gus Björklund, Instructor
gus@progress.com
This workshop is intended to introduce you to the inbuilt multi-tenant capabilities of the OpenEdge 11 RDBMS and show you how to make use of them in 4GL applications.

We have alternated between lecture and hands-on segments so you will have a chance to try for yourself the things that we will talk about.
Preliminaries

- Ask questions when you wish

- Focus of labs is on basic 4GL *programming* for data access. So sorry, no GUI stuff.

- Labs are not too long, except for the ones that are

- Take bio breaks as needed when you finish a lab
LAB Machines

- You each get your own virtual lab machine
- Hosted on Amazon EC2 and accessible via Windows Remote Desktop
- The OpenEdge 11.3.1 release
- The directory C:\mt has some files you will need for the lab portions
LABs

- Handouts have detailed instructions for each lab
- We have helpers who will assist you if you need help with something or you get stuck
- If you finish a lab section early, you can explore or try some other things while you wait for everyone else to finish.
Lab 0

Get connected to your Amazon EC2 AMI
Multi-tenant concepts
Who Cares about Multi-tenancy?

**SaaS vendors do.**

- Lower costs and operational excellence
  - Reduce machine resource requirements (cpu, memory, and disk)
  - Reduce operational costs
  - Reduce the number of instances
  - Cheaper and easier to manage
  - Requires fewer administration staff
  - Gain economies of scale

- Service efficiency is accomplished best by automation, which requires consistency
  - One good way to make that happen for application delivery is with multi-tenancy ...
Who Cares about Multi-tenancy?

SaaS vendors do.

Much to our surprise, we found that people who do not do SaaS are interested too.
What is a Tenant Anyway?

Tenants are:

- Named groups of people (users) that are related in some (organizational) way, share data, and use the same application(s)
- They might work in the same company, work in same division or dept. of a larger company, or belong to the same club
- Tenants don't know others may be using the same system
- For example, tenants could be the makers of these fine refreshing beverages:

![Sternburg](image1.png)  
![Krombacher](image2.png)  
![Oettinger](image3.png)
Multi-tenancy Options Continuum

**ISOLATED TENANCY**
- Tenant1
- Tenant2
- Tenant3
- App
- DB
- Infra.

**INFRASTRUCTURE TENANCY**
- Tenant1
- Tenant2
- Tenant3
- App
- DB
- Infrastructure

**APPLICATION TENANCY**
- Tenant1
- Tenant2
- Tenant3
- App
- DB
- Infrastructure

**SHARED TENANCY**
- Tenant1
- Tenant2
- Tenant3
- App
- DB
- Infrastructure

**Isolating**
- Easier customization, security
- Simpler throttling control
- Target dissimilar customers
- No transformation

**Sharing**
- Better economy of scale
- Simpler management
- Target like-customers
- Least cost to serve
Why Multi-tenancy? Vendors Want to...

- Increase infrastructure efficiency
  - Do the job will less hardware or more with same

- Reduce operational and administrative labor
  - Do the job with less work

- Decrease operating costs
  - Allow higher profits to provider
  - Allow lower prices to customers
SaaS Application Customers Want

- Low startup cost
- Fast deployment
- 100% uptime
- Responsive applications
- Data security (well, they *should* anyway)
- Low prices
Why *Database* Multi-tenancy?

- Lower SaaS application development cost and time
- Lower SaaS application deployment cost and time
- Lower operational costs
- Lower administrative costs
- Provide more flexibility for OpenEdge ISV partners
- Provide more flexibility for OpenEdge customers
In 10.2B, you can do this:
Extra “Tenant ID” Column for Multi-tenancy

<table>
<thead>
<tr>
<th>Tenant ID</th>
<th>Cust ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Lift Line Skiing</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>Urban Frisbee</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>Hoops Croquet</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>Fanatical Athletes</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>Game Set Match</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>Lift Line Skiing</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>High Tide Sailing</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>Pedal Power</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>Hoops Croquet</td>
</tr>
</tbody>
</table>

FOR EACH CUSTOMER WHERE (TenantID = A) and (regular stuff):
What's wrong with that?

Do we need more?
It Works, But There Are Just a Few Small Disadvantages

- Invasive: you have to change a lot of 4GL code
- Mistakes likely – then data given to wrong tenant
- Lock conflicts can occur among tenants
- Suboptimal performance
  - Low locality of reference
  - Low database buffer cache efficiency
  - Low I/O efficiency
And Still Other Disadvantages

- Per tenant bulk operations difficult
  - Backup, restore, reindex, delete, copy, move
- Tenant-level performance analysis difficult
- Tenant resource consumption metrics difficult
- Tenant resource utilization controls difficult
- and a bunch of other things
Yes! You do need more. And with OpenEdge 11, you get more.

The RDBMS has inbuilt multi-tenancy for both 4GL and SQL applications.
Main purpose of OpenEdge 11 inbuilt multi-tenancy is to:
Reduce costs for SaaS vendors

How does it work?
Multi-tenancy Options Continuum

Isolating

- Easier customization, security
- Simpler throttling control
- Target dissimilar customers
- No transformation

Sharing

- Better economy of scale
- Simpler management
- Target like-customers
- Least cost to serve
## Multi-tenancy Options Continuum

### Isolating
- Easier customization, security
- Simpler throttling control
- Target dissimilar customers
- No transformation

### Sharing
- Better economy of scale
- Simpler management
- Target like-customers
- Least cost to serve
### OpenEdge Multi-tenant Tables: NO Extra Column for Tenant ID

<table>
<thead>
<tr>
<th>Tenant ID</th>
<th>Cust ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Lift Line Skiing</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>Urban Frisbee</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>Hoops Croquet</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>Fanatical Athletes</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>Game Set Match</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>Lift Line Skiing</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>High Tide Sailing</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>Pedal Power</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>Hoops Croquet</td>
</tr>
</tbody>
</table>

**FOR EACH CUSTOMER WHERE (TenantID = A)**
OpenEdge Multi-tenant Tables: NO Extra Column for Tenant ID

<table>
<thead>
<tr>
<th>Cust ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lift Line Skiing</td>
</tr>
<tr>
<td>2</td>
<td>Urban Frisbee</td>
</tr>
<tr>
<td>3</td>
<td>Hoops Croquet</td>
</tr>
<tr>
<td>1</td>
<td>Fanatical Athletes</td>
</tr>
<tr>
<td>8</td>
<td>Game Set Match</td>
</tr>
<tr>
<td>9</td>
<td>Lift Line Skiing</td>
</tr>
<tr>
<td>2</td>
<td>High Tide Sailing</td>
</tr>
<tr>
<td>7</td>
<td>Pedal Power</td>
</tr>
<tr>
<td>9</td>
<td>Hoops Croquet</td>
</tr>
</tbody>
</table>

FOR EACH CUSTOMER:
OE 11 Multi-tenant Tables

Multi-tenancy Simplifies Development of Multi-tenant Applications

- Multi-tenancy built into the database
- Data physically partitioned by tenant identity
- Tenants share same schema definition
- **Minimal** application changes
  - Just set a per-database tenant name

*Fictitious example*
Multi-tenant Tables: Data Access

- Keys unique per tenant partition

### Multi-tenant Tables

<table>
<thead>
<tr>
<th>Tenant 1</th>
<th>Tenant 2</th>
<th>Tenant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Hertz Boston</em></td>
<td><em>Hertz London</em></td>
<td><em>Rent-a-wreck</em></td>
</tr>
</tbody>
</table>

#### Car Rental Application

<table>
<thead>
<tr>
<th>Customer</th>
<th>Tenant 1</th>
<th>Tenant 2</th>
<th>Tenant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lance Armstrong</td>
<td>2</td>
<td>John Cleese</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>John Cleese</td>
<td>Dennis Rodman</td>
</tr>
<tr>
<td>3</td>
<td>Tipper Gore</td>
<td>8</td>
<td>William Shatner</td>
</tr>
<tr>
<td>4</td>
<td>Shaquille O'Neal</td>
<td>7</td>
<td>Ben Stein</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>8</td>
<td>William Shatner</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>7</td>
<td>Ben Stein</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>9</td>
<td>Lindsay Lohan</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>10</td>
<td>Al Gore</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Fictitious example*
Multi-tenant Tables: Data Access

Multi-tenancy Simplifies Development of Multi-tenant Applications

- Keys unique per tenant partition
- Query is tenant-specific
  - Authenticate as tenant
    - _User
    - Client Principal
  - Assert tenant identity

*Fictitious example*
Multi-tenant Tables: Data Access

- Keys unique per tenant partition
- Query is tenant-specific
  - Authenticate as tenant
    - _User
    - Client Principal
  - Assert tenant identity

*Fictitious example*
Multi-tenant Tables: Data Access

**Multi-tenancy**

Simplifies Development of Multi-tenant Applications

- Keys unique per tenant partition
- Query is tenant-specific
- “Super-tenant” query
  - Authenticate & assert identity
  - No data of their “own”
  - Access to all tenant data by tenant ID or name

---

**Customer Schema**

Car Rental Application

<table>
<thead>
<tr>
<th>Super-tenant:</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR EACH customer</td>
</tr>
<tr>
<td>TENANT-WHERE</td>
</tr>
<tr>
<td>Tenant-id &gt; 0:</td>
</tr>
<tr>
<td>DISPLAY</td>
</tr>
<tr>
<td>cust-num, name.</td>
</tr>
</tbody>
</table>

---

*Fictitious example*
Multi-tenant Tables: Data Access

- Keys unique per tenant partition
- Query is tenant specific
- “Super-tenant” query
- Row-level tenant identification
- Virtual column available for display or selection (not in table definition)

Simplifies Development of Multi-tenant Applications

*Fictitious example*
3 Types of Tenants

- Default
- Regular
- Super
Lab 1

Creating a multi-tenant database
Tenant data storage
Multitenant Storage Area Structure: Tenant Data Partitions

Area Control Object

Customer Table

Order Table

CustID Index

Free page-cluster list
Multitenant Storage Area Structure: Tenant Data Partitions

Area Control Object

Free page-cluster list

Tenant A’s data

Customer Table

Order Table

CustID Index
Multitenant Storage Area Structure: Tenant Data Partitions

- **Area Control Object**
- **Free page-cluster list**
- **Tenant A's data**
- **Tenant B's data**
- **Tenant C's data**

**Customer Table**

**Order Table**

**CustID Index**
Tables: Physical Storage View (Type ii Data Areas)

Linked list of page-clusters

Shared Customer Table
OpenEdge Multi-tenant Tables: Automatic Table Partition for Each Tenant

Linked list of page-clusters for each tenant's data

Tenant A

Tenant B

Tenant C

Multitenant Customer Table
Numbers

500 tables
10 indexes per table (maybe a bit high)
100 tenants

= (500 * 100) + (500 * 10 * 100)

= 505,000 partitions !!!
Strategies for Storage Layout

With very many partitions, you have to keep it simple.
Strategies for Storage Layout

- Shared tables all in one area
- All tenants in one area
- 5 tenants per area
- "stripe" p partitions over n areas (p >> n)
- One storage area per tenant
- 3 areas per tenant (data, index, lob)
Tenants have their own data partitions

How does database know to which tenant a user belongs?
**DOMAINS**

- A tenant is a collection of users
- A user is a "person"
- A *security domain* is named set of rules ("policies") for how a group of users identity and tenant association is verified
- Every tenant must have *at least one* domain

<table>
<thead>
<tr>
<th>PK</th>
<th>_Tenant-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>TenantID</td>
</tr>
<tr>
<td></td>
<td>Tenant-ExtId</td>
</tr>
<tr>
<td></td>
<td>Tenant-Description</td>
</tr>
<tr>
<td></td>
<td>Tenant-Type</td>
</tr>
<tr>
<td></td>
<td>Tenant-Attributes[64]</td>
</tr>
<tr>
<td></td>
<td>Tenant-DataArea-Default</td>
</tr>
<tr>
<td></td>
<td>Tenant-IndexArea-Default</td>
</tr>
<tr>
<td></td>
<td>Tenant-LobArea-Default</td>
</tr>
<tr>
<td></td>
<td>Tenant-Sequence-Block</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>_Userid</th>
<th>Domain-Name*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User-Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Password</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TenantID*</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>Domain-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>Domain-Type</td>
</tr>
<tr>
<td>I2</td>
<td>Domain-Desc</td>
</tr>
<tr>
<td>I3</td>
<td>Domain-Enabled</td>
</tr>
<tr>
<td></td>
<td>Auditing-Context</td>
</tr>
<tr>
<td></td>
<td>Domain-Access-Code</td>
</tr>
<tr>
<td>I4</td>
<td>Tenant-Name*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>Domain-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>Domain-Type</td>
</tr>
<tr>
<td>I2</td>
<td>Domain-Desc</td>
</tr>
<tr>
<td>I3</td>
<td>_PAM-Module-Name</td>
</tr>
<tr>
<td></td>
<td>_PAM_Callback_Procedure</td>
</tr>
</tbody>
</table>
DOMAINS

- When you create a tenant, you must also create a domain.
- The domain specifies how user identity is validated.
- Possibilities include:
  - _user table has user name and password
  - operating system identity
  - external system like LDAP, Active Directory, etc.
  - Your 4GL code
How Users and Tenants Are Identified

- Users have names
- Tenants have domains
- Domains have names
- Together the two names are unique

user-name@domain-name
DOMAINS

When you log in you must specify user id and you must also specify a domain.

for example:
mpro –db foo –U user@domain –P password

we will see some other ways later.
Lab 2
Defining tenants, domains, users
Continuing with multi-tenant concepts
Multi-tenancy: Data Access, Sharing

**Tenant Groups**

- Some tenants can share the same data/partition
- Employee access to shared customer list

*Fictitious example*

<table>
<thead>
<tr>
<th>Car Rental Application</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td></td>
</tr>
<tr>
<td>Tenant 1 (Hertz Boston)</td>
<td></td>
</tr>
<tr>
<td>1 Lance Armstrong</td>
<td></td>
</tr>
<tr>
<td>2 John Cleese</td>
<td></td>
</tr>
<tr>
<td>3 Tipper Gore</td>
<td></td>
</tr>
<tr>
<td>4 Shaquille O’Neal</td>
<td></td>
</tr>
<tr>
<td>7 Ben Stein</td>
<td></td>
</tr>
<tr>
<td>8 William Shatner</td>
<td></td>
</tr>
<tr>
<td>Tenant 3 (Rent-a-wreck)</td>
<td></td>
</tr>
<tr>
<td>2 Dennis Rodman</td>
<td></td>
</tr>
<tr>
<td>7 Nick Nolte</td>
<td></td>
</tr>
<tr>
<td>9 Lindsay Lohan</td>
<td></td>
</tr>
<tr>
<td>10 Al Gore</td>
<td></td>
</tr>
</tbody>
</table>
Multi-tenancy: Data Access, Sharing

Tenant Groups

- Some tenants can share the same data/partition
  - Employee access to shared customer list

*Fictitious example*
Multi-tenancy: Data Access, Sharing

Tenant Groups

- Some tenants can share the same data/partition
  - Employee access to shared customer list
- Data exists for the life of the group
  - e.g. Regional data
- Row identity associated with group
  - BUFFER-GROUP-ID()
  - BUFFER-GROUP-NAMES()
- Group membership is per table

*Fictitious example*
Multi-tenancy: Data Model

The Data Model

- Multi-tenant objects
  - Tables and associated indexes & LOBs
  - Sequences
- Shared objects still available
  - Same as today
- Shared only, not multi-tenant
  - Triggers & stored procedures
  - Initial values
- Limits
  - Support for up to 32,767 tenants

*Fictitious example

Car Rental Application

<table>
<thead>
<tr>
<th>Tenant 1</th>
<th>Tenant 2</th>
<th>Tenant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Hertz Boston)</td>
<td>(Hertz London)</td>
<td>(Rent-a-wreck)</td>
</tr>
<tr>
<td><strong>Customer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Lance Armstrong</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>John Cleese</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Tipper Gore</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Shaquille O’Neal</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Ben Stein</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>William Shatner</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Dennis Rodman</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Nick Nolte</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Lindsay Lohan</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Al Gore</td>
<td></td>
</tr>
</tbody>
</table>
Multi-tenancy: Tenant Provisioning

Managing Tenants

- Tenant creation: ABL, APIs, DDL & GUI
  - Programmatic tenant provisioning
  - Tenant partition creation optional
  - Tenant level activation/deactivation
- Identification (via “_Tenant” table)
  - Database specific tenant ID
  - User friendly name: “Hertz, Boston”
  - App specific ID (could be UUID)
- Resource access
  - Runtime security by user by tenant
  - Governors: Limit resource usage

*Fictitious example
Multi-tenant Tables: Operational Features

Operational Features

- **Tenant partition maintenance**
  - Tenant-specific object move
  - Add/drop tenants/objects
  - Data dump/load
  - .df support
  - Index maintenance tools

- **Monitoring**
  - Promon, VSTs
  - Analysis tools
  - .lg file (other log files)
Regular Tenant 4GL Queries
Note: 4GL Permissions

- 4GL user permissions for tables and columns work the same as before
  - CAN* permissions still apply: CAN-READ, CAN-WRITE, CAN-CREATE, CAN-DELETE, CAN-LOAD, CAN-DUMP
  - Only one set of permissions exists for tables, including multi-tenant tables

- All database users are subject to permission settings
  - Super-tenants users
  - Regular tenant users
  - Default tenant users
  - Administrators can change permissions, super-tenants by default cannot

- No need to say more.
4GL Queries

- Work the same as before
- For regular tenants, your code should work without change
- Effective tenant id determines what data is returned.
- What you see depends on who you are
- Same query returns different data for different tenants

for each customer:
  display custnum name.
end.
Lab 3

Looking at tenant data
Now you must go to the principal's office
What data will you see? Depends who you are. Database uses your identity to decide.

CLIENT-PRINCIPAL is basis for identity.
## Multi-tenant Identity

### The _User table (ABL & SQL) and friends

<table>
<thead>
<tr>
<th>_Tenant*</th>
<th>PK</th>
<th>_Tenant-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>TenantID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tenant-ExtId</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tenant-Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tenant-Type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tenant-Attributes[64]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tenant-IndexArea-Default</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tenant-LobArea-Default</td>
<td></td>
</tr>
</tbody>
</table>
| | Tenant-Sequen
c

<table>
<thead>
<tr>
<th>_User</th>
<th>PK</th>
<th>Userid</th>
<th>Domain-Name*</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td>User-Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Password</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>_TenantID*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_sec-Authentication-Domain</th>
<th>PK</th>
<th>Domain-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>Domain-Type</td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>Domain-Desc</td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>Domain-Enabled</td>
<td></td>
</tr>
<tr>
<td>I4</td>
<td>Auditing-Context</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain-Access-Code</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_Tenant-Name*</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_sec-Authentication-System</th>
<th>PK</th>
<th>Domain-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>Domain-Type</td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>Domain-Desc</td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>PAM-Module-Name</td>
<td></td>
</tr>
</tbody>
</table>
| | PAM_Call

 supplementary comments on the relationships:

1. U1 is the primary key for the _Tenant table.
2. Userid is the primary key for the _User table.
3. Domain-Name is a foreign key in the _sec-Authentication-Domain table referencing the _Tenant table.
Creating CLIENT-PRINCIPAL tokens

Easy, peasy

```plaintext
DEFINE VAR hCP1 AS HANDLE.
CREATE Client-Principal hCP1.
    hCP1:Initialize(“Alice@avis”).
    hCP1:SEAL("password1").
```
Creating CLIENT-PRINCIPAL tokens 2

Easy, peasy

```c
DEFINE VAR hCP2 AS HANDLE.
CREATE Client-Principal hCP2.
hCP2::Initialize("Bob@hertz").
hCP2::SEAL("password2")
```

And there are lots of properties you could set also
### Client Principal Object Properties

<table>
<thead>
<tr>
<th>SESSION-ID</th>
<th>LOGIN-EXPIRATION-TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER-ID</td>
<td>LOGIN-HOST</td>
</tr>
<tr>
<td>DOMAIN-NAME</td>
<td>LOGIN-STATE</td>
</tr>
<tr>
<td>AUDIT-EVENT-CONTEXT</td>
<td>QUALIFIED-USER-ID</td>
</tr>
<tr>
<td>CLIENT-TTY</td>
<td>ROLES</td>
</tr>
<tr>
<td>CLIENT-WORKSTATION</td>
<td>SEAL-TIMESTAMP</td>
</tr>
<tr>
<td>DB-LIST</td>
<td>STATE-DETAIL</td>
</tr>
<tr>
<td>DOMAIN-DESCRIPTION</td>
<td>TYPE</td>
</tr>
<tr>
<td>DOMAIN-TYPE</td>
<td>LIST-PROPERTY-NAMES()</td>
</tr>
<tr>
<td>INSTANTIATING-PROCEDURE</td>
<td>TENANT-ID()</td>
</tr>
<tr>
<td></td>
<td>TENANT-NAME()</td>
</tr>
</tbody>
</table>
Switching Identity with CLIENT-PRINCIPALs

SET-DB-CLIENT(hCP1).
/* now we are Alice */
FIND Customer WHERE name = "Alices Customer".
SECURITY-POLICY:SET-CLIENT (hCP2).
/* Now we are Bob */
CREATE Customer.
name = "Bobs Customer".
Other Ways to Establish Identity

With a userId@domainName, do:

```bash
SETUSERID("alice@hertz", "revolution").
```

or:

```bash
CONNECT –U alice@hertz –P revolution.
```

A CLIENT-PRINCIPAL token will be created for you automatically, under the covers.
Lab 4
Looking at tenant data
Using the Super-tenant
Why Do We Need Super-tenants?

- Sometimes you need to operate on data that belongs to other tenants
- Super-tenants exist to allow housekeeping cross-tenant tasks such as
  - Saas administration i.e. billing, moving tenants..
  - Migration from previous database versions
  - Handling of aggregate information across tenants
- Super-tenants have no data of their own
- Super-tenants have special ABL to allow them to:
  - Get access to regular tenant data
  - Execute legacy code
Super-tenant

- Special tenant, unlike any other
- Can read and write all tenants data
- Has users, like other tenants
  - alice@super, bob@super
- You will have to write NEW code for super tenant
- New 4GL functions for super tenant programming
Some New and a Few Modified 4GL Functions

- IS-DB-MULTI-TENANT( ) function
- IS-MULTI-TENANT Property
- SET-EFFECTIVE-TENANT( ) function
- GET-EFFECTIVE-TENANT-ID( ) function
- GET-EFFECTIVE-TENANT-NAME( ) function
- TENANT-WHERE clause
- TENANT-NAME-TO-ID( ) function
- CREATE statement FOR TENANT qualifier
- TENANT-ID( ) function
- TENANT-NAME( ) function
- BUFFER-CREATE Method
- BUFFER-TENANT-ID( ) function
- BUFFER-TENANT-NAME( ) function
- BUFFER-TENANT-ID attribute
- BUFFER-TENANT-NAME attribute
- REPOSITION query TO ROWID statement
- REPOSITION-TO-ROWID method

- Check if multi-tenant
- Set/get effective tenant
- filter query by tenant
- convert name to number
- Identify tenant(s)
- Qualify ROWID with tenant
SET-EFFECTIVE-TENANT () function

- Supertenant can become another tenant
- Can then read and write their data as if you were they

```
SET-EFFECTIVE-TENANT ("Avis").
for each customer:
    display custnum
    name.
end.
```
TENANT-WHERE query clause

- Super tenant can get all tenants data or some
- Add TENANT-WHERE clause to query

```plaintext
for each customer
    TENANT-WHERE tenant-id () > 0
    and tenant-name() < "M":
        display custnum
        name.
end.
```
BUFFER-TENANT-NAME () function

- Tells you which tenant owns buffer contents

for each customer
  TENANT-WHERE tenant-id () > 0
  and tenant-name() < "M":

  display BUFFER-TENANT-NAME (customer)
    custnum
    name.

end.
Lab 5

Let's play super-tenant
Migration of Existing Data
How can we get our existing data organized (moved) into the right tenants partitions?
Default Tenant

- Special tenant, unlike any other
- NOT intended for general use
- Has tenant id zero and default partition(s)
- Purpose: enable conversion of existing data
- Owns data when you conv1011 and mark tables with data as multi-tenant
- We assume
  - you will move the data
  - code to move data will be super tenant code
- Once data are moved, default tenant has nothing
Default Tenant

- I lied. But only a little.
- The default tenant can access regular shared tables.
- All users belong to default tenant when database is not multi-tenant enabled.
  - 10.2 and earlier databases are not multi-tenant.
Multi-tenant Tables: Data Migration with DIY Tenant ID Column

- Enable multi-tenancy on existing db
- Mark existing table as multi-tenant table
- Data in default tenant partition
- Set super-tenant identity
- Move data
- Truncate empty partition

<table>
<thead>
<tr>
<th>Schema</th>
<th>Default Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>Lance Armstrong</td>
</tr>
<tr>
<td>1 2</td>
<td>John Cleese</td>
</tr>
<tr>
<td>1 3</td>
<td>Tipper Gore</td>
</tr>
<tr>
<td>2 4</td>
<td>Shaquille O’Neal</td>
</tr>
<tr>
<td>2 7</td>
<td>Ben Stein</td>
</tr>
<tr>
<td>2 8</td>
<td>William Shatner</td>
</tr>
<tr>
<td>3 2</td>
<td>Dennis Rodman</td>
</tr>
<tr>
<td>3 7</td>
<td>Nick Nolte</td>
</tr>
<tr>
<td>3 9</td>
<td>Lindsay Lohan</td>
</tr>
<tr>
<td>3 10</td>
<td>Al Gore</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shared Tenancy</th>
<th>OE11 Shared Tenancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tenant1 Tenant2 Tenant3</td>
<td>Tenant1 Tenant2 Tenant3</td>
</tr>
<tr>
<td>App</td>
<td>App</td>
</tr>
<tr>
<td>DB</td>
<td>DB</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Infrastructure</td>
</tr>
</tbody>
</table>
Moving the Data with DIY Tenant ID Column

```
DEFINE BUFFER bCust FOR cust.
FOR EACH Cust WHERE Cust.tenant-id = 1
    TENANT-WHERE BUFFER-TENANT-ID(Cust)=0:
    CREATE bCust USE-TENANT 1.
    BUFFER-COPY Cust TO bCust.
    DELETE Cust.
END.
```
Moving the Data with DIY Tenant ID Column

```plaintext
DEFINE BUFFER bCust FOR customer.
FOR EACH customer:
    FIND myTenant WHERE
        myTenant.tenantId = customer.tenantId.
    SET-EFFECTIVE-TENANT (myTenant.Name).
    CREATE bCust.
    BUFFER-COPY customer TO bCust.
    DELETE customer.
END.
```
SETUSERID “alice@hertz1”. FOR EACH customer: DISPLAY customer.

NOTE: Existing tenant column remains but is no longer needed for new multi-tenant queries.
Multi-tenant Tables: Data Migration with Database per Tenant

<table>
<thead>
<tr>
<th>Tenant1</th>
<th>Tenant2</th>
<th>Tenant3</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>App</td>
<td>App</td>
</tr>
<tr>
<td>DB</td>
<td>DB</td>
<td>DB</td>
</tr>
</tbody>
</table>

**Infrastructure**

**DB #1** *(Hertz Boston)*

- 1 Lance Armstrong
- 2 John Cleese
- 3 Tipper Gore

**Customer**

**DB #2** *(Hertz London)*

- 4 Shaquille O’Neal
- 7 Ben Stein
- 8 William Shatner

**Customer**

**DB #3** *(R.W.)*

- 2 Dennis Rodman
- 7 Nick Nolte
- 9 Lindsay Lohan
- 10 Al Gore

**Customer**

**OE11 Shared Tenancy**
Multi-tenant Tables: Data Migration with Database per Tenant

<table>
<thead>
<tr>
<th>DB #1 (Hertz Boston)</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 Lance Armstrong</td>
</tr>
<tr>
<td></td>
<td>2 John Cleese</td>
</tr>
<tr>
<td></td>
<td>3 Tipper Gore</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DB #2 (Hertz London)</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 Dennis Rodman</td>
</tr>
<tr>
<td></td>
<td>7 Nick Nolte</td>
</tr>
<tr>
<td></td>
<td>9 Lindsay Lohan</td>
</tr>
<tr>
<td></td>
<td>10 Al Gore</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DB #3 (R.W.)</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 Lance Armstrong</td>
</tr>
<tr>
<td></td>
<td>7 Ben Stein</td>
</tr>
<tr>
<td></td>
<td>9 Lindsay Lohan</td>
</tr>
<tr>
<td></td>
<td>10 Al Gore</td>
</tr>
</tbody>
</table>

- **Create new multi-tenant db**
  - Can convert an existing one
  - Add tenants
  - Load multi-tenant schema

- **Dump from current**
  - `proutil DB1 –C dump customer`

- **Load to new**
  - `proutil MTdb –C load customer tenant hertz2`
### Multi-tenant Tables: Data Migration with Database per Tenant

#### Infrastructure or Application Tenancy

<table>
<thead>
<tr>
<th>Tenant 1</th>
<th>Tenant 2</th>
<th>Tenant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>App</td>
<td>App</td>
</tr>
<tr>
<td>DB</td>
<td>DB</td>
<td>DB</td>
</tr>
</tbody>
</table>

#### Schema

- **Tenant 1 (Hertz)**
  - 1 Lance Armstrong
  - 2 John Cleese
  - 3 Tipper Gore
- **Tenant 2 (Hertz)**
  - 4 Shaquille O'Neal
  - 7 Ben Stein
  - 8 William Shatner
- **Tenant 3 (Rent-a-wreck)**
  - 2 Dennis Rodman
  - 7 Nick Nolte
  - 9 Lindsay Lohan
  - 10 Al Gore

#### OE11 Shared Tenancy

<table>
<thead>
<tr>
<th>Tenant 1</th>
<th>Tenant 2</th>
<th>Tenant 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>DB</td>
<td>INFRASTRUCTURE</td>
</tr>
</tbody>
</table>

---

**Customer**

- 1 Lance Armstrong
- 2 John Cleese
- 3 Tipper Gore
- 4 Shaquille O'Neal
- 7 Ben Stein
- 8 William Shatner
- 2 Dennis Rodman
- 7 Nick Nolte
- 9 Lindsay Lohan
- 10 Al Gore
Benefits of “the best thing since sliced bread”

Simplifies development
- Minimal application changes
- No tenant-based customizations for queries or other data access

Eases deployment
- Tenant access to data is transparent, based on identity
- Tenants can be quickly and efficiently added, removed, and managed

Decreases maintenance overhead
- Fewer databases to manage, better resource utilization
- Tenant-based utilities and tools make maintenance tasks easier

Maintains security of tenant data
- Physical separation within database
- Tenant authentication required for data access
All Questions answered