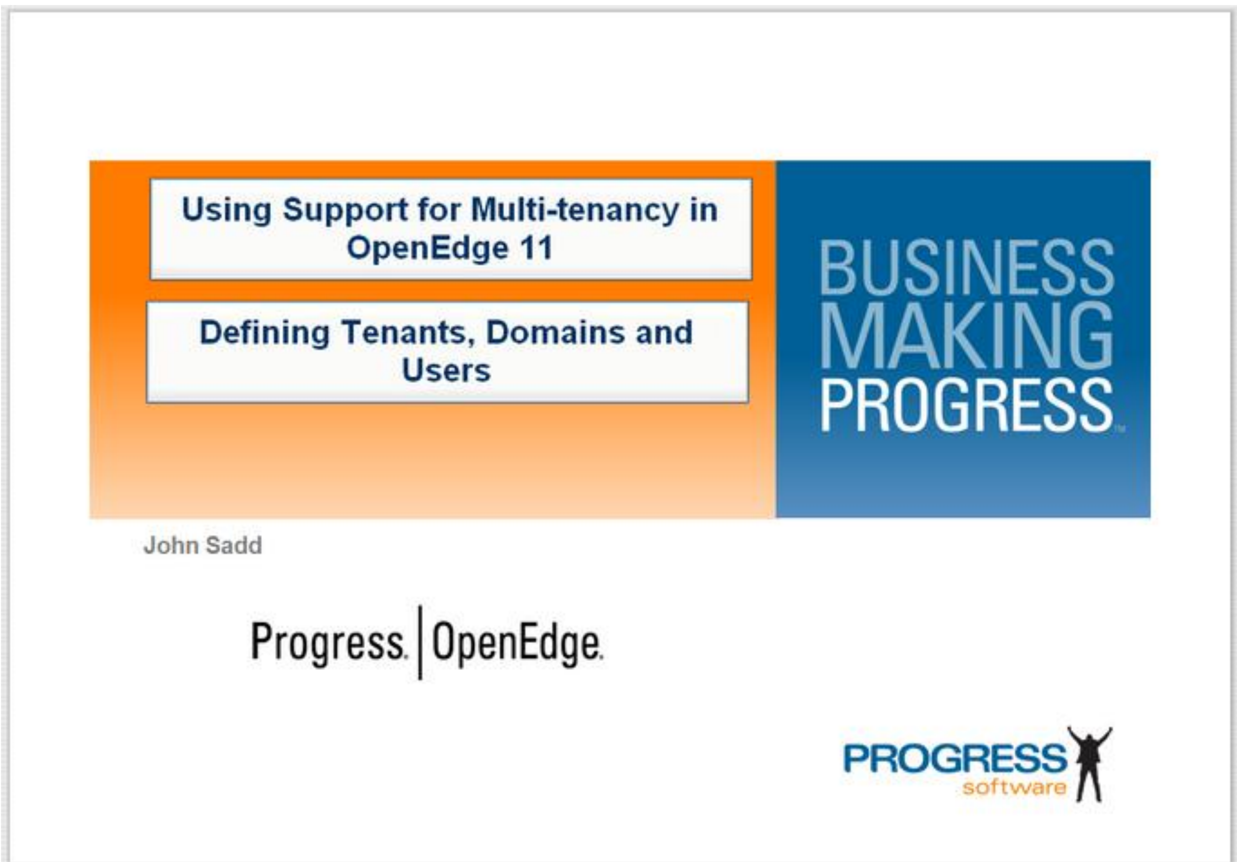
DEFINING TENANTS, DOMAINS AND USERS

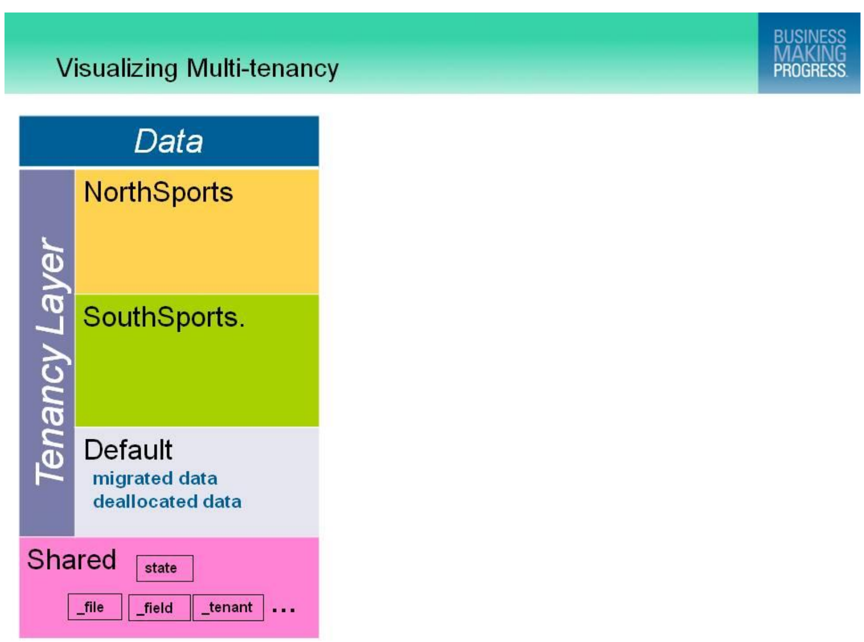
John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
November 2011



DISCLAIMER

The first video and paper in this series on support for multi-tenancy in OpenEdge 11 introduced you to some of the basic concepts and to the support for managing multi-tenant tables in the OpenEdge tools. This paper accompanies the next three videos in the series, on defining tenants, enabling tables for multi-tenancy, and defining security domains and users. In this paper I go into the definition of tenants in the database in somewhat more detail, and show you how to define tenants in the Database Administration Console. I've enabled the **sportsmt** database for multi-tenancy, and now it's time to define some actual tenants, one for each of the databases that are being combined here into one.

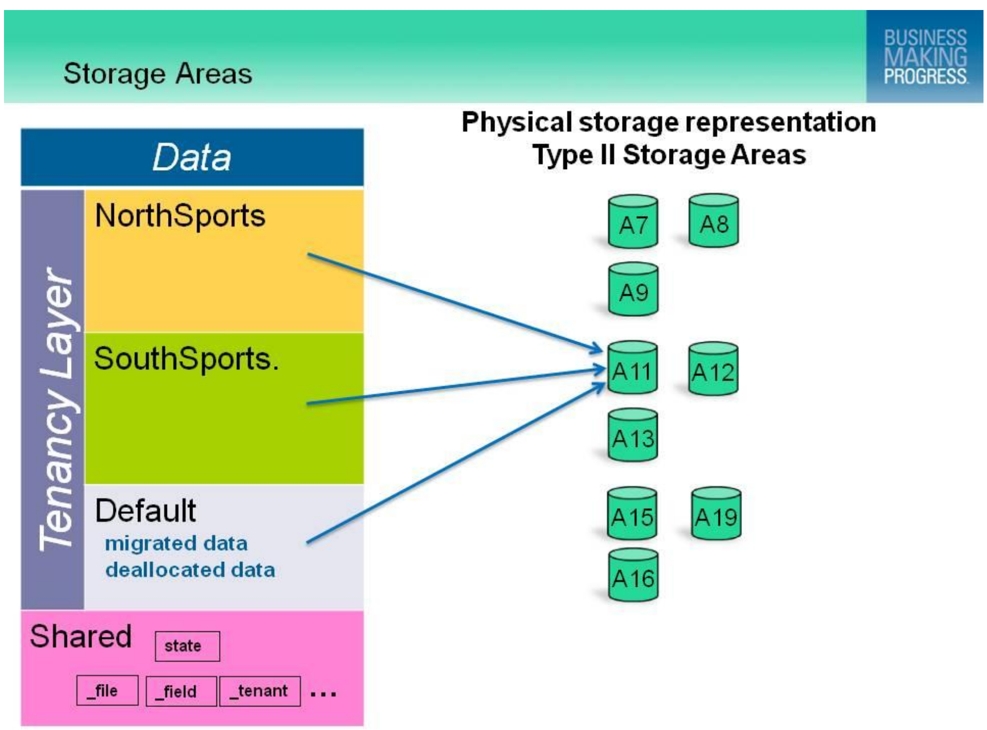
First let's look at some diagrams that illustrate at a high level how this is all organized. When you enable multi-tenancy in a database, and then define specific tenants, the OpenEdge DBMS separates their data in what we refer to as the tenancy layer of the database.



The data for different tenants is kept absolutely separate, so that there is no danger of applications and their users retrieving data that they should not see. The diagram above shows two of the four tenants I'm going to create, NorthSports and SouthSports. There's also something called a **Default Tenant**. In a sense the default tenant is no tenant at all. If you convert a database from OpenEdge 10 to OpenEdge 11, which is something I'll do later in this series, then initially all the migrated data is placed in the Default Tenant area until you move it to where it belongs. It can also be a place to store data that's no longer part of a particular tenant, typically when a tenant is removed from the database, until you determine what to do with it.

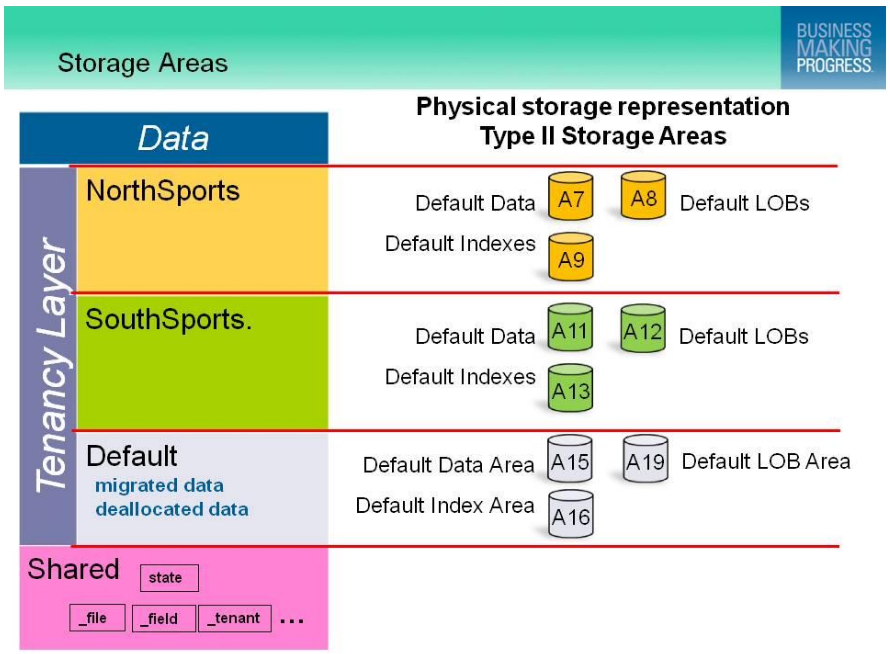
We also refer to **shared data** in a multi-tenant database. This is effectively what all data is in databases prior to OpenEdge 11. We continue to have space for shared data for two reasons, as illustrated above. One is that some tables may contain data that is common to all tenants, for example the State table in the sportsmt database. There's no need for separation of data by tenant for the list of states. The second use is that the meta-schema tables are shared by all tenants, so they always go into the shared area.

I'm not going to go into any detail in terms of the internals of the OpenEdge database, but you should know by now that multi-tenant data goes into Type II storage areas. There's no restriction on how you allocate data within the Type II areas you define. You can put data for multiple tenants into a single storage area, as shown below, and the database manager will make sure it's all kept separate:

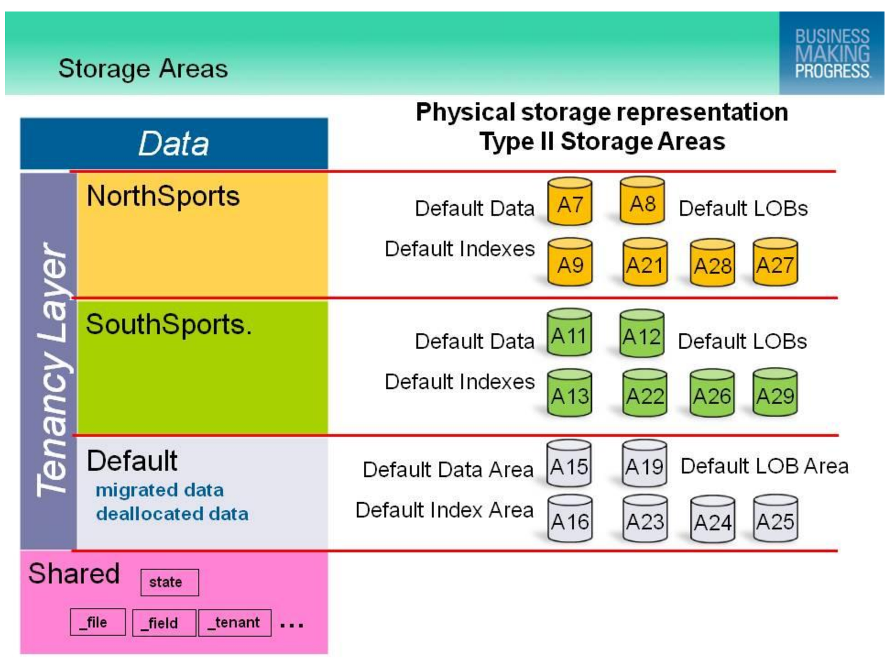


However, for reasons of database maintenance and storage management, you may find it advantageous to allocate data for different tenants in different sets of Type II areas, as shown below.

With that in mind, you need to understand that you can allocate three different kinds of information independently. The first is the actual data in the tables, except for large objects. The second is a table's indexes, and the third is a table's large object fields, its CLOBs and BLOBs. You can put these all into the same areas or allocate them independently. The following diagram also illustrates that when you define tenants, you define a default storage area for each of these three types of data. However, you can change and specialize that allocation as you go along:

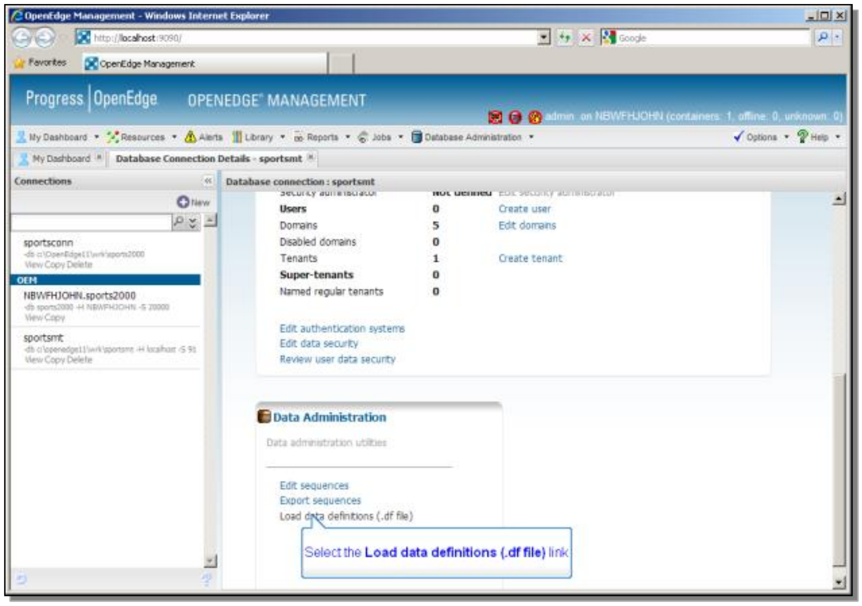


You can have additional storage areas that you allocate data, indexes or LOBs to, in addition to the default areas, if you wish:

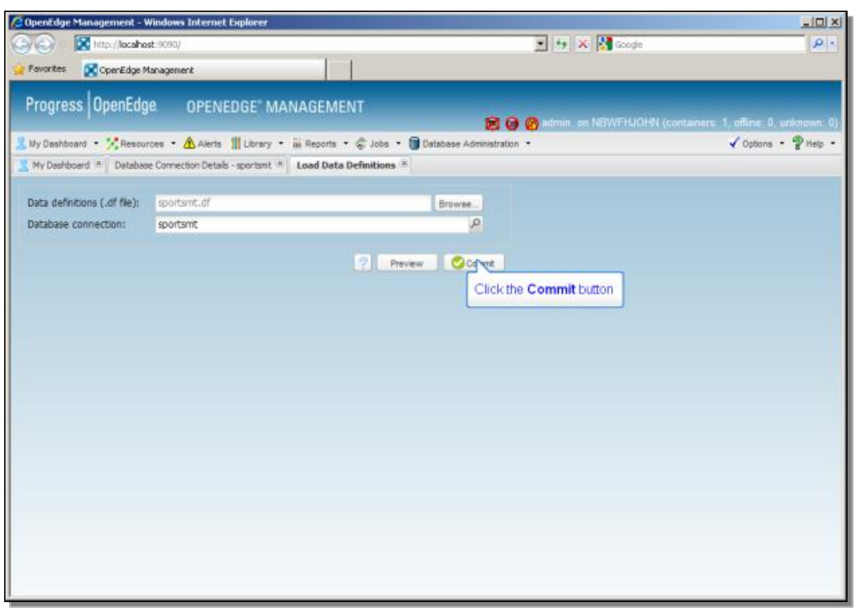


For shared tables and for the schema tables, it’s possible to use Type I storage areas, but for a new database, type II storage areas are always recommended.

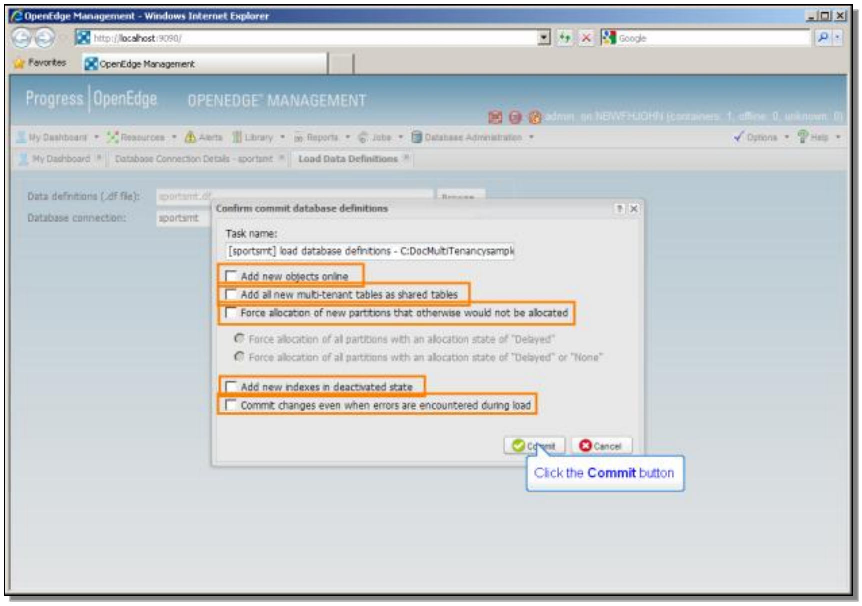
That’s enough to get you started. I created sportsmt as an empty OpenEdge 11 database, and enabled multi-tenancy, but that’s all that’s been done so far. The next job is to load the schema definitions into the database, just as you would do in earlier releases. You can do this directly from the Admin Console:



I locate the .df file that was dumped from one of the identical OpenEdge 10 databases that I’m going to combine. I could **Preview** the contents of that file in a visual form and possibly make some adjustments, but for now I just press the **Commit** button to load the file:



Here you can see a number of options in the Commit dialog:



Add new objects online lets you define new tables, fields, indexes, or sequences beyond what's in the .df you're loading.

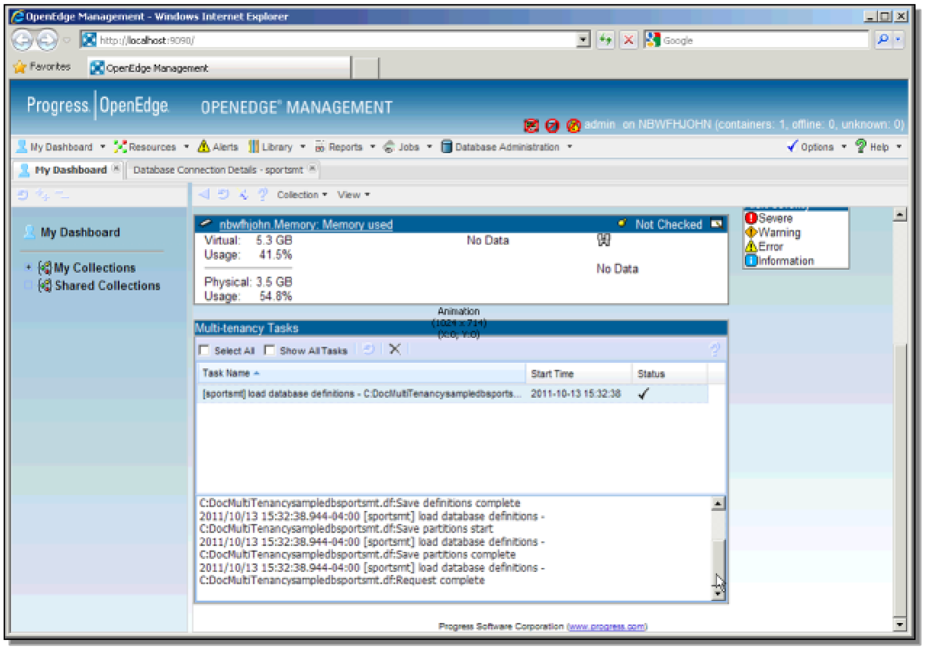
Add multi-tenant tables as shared tables doesn't apply to the current example, because I'm presuming that the .df came from an OpenEdge 10 database. But if you were loading a .df from an OpenEdge 11 database, it might have statements in it that define multi-tenant tables in that database, and this option tells the loader to disregard those definitions and make all the tables start out as shared tables.

Force allocation is a little advanced to discuss right now; later you'll see that when you define a tenant you can specify that you want the actual allocation of its partitions to be delayed, or even suspended until the first actual data for the tenant is defined. This option would override that.

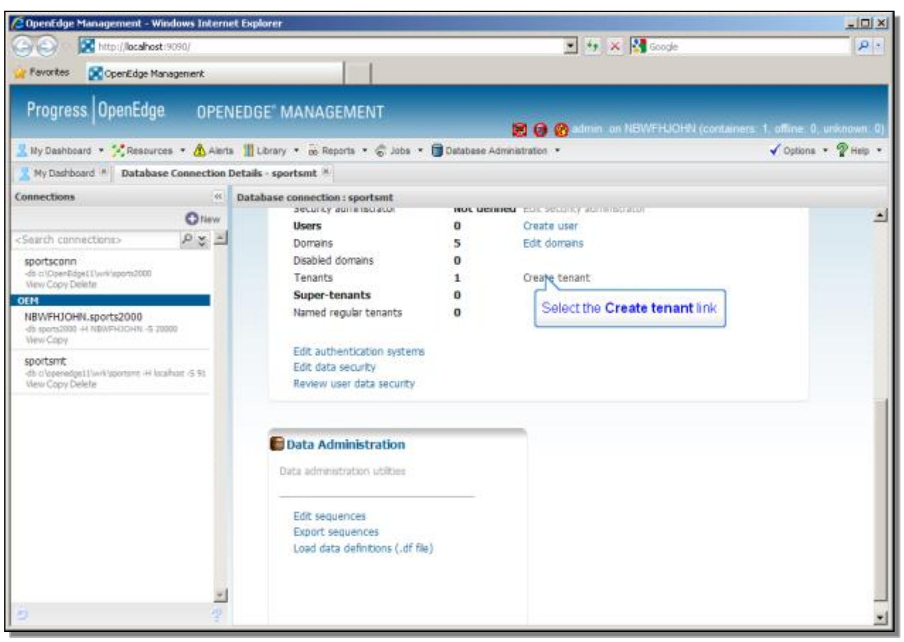
The next option tells the loader to **add any new indexes in the .df file as initially deactivated**.

The final option to **Commit changes even when errors are detected** means what it says; note that if errors *are* encountered you might wind up with your database in a corrupt state. I just take the defaults and click **Commit**.

Remember that even if you're just running Explorer, you do get a **Dashboard** with a multi-tenancy tasks pane, or viewlet, since many operations you select are run as batch jobs. The dashboard comes up automatically, and if I click the **Refresh** button, I see my **load database definitions** task. If I select it, I can see that it's executing, and in fact has already run to completion:



Now I reselect the **Database Connection Details** tab to get back to where I was, and I'm ready to start creating tenants for the new database. There's a link to do that right on the main display in the Details Frame:



The screenshot below shows the **New Tenant** form with all the essential information filled in.

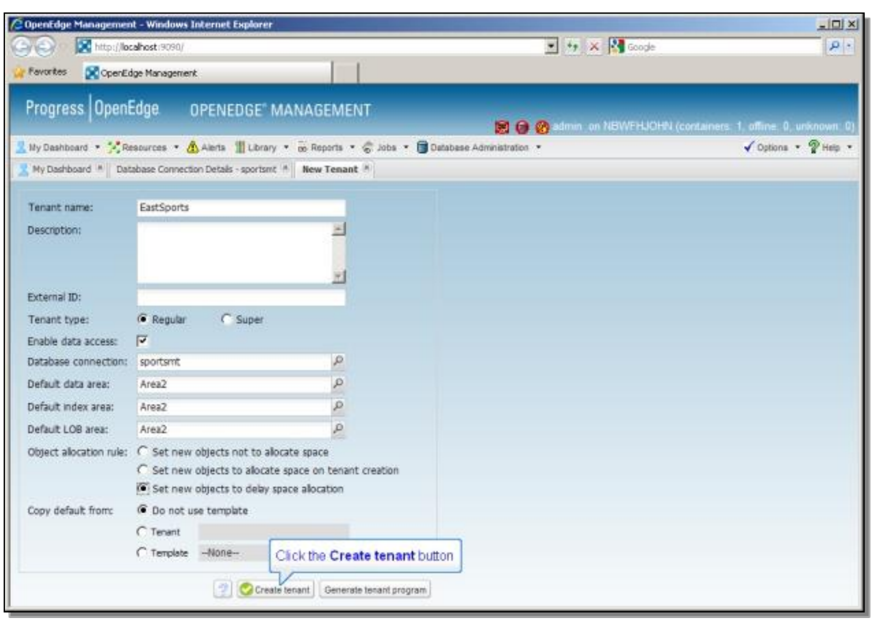
I call the first tenant **EastSports**. Remember that my goal is to combine the data in four existing databases that were set up in OpenEdge 10 for four different customers, who become my new tenants. I could add a free-form description if I wanted, and also what's called an **External ID**, which can help later to follow the progress of transactions as they are committed. EastSports is a **Regular** tenant. I introduce super tenants later in this series.

Unchecking the **Enable data access** checkbox would allow me to disable data access for the tenant, so that users defined for that tenant would not be able to access multi-tenant data. This is something you can also set on and off later on as part of security management.

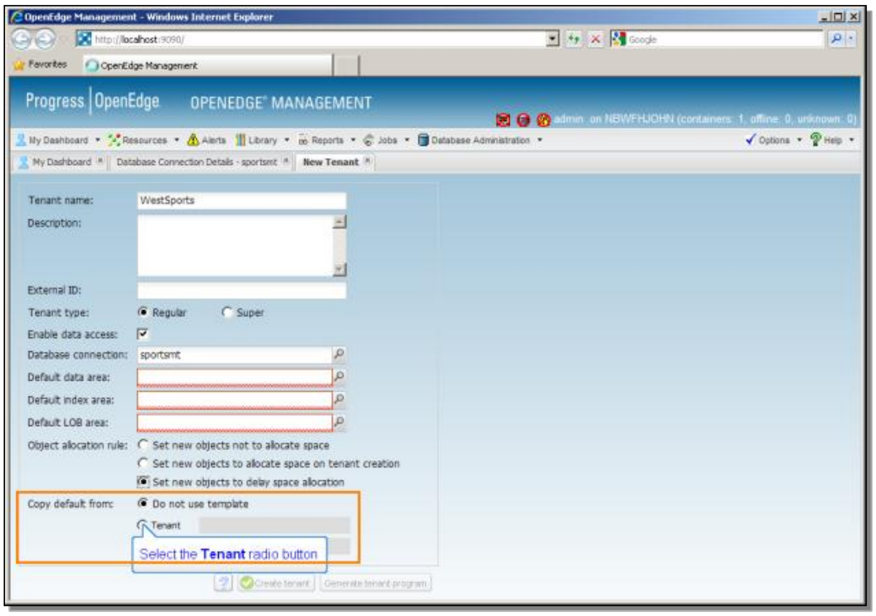
The **Database connection** I'm defining the tenant for is of course **sportsmt**.

Next you can see what the diagrams in the introduction to this video illustrated: I can define a default storage area for data, for indexes, and for LOBs. The easiest way to do that is to select the magnifier icon. This brings up a list of all the database's Type II storage areas. I just have two to choose from, and for this tenant, I choose **Area2**. I do the same for the indexes, though remember that you can define independent areas for indexes for a tenant, as well as for the default LOB area.

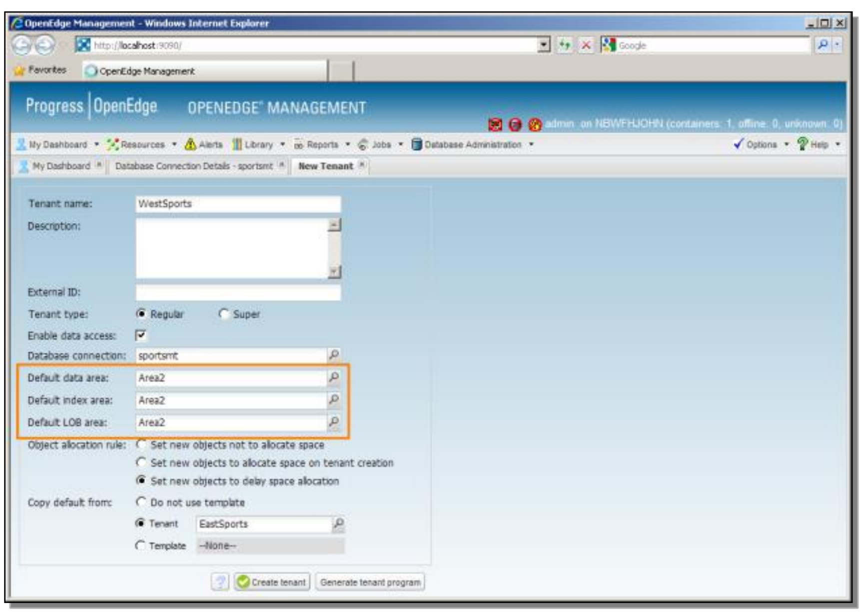
Now that I've defined the default allocation for all the data partitions for this tenant, I have to define when I want that space allocated. The default is **on tenant creation**, which we also refer to as *immediate*. As soon as the tenant is created an initial cluster of space is allocated in its default areas for all the database's multi-tenant tables, including their indexes and large objects. You may not want this default, for one of two broad reasons. You may not want to pre-allocate space at all. In this way you avoid allocating space that may never be used, if not all of the database's tenants wind up having records in all the multi-tenant tables. Your customers who are represented as tenants may not purchase all the modules that use all the database tables, for instance. The other case is that you may want to delay allocation so that you can assign individual table data, or their indexes and LOBs to different areas on a more fine-tuned basis than just using the defaults, either using the Admin Console, or as part of individual database transactions once your application is in production. For this tenant I just choose **delay space allocation**, and click **Create tenant** to finish creating this first tenant:



Now I need to move on and create another tenant. I call this one **WestSports**. WestSports has its own Customers and Orders and Orderlines, just as EastSports does. I select **delay space allocation** here as well. When I selected the delay allocation option, the final **Copy default from** choices shown below are enabled. If allocation is not immediate, I can copy the rest of the tenant definition from another tenant, or from a template I could create. I select **Tenant**:



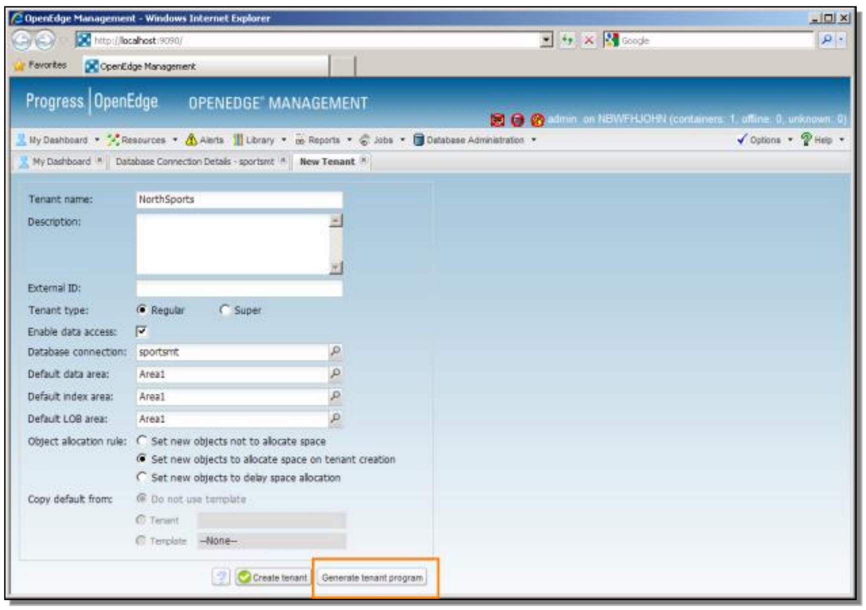
This brings up a list of existing tenants, and of course there's only one – EastSports – so I select that, and WestSports is assigned the same default areas as EastSports:



Remember that you can assign different tenants to the same storage areas, and the database still keeps their data securely separated.

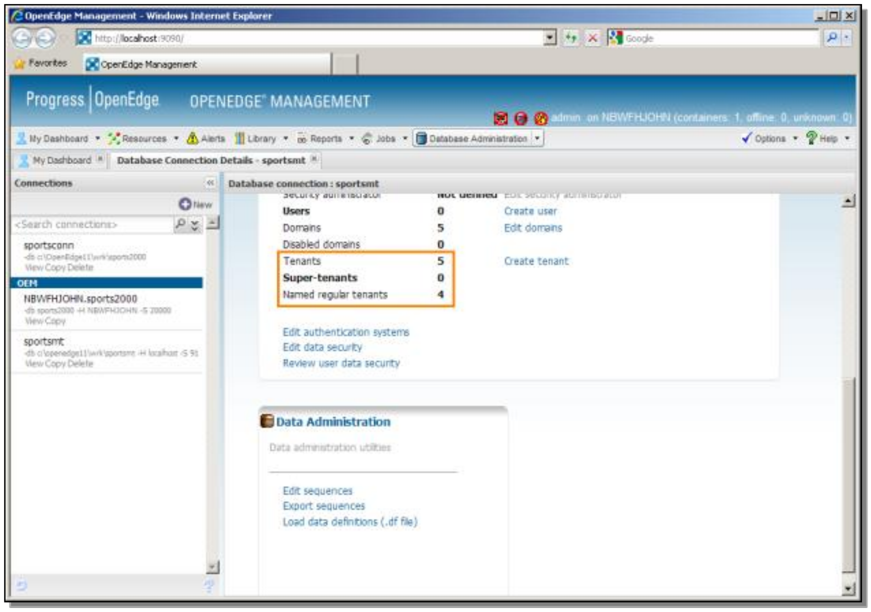
I create one more tenant, called **SouthSports**, and likewise, copy the area assignments from EastSports. My final tenant I name **NorthSports**, and I assign its default allocation to **Area1**. I do the same for its indexes and LOBs. I leave its allocation rule set to **on tenant creation**.

In passing, I'll just briefly mention the **Generate tenant program** button shown below. In addition to using a tenant as a template for another tenant, you can click this button to generate an ABL procedure containing the code to create the tenant you've defined. You could then parameterize the procedure to allow you to create additional similar tenants programmatically, perhaps even during execution of your application.

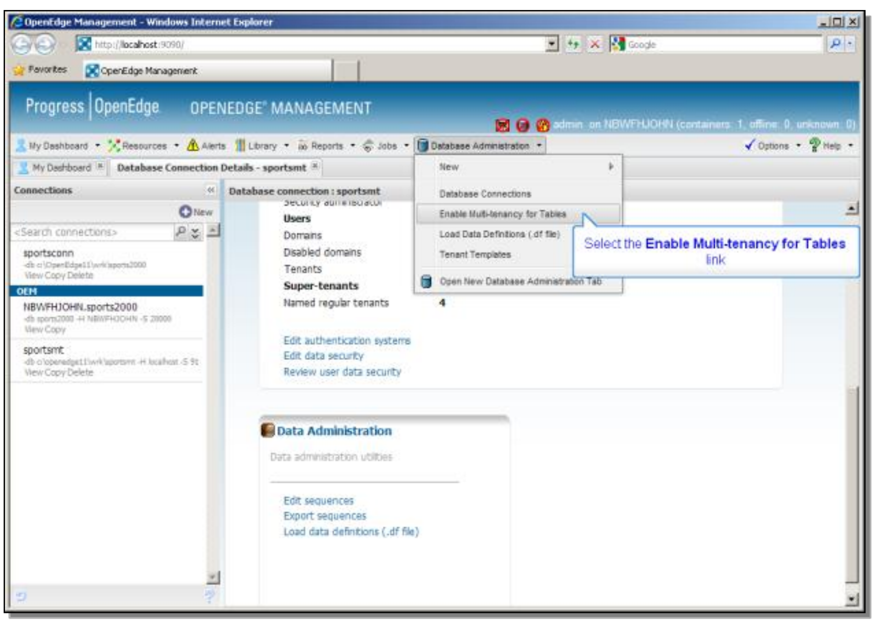


For now I just create NorthSports in the usual way, by clicking **Create tenant.**

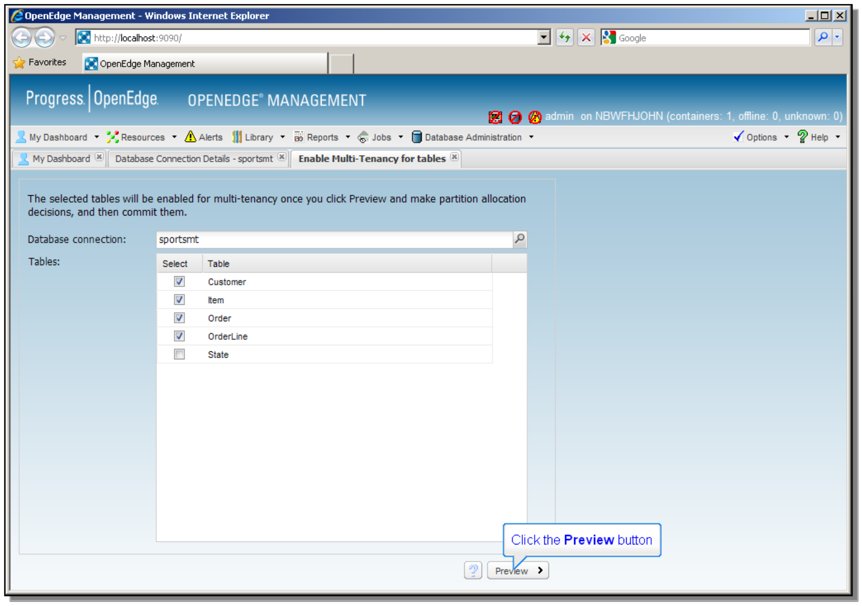
I select the **View** link under the database connection to force a refresh of the details display, which shows that I have four named regular tenants defined for the database, in addition to the default tenant:



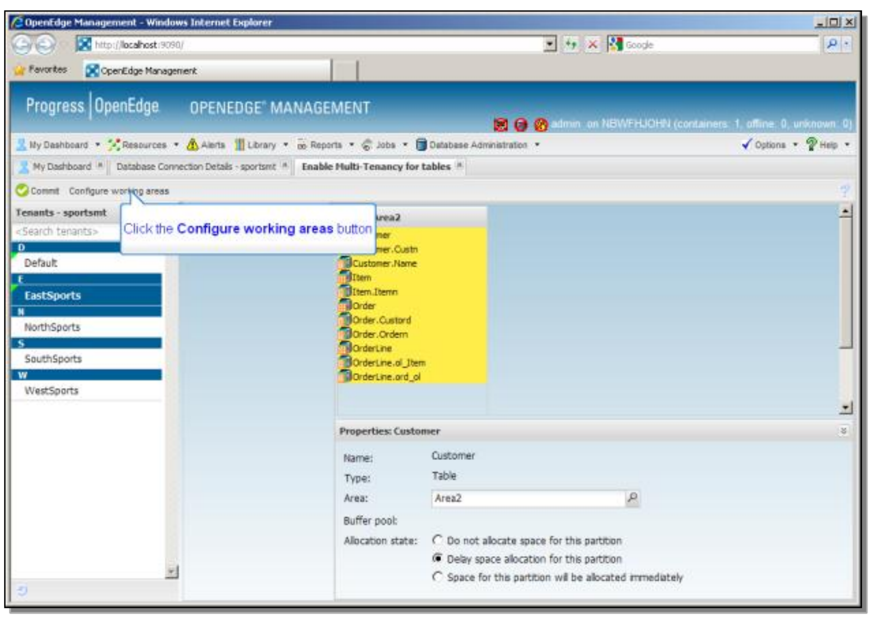
The next job is to define which of my database tables should be made multi-tenant, so that they can hold data securely in separate partitions for each of my tenants. In order to do that, I select the dropdown arrow on the **Database Administration** tab. Generally, the dropdown menus provide shortcuts to common operations you'll also find elsewhere, but in this case this is the one place for enabling tables for multi-tenancy:



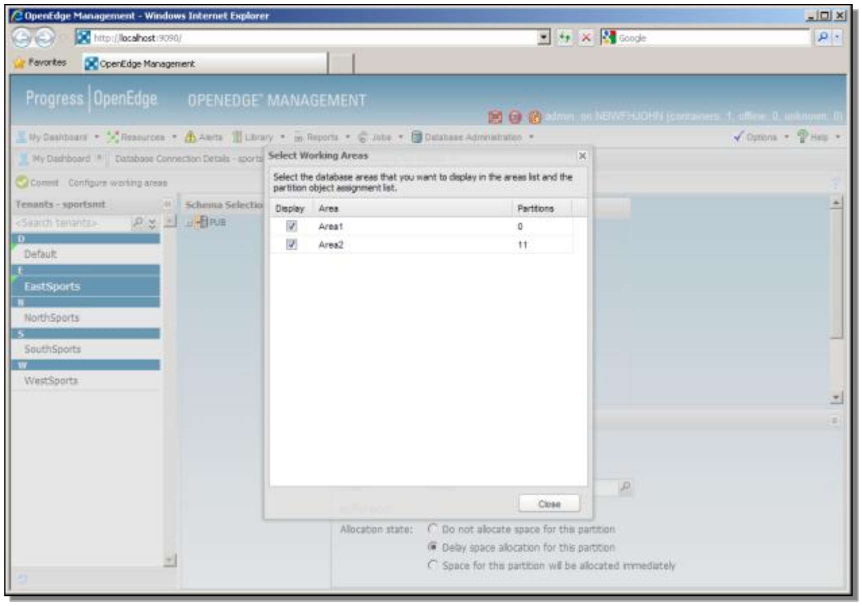
For the sportsmt database, I want all the tables except for the shared State table to be enabled for multi-tenancy:



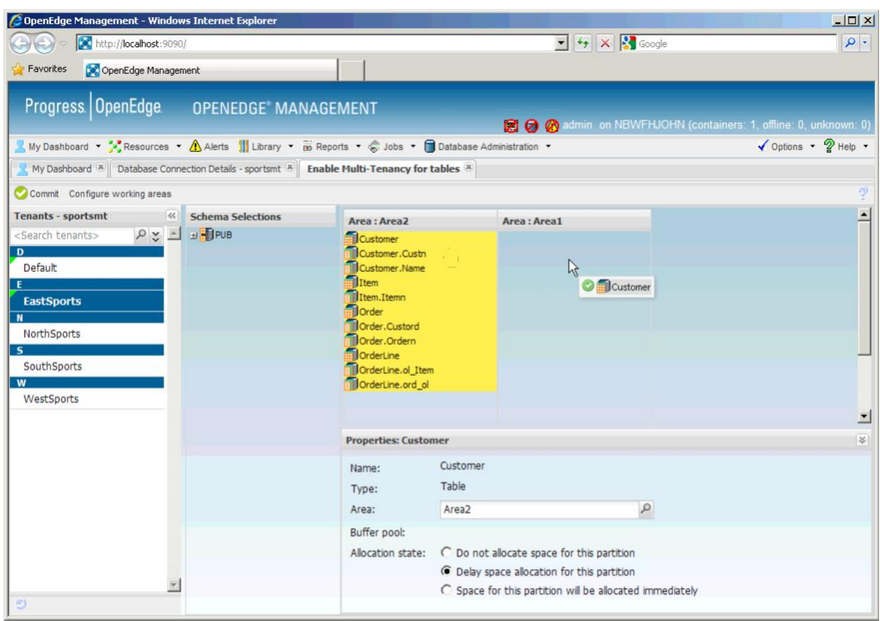
I click the **Preview** button, and I see a visual representation of how data is going to be allocated for all the tables, their indexes, and LOB fields. If I select the EastSports tenant, for instance, I can see that its default allocation area is Area2. The yellow color-coding means that allocation is delayed, which is what I specified as the default for tenant EastSports. If I select a single table, like Customer, I see as shown below that default applied to the Customer table. Because I asked that allocation be delayed, I can make changes to the allocation. First I select the **Configure working areas** button:



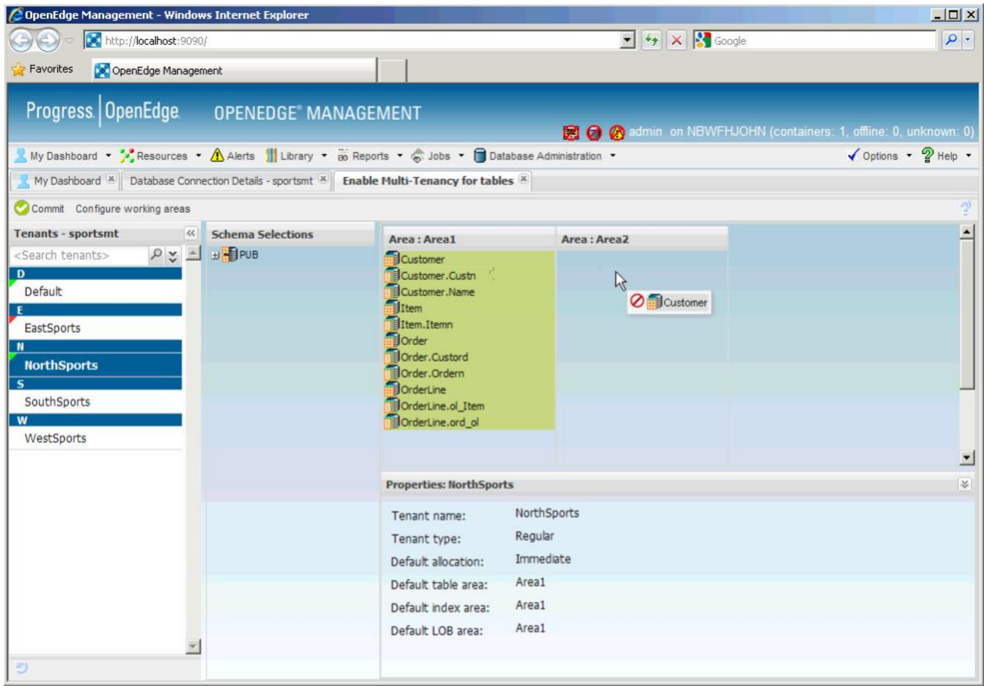
Then I can select Area1 to add to the display:



Now I can select the Customer table again, and drag it over to Area1. This would override the default and place data for the Customer table for the tenant EastSports into Area1:

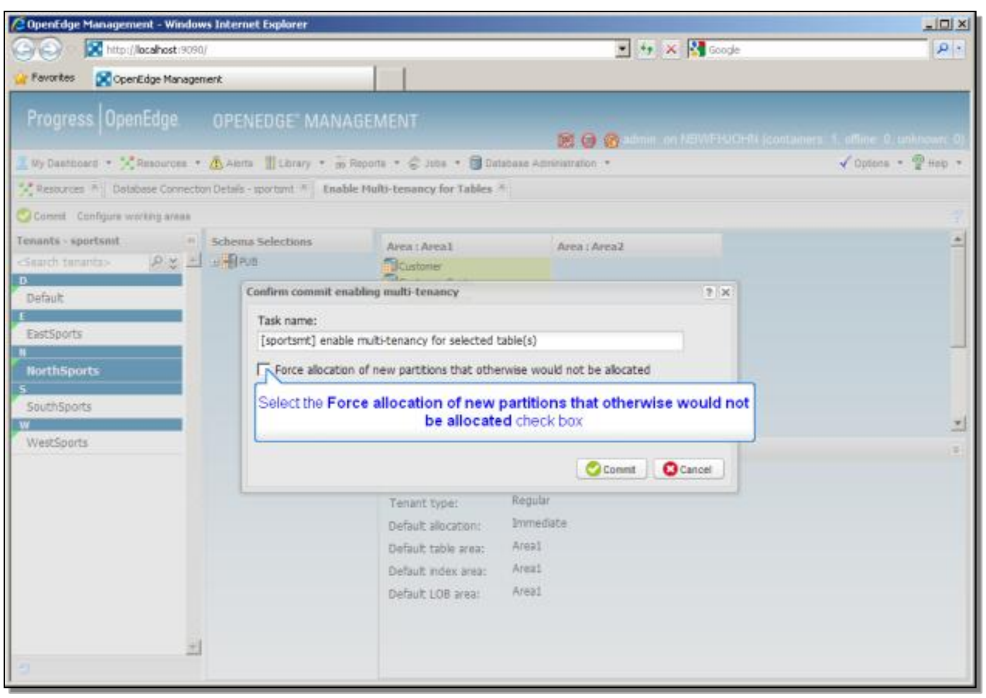


But I'll put it back where it was, and select a different tenant, NorthSports. Its tables and other objects are green, which indicates that they were immediately allocated space in Area1. I can add Area2 to the display, but when I try to drag the Customer table to Area2, which is not the default for this tenant, it doesn't work, because initial space has already been allocated in Area1 for all tables for NorthSports:



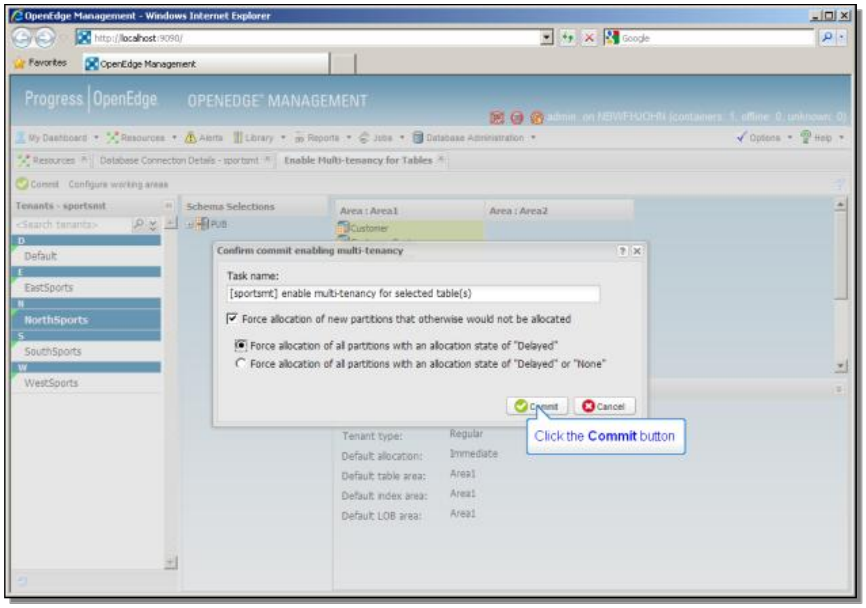
This illustrates the key difference between immediate and delayed allocation.

Now I click the **Commit** button on the upper left. I've just shown how you can adjust the allocation details for tenants that have delayed allocation, but at some point you must allocate partitions before you can actually load data for those tenants, so I do that here:

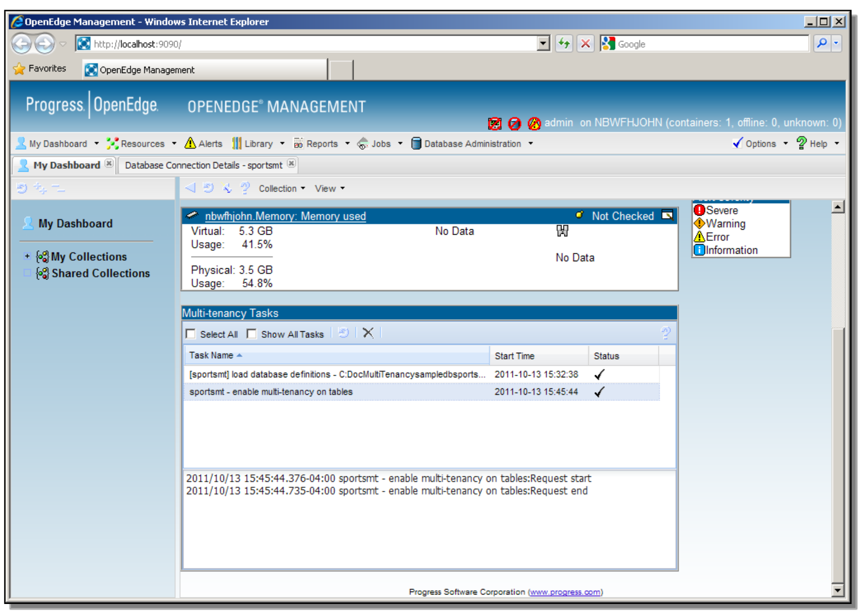


Three of my four tenants have an allocation state of delayed, so it's those partitions that I want to go ahead and finalize here.

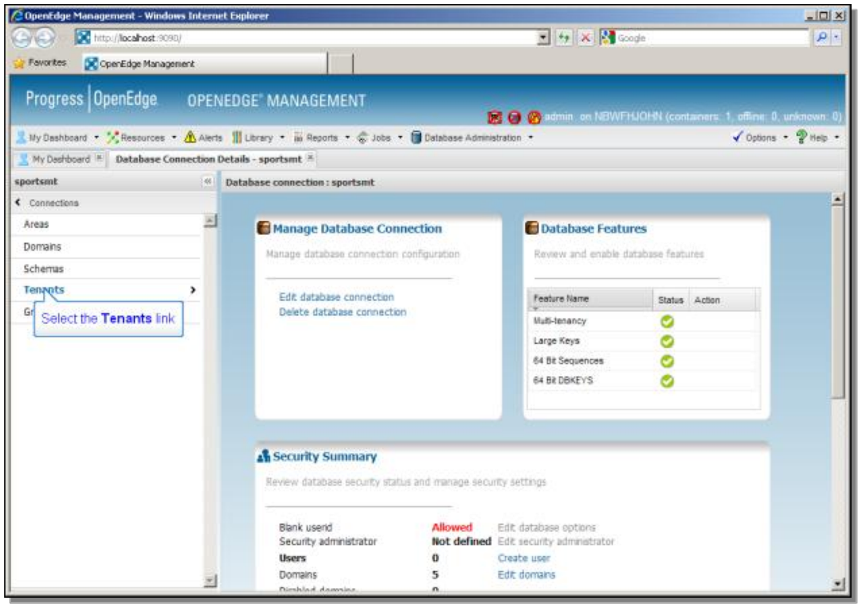
Now I can **Commit** that allocation:



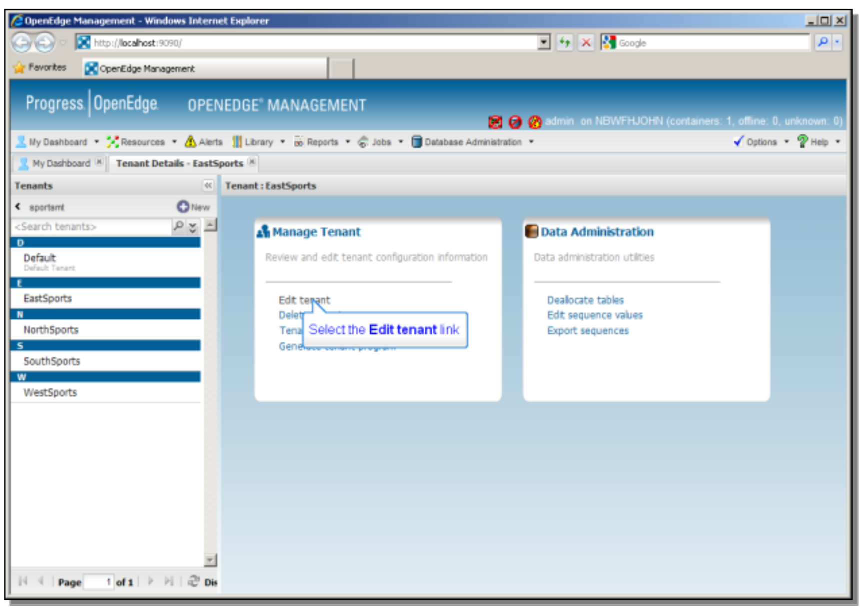
Once again this shows up as a background task I can monitor, and I can see that it's done:



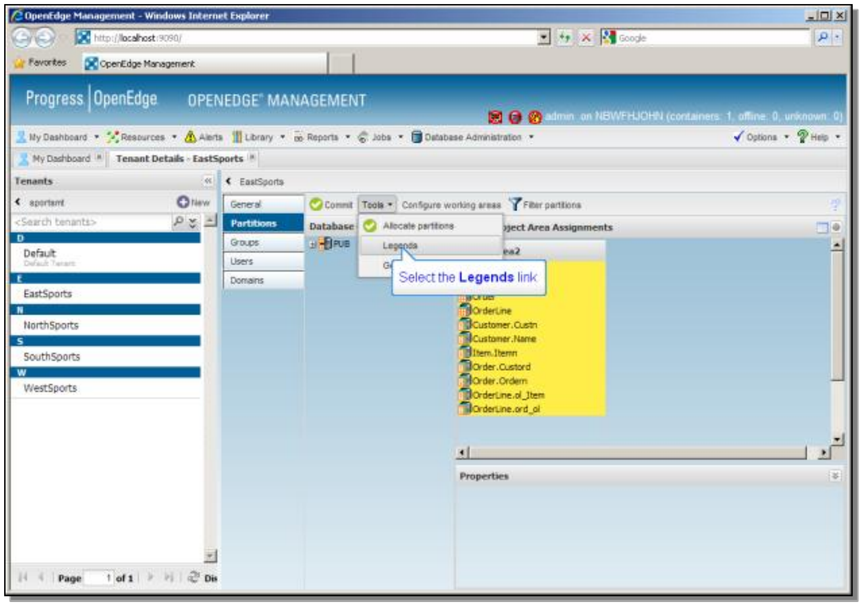
If I select the sportsmt database connection link itself, then a set of choices appears in the list pane:



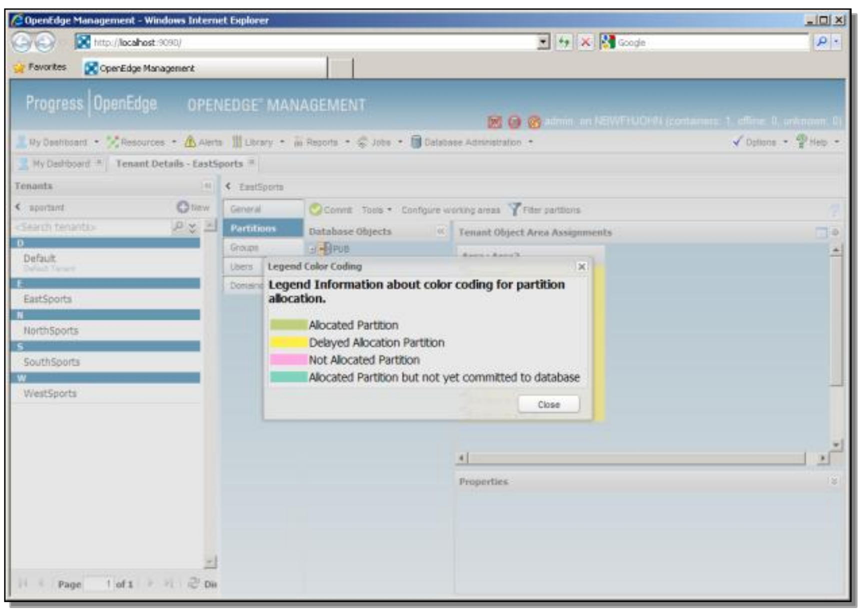
Selecting **Tenants**, and then one of my named tenants, such as EastSports, I can then then select **Edit tenant**:



The basic tenant information comes up with its own side menu. If I select **Partitions** from that menu, then I see the same allocation display I saw when I was first enabling tables for multi-tenancy. Here I can open the helpful **Legends** alert box from the **Tools** menu:

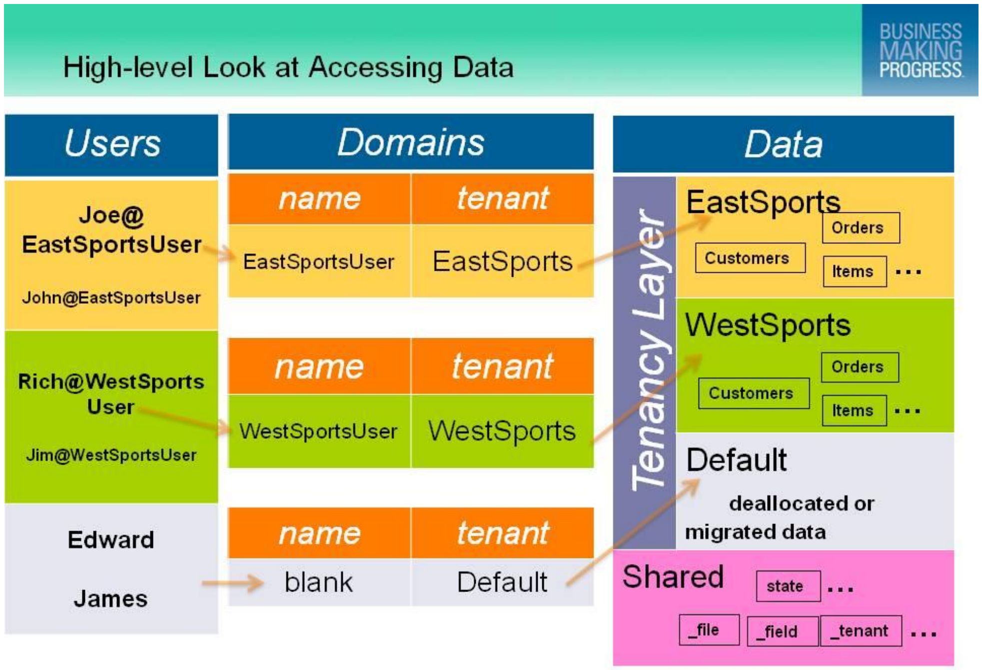


This brings up a message box where you can see the meaning of the color coding of the different allocation rules:



Now I can take advantage of the breadcrumbs built into the new user interface to return to my database connection, and from there to the list of all connections. To close out, I just **View** the sportsmt connection again, and I'm done enabling tables for multi-tenancy. In the next video in the series, and the final section of this paper, I set up the first security domains that allow access to tenant data, and then some users who will access data through those domains.

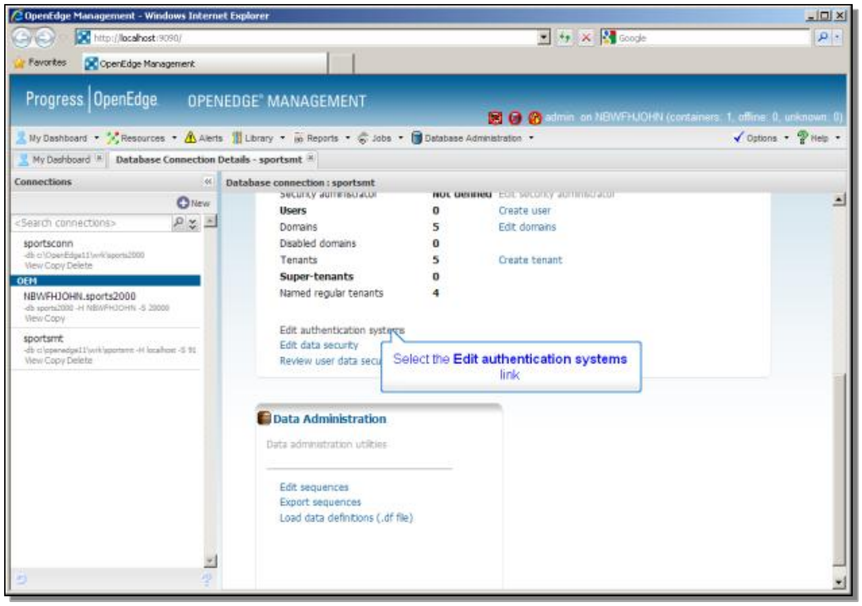
In the Database Administration Console, there are prompts here for creating users and editing the list of domains. Let me talk for a moment about what those things really mean in the context of multi-tenancy, using this diagram:



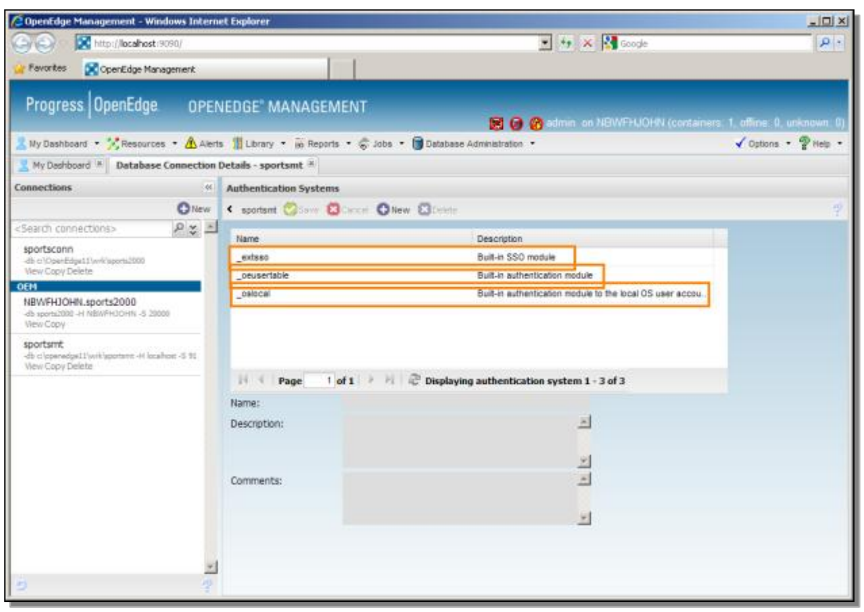
On the right you can see the representation of the database. There's a **tenancy layer** for each of the named tenants, plus a **default tenant**, and within that layer each tenant has allocated for it what we can refer to as a **table instance** for each multi-tenant table, in effect a private copy of that table for the tenant's data.

So how do you control access to each tenant's private data? You do it through what are called **security domains**. Each domain defines a means of access and authentication for a single tenant. In this new system, every user who logs into an application has to provide not just a username and password, but the name of the domain they belong to. The domain acts as the intermediary between the user and the tenant data they're supposed to be able to see. The only exception to this is that, if you don't disallow it, it could be possible for someone to log in just with a username, that is, with a blank domain name. In that case, they would have access to shared tables and whatever may be in the default area. That is an exceptional case that you will probably want to disallow in a production environment.

Let's see how to set up the pieces you need to do all this. In the details frame for the database connection is a link labeled **Edit authentication systems**. That's what we need to look at first:



The authentication systems records, which are part of the schema, represent different ways in which users can be authenticated to get access to the database. There are three built-in ones:

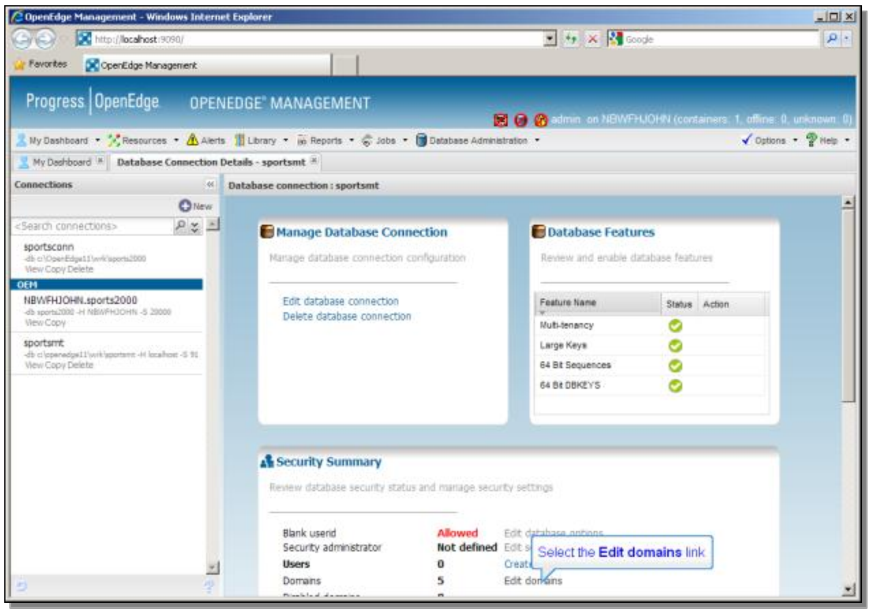


The first, named **_extsso**, provides support for single sign-on access to multiple systems.

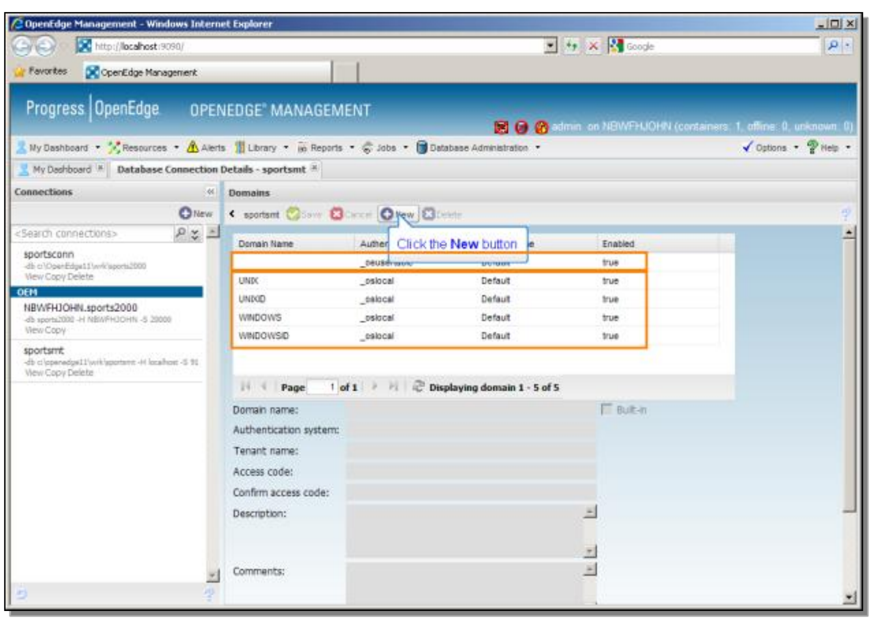
Another built-in authentication system, **_oslocal,** lets you use an operating system-level user ID to access the database.

The third is named **_oeusertable**, which uses the traditional OpenEdge **_User** table to identify and authenticate users. You can define additional authentication systems here, including for instance a new type to represent your own application-level authentication, which is very common in OpenEdge applications. To do that you would just click **New**, and give the new system a name and an optional description. There's no other built-in support for or understanding of your own authentication system, only that you are taking responsibility for it. In order to show you how to create users, I'll use _oeusertable for my example, even though I want to emphasize that there's no need to use the _User mechanism in order to take advantage of multi-tenancy. I just leave these records as they are.

Once I have identified the authentication system that I want to use, whether it's new or built-in, I need to define domains to provide access to the tenants. Remember that every user gets access to its tenant data through a domain. I click the edit domains link:



Here you can see that there are five built-in domains:



One provides access for the blank domain to the Default area, as I showed in the slide, and which I advised that you probably want to disable when you set up your security system. The others provide a starting point for operating system authentication using _oslocal, for Windows and UNIX. I want a domain specifically for each of my tenants, so I click **New**.

I call my first domain **EastSportsUser**, just to emphasize that I'm choosing to use the _User table mechanism for authentication. You might have only one domain definition for a tenant, or you might have several. If you wanted to segregate different sets of users and use different authentication systems for them, you would define multiple domains for the tenant. If you intend to use only one authentication system for a tenant, then that tenant will have just one domain. This may be pretty typical, and in that case there's probably no reason not to make the domain name the same as the tenant name; that's perfectly valid.

In this example I want to make sure you understand that what the users specify when they login is not the tenant name, though they may think of it that way. Rather, it's the domain name, so I make the names a little distinct for that reason, by adding **User** to the end of the name.

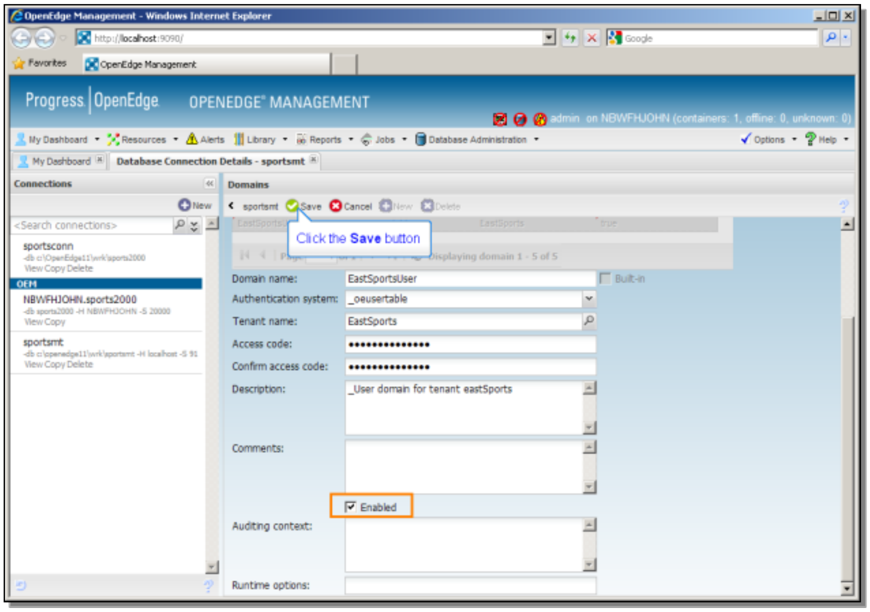
Next I need to match up the domain with its authentication system. As I said earlier, I'm going to use **_oeusertable**, just so that it will make sense for me to create some users later on.

Next I pick a tenant to match up with the domain. Each domain is for a specific tenant. There can be multiple domains for a tenant, but never multiple tenants for a single domain. I choose **EastSports**.

Using domains provides an additional level of security, because each domain has its own **access code**, rather like a password for the domain. All the alternatives for how you might secure this access code are beyond the level of this presentation, but when you define a domain you specify an access code for it, and this is stored in encrypted form as part of the domain record in the schema. I enter an access code, which is suppressed in the display, like a password. I make it the same as the domain name for this simple example.

I can enter a description for the domain if I want, and there are some other optional fields I could fill in that I won't go into here.

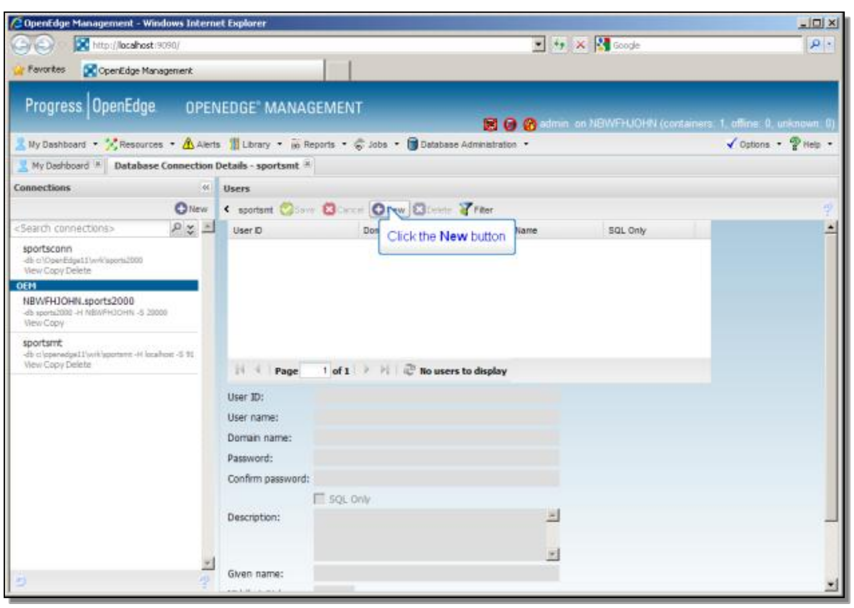
Note that there is a checkbox that as an administrator you can check on or off if you wish to enable or disable a domain, in order to restrict access to the database for that domain's users. This screenshot shows the comnpleted form:



I click **Save** for this first new domain.

Next I have to create at least one domain for each of the other tenants, so that their users can get also into the database. I create one for **WestSports** called **WestSportsUser**, one for **SouthSports** called **SouthSportsUser**, and one for **NorthSports** called **NorthSportsUser**. I now have a total of nine domains, the five built-in ones, and the four I've just finished creating.

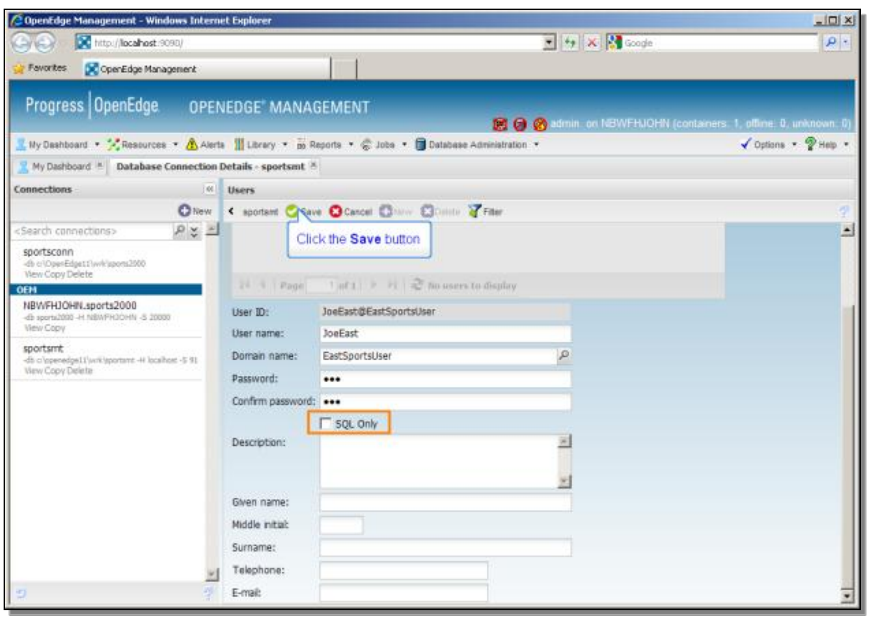
Right above the **edit domains** link is a link that prompts me to **create user**s. Once again I want to emphasize that there's normally no reason to create users in the Admin Console unless you are using the built-in _User table to maintain your users, which means that your domains use the _oeusertable authentication system. Since that's what I'm doing in my example, I do select **create user**. I'm placed into user maintenance, so I need to click **New**:



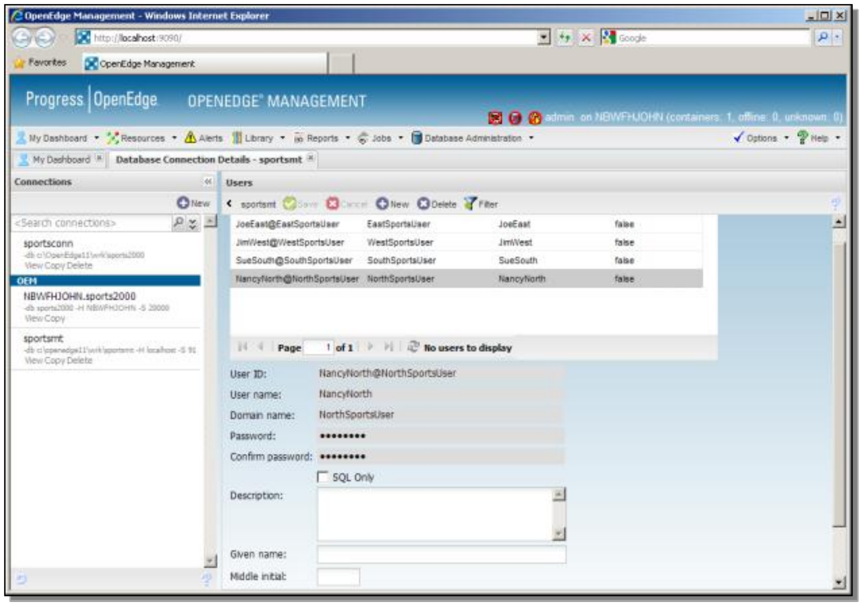
I create one user for each domain, which in turn provides access to one tenant's data. First is **JoeEast**, who works for **EastSports**, so he gets connected to the **EastSportsUser** domain. Keep in mind that this pairing of user name and domain name means that you can easily have the same user name for different domains, two people named Joe for instance who work for different companies that are tenants in your database. I've tacked "East" onto the end of Joe's Userid just to help clarify who is who, but of course in a real application you probably wouldn't do that; it's the domain name that identifies which user gets to see which data.

Next of course I have to enter a password. Remember that I'm just doing _User table maintenance here, just as I would have (and could still do) in the older Data Administration tool. JoeEast's password is just **Joe**. Note that although the Userid format with the "@" sign is the way OpenEdge recognizes the combination of Userid and domain name, an end user might login in any number of ways, where for instance the domain is looked up by the login module and supplied without the user having to enter it directly. The format with the "@" sign is the form recognized as a complete user identifier by OpenEdge.

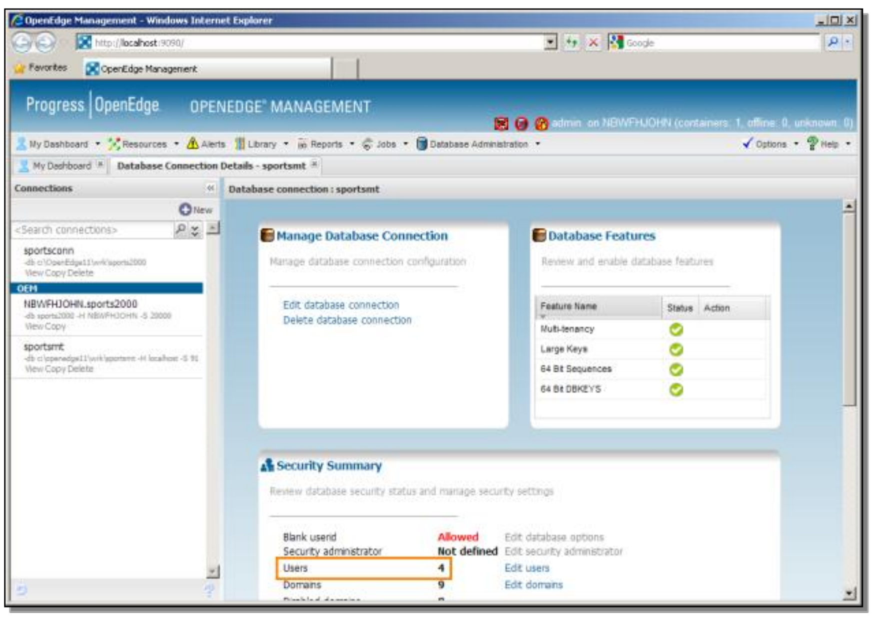
There's other optional user information that I could enter that goes into the user record as well. The one thing I want to note here is the **SQL Only** checkbox. You can use _User records for access to the database through SQL, for reporting, for instance, without having those users be active or recognized from an ABL application. You would check this box to enable that SQL-only access.



I've entered all the information I need to, so I **Save** this first user, and start in to define another user. The second one is Jim, who works for WestSports, so of course Jim's Userid is connected to the WestSportsUser domain. In all, I create four users, one for each domain, to provide access in turn to each tenant's data. Here you see all four users:

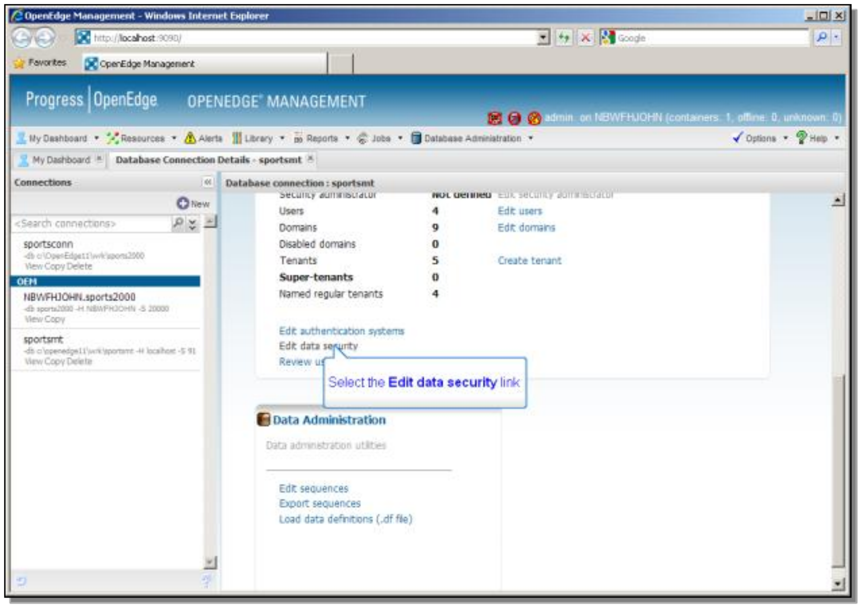


I return to the main display for sportsmt, and the **Security Summary** shows that I now have four users:

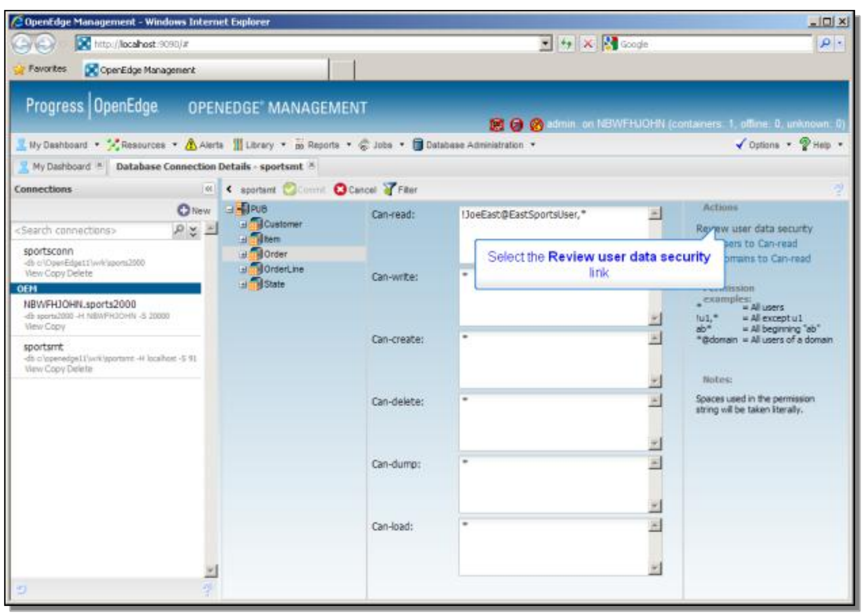


At this point I have everything set up in the Admin Console to allow people to actually log into the database and start working. There's one more thing I should show you before finishing up this paper.

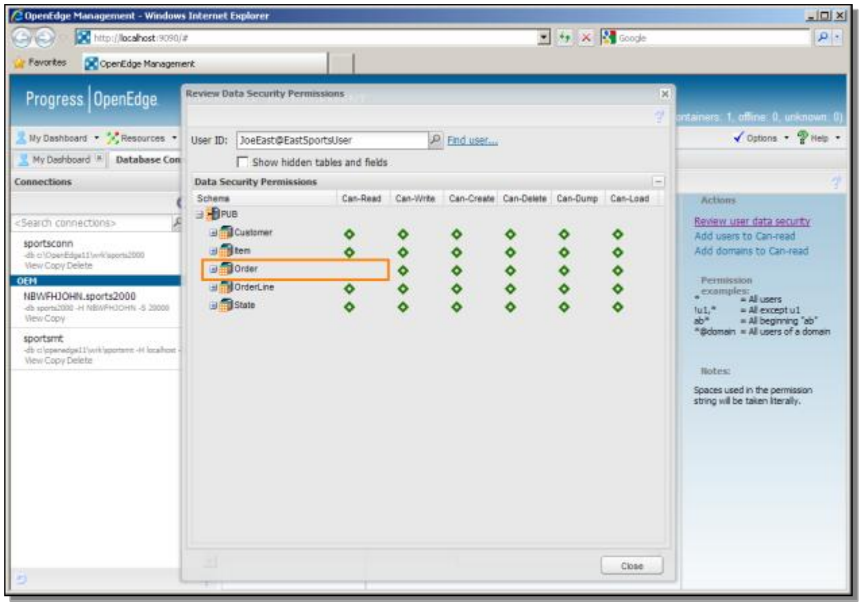
Toward the bottom of the Security Summary area is a link you can select to edit data security for individual users and groups of users:



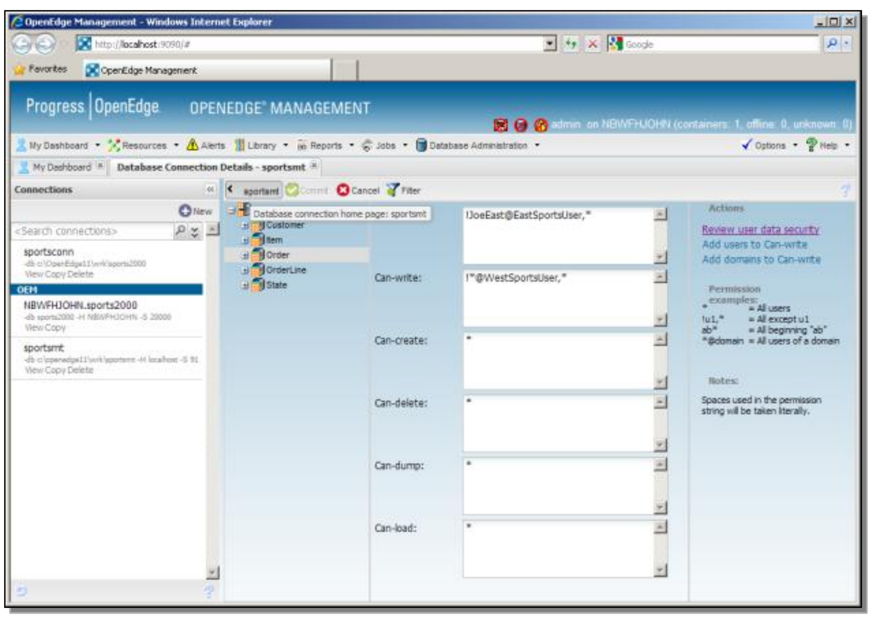
You're familiar with all the **Can-Read** and **Can-Write** settings for users. They're the same as they've always been. However, there's one new wrinkle I want to make sure you're aware of. If I select the Order table, and then the Can-Read privilege setting, for example, then if I want to enable or disable a user, I need to include the user's domain name, with the "@" sign in between, since the domain name is an essential qualifier for the Userid, to make sure each combination is unique. Below I've removed Read permission on the Order table for JoeEast:



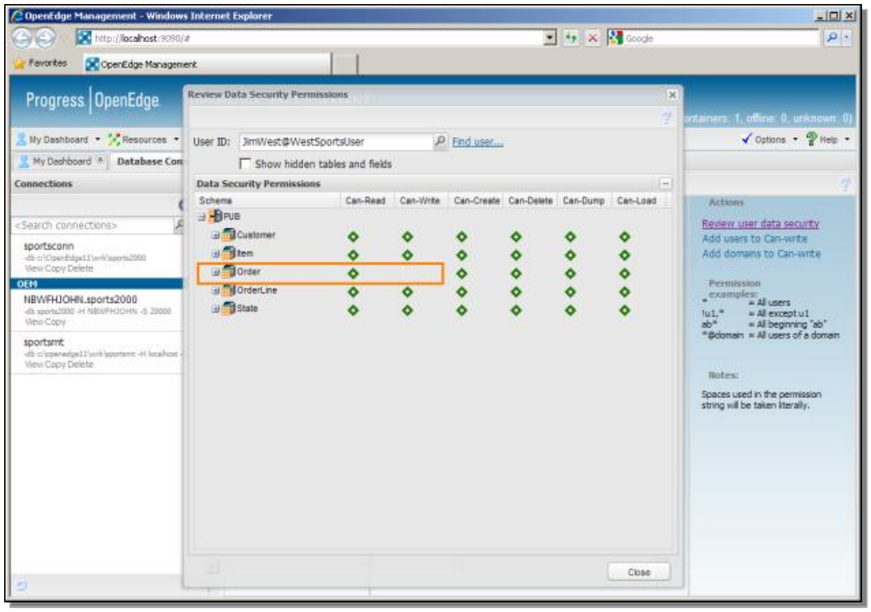
If I press **Commit** to save the change, I can then select **Review user data security** to take a look at the effects of the setting. I click the **Find user** link, select JoeEast, and I can see that I have successfully turned off Can-Read permission for JoeEast for the Order table:



There’s one other thing you need to understand when working with domains. You can enable or disable permission for all users in a domain by using the domain name, preceded by an asterisk (*) and the at sign (@), in a setting. Below for example I’ve removed Can-Write permission on the Order table for all users in the WestSportsUser domain:



Once again I can confirm the effects of that, by retrieving JimWest, the one WestSports user, and I see that he no longer has Can-Write permission on the Order table:



The same would be true for anyone else in the WestSportsUser domain. That gives you an idea of what's new in user permission settings.

Now that I've set up everything I need to in the Admin Console, I need to show you how you actually program your ABL login to provide the proper authentication of a user to the domain he uses to access data for his tenant. That will be in the next videos in this series, along with the white paper that accompanies them.