Configuring Instance-Level security for CQL\_CSM

Version 1.3.1

By Mark Grand  
mgrand@emory.edu

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| January 25, 2010 | 1.1 |  | Mark Grand |
| July 27, 2010 | 1.3.1 |  | Mark Grand |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

This article explains some details of how the CQL\_CSM replacement CQL processor is configured by working through a detailed example. It describes a security policy, shows screen shots of the CMS configuration to implement the policy, and then shows some CQL queries and the SQL that is produced based on the CQL and the CMS configuration.

This document does not describe installation details for CQL\_CSL. Installation details are described in the software distribution’s README file.

The security policy that we describe applies to a caGrid data service that provides read-only accesses to the database of an application called OpenClinica. OpenClinica is an open-source clinical trial management application.

The classes in the data service’s domain model include:

* Study: Instances of study describe a clinical research study that involves many subjects.
* Subjects are people who participate in one or more studies.

A security policy that we discuss governs access to study and subject objects. If someone is an investigator for a study, they are allowed to see information describing the study and also information describing the subjects who are participating in the study. More specifically, investigators for a study are allowed to read the study object that describes the study and also subject objects that describe subjects that participate in the study.

The rest of this article describes how CQL\_CSM works by following the details of an example to implement this authorization policy.

# CSM Configuration

Configuring CSM to support this policy involves a number of different types of data. These are:

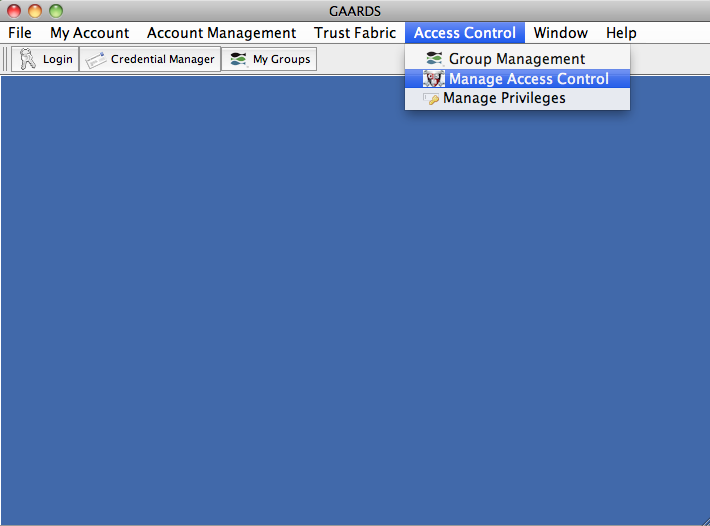
* Instance-level configuration (in the table CSM\_FILTER\_CLAUSE)
* Protection Elements (in the table CSM\_PROTECTION\_ELEMENT)
* Protection Groups (in the table CSM\_PROTECTION\_GROUP)
* Roles (in the table CSM\_ROLE)
* Groups (in the tables CSM\_GROUP, CSM\_USER and CSM\_USER\_GROUP)
* Permissions (in the table CSM\_USER\_GROUP\_ROLE\_PG)

To show how these types of data are configured, we use the UI screens for the CSM Grid Service, a tool for managing CSM configuration data.

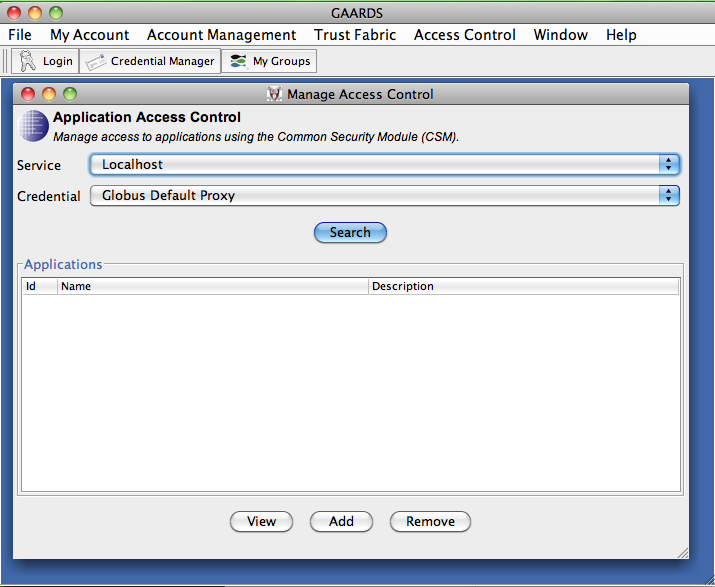
The following description assumes that you have read the documentation for the CSM GAARDS UI and know how to get to the **Manage Access Control** window.

You begin the process of configuring instance-level authorization by using the CSM-UI GAARDS client. Once you have started it up, log into caGrid using an identity that you authorized to administer CSM by using the CSM service’s  
ant addAdmin

To access most CSM administration features, we must use the **Application Access Control** window. We click on the**Access Control** menu item and choose the **Manage Access Control** item to launch the window.

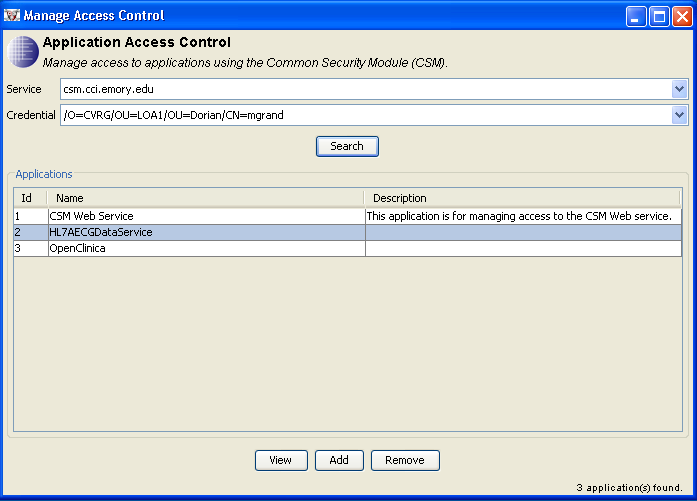


The **Manage Access Control** window appears.



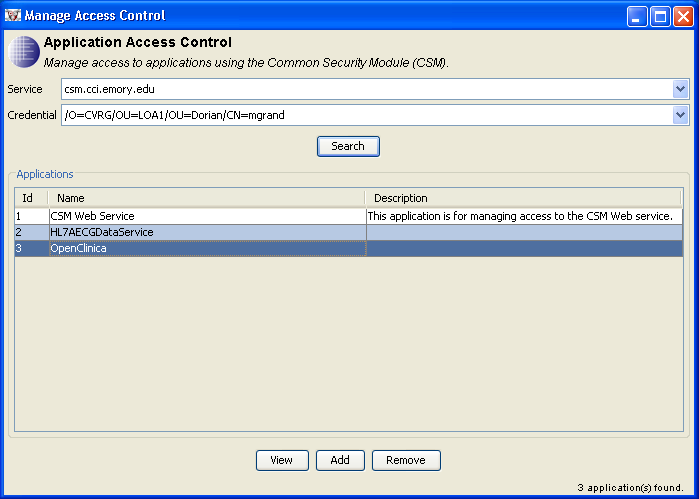
On the top of the **Manage Access Control** window in the **Service** field, we select the CSM service that we will use to administer authorization policies. This is easy if the UI is configured to display the URL of only one CSM service. Otherwise, we just need to know the correct CSM service to choose.[[1]](#footnote-1)

In the **Credential** field, we select the credential that we will use to do the administration. This should be the grid identity we just logged in with.[[2]](#footnote-2)

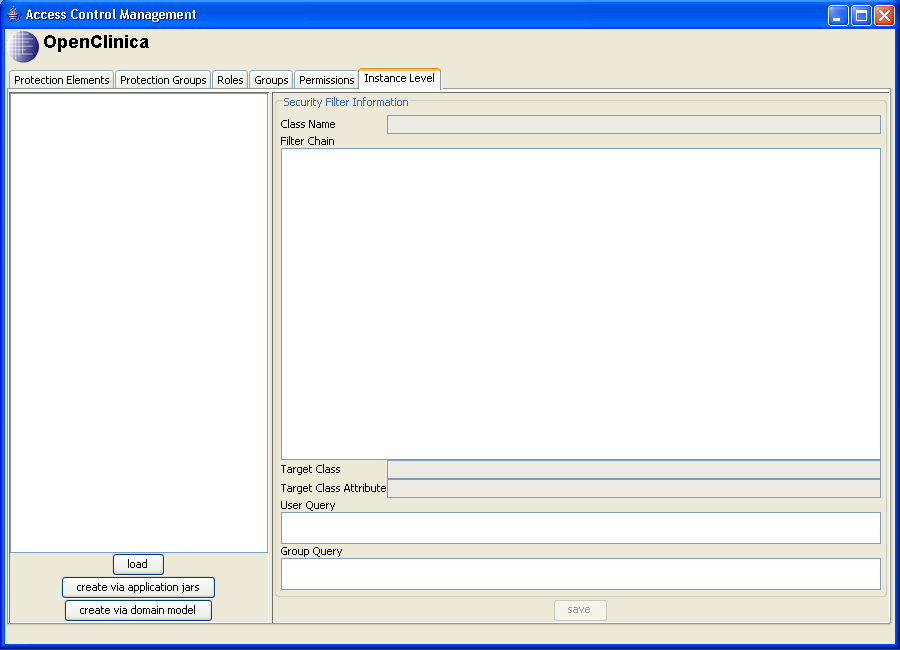
We click on the **Search** button. A list of services and applications for which the CSM service manages authorizations appears in the lower half of the **Manage Access Control** window. It looks like this: 

The rows of the list have a background that alternates between white and light blue. This may make the list easer to read when there are many rows. When there are just a few rows, do not mistake the light blue for a selected row.

We select the application for which we will be managing authorizations. This is what the list looks like with a selected row:



We click the **View** button. after clicking on the **View button, the \*Application Access Management** window appears to administer CSM policies for the OpenClinica service. We click on its **Instance Level** tab. It looks like this.



The next step is to load the data service's domain model into the CSM service. This is necessary because the CSM service needs the data model to understand instance-level security policies. The Domain model is not stored in the database, so we need to load it from a file.

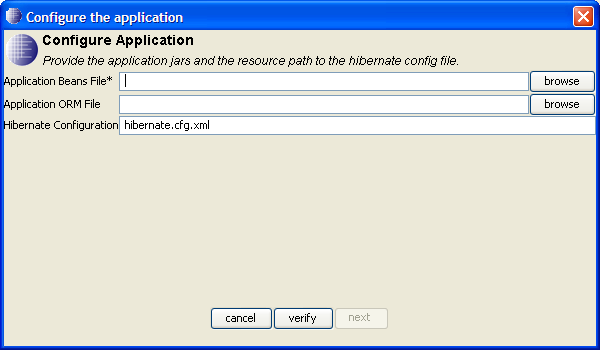
There are two ways of loading the domain model. Which way to use depends on whether you are loading a domain model for a caCORE-based data service or for something else.

caCORE-based data services have two .jar files that contain their hibernate model. These files have names that follow the pattern databaseName-beans.jar and databaseName-orm.jar. If the name of the database is openclinica then the names of the files are openclinica-beans.jar and openclinica-orm.jar.

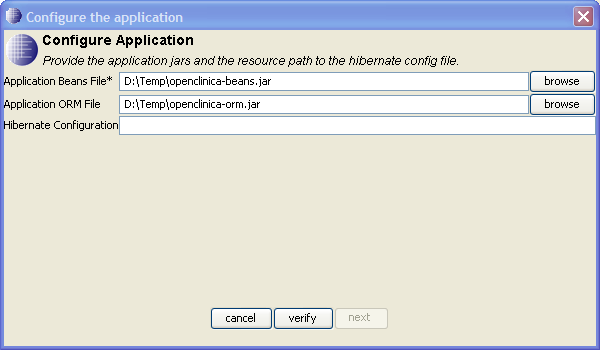
For data services not based on caCORE, we get the domain model from a domain model file in the data service’s etc directory. The name of the file is follows the pattern *ServiceName*‑domainmodel.xml. For example, if the service name is HL7AECGDataService then the name of the file is HL7AECGDataService‑domainmodel.xml.

To initiate loading the domain model from a domain model file, click on the **create via domain model** button. The rest of the process is similar to the process of loading the domain model from .jar files, which is illustrated in the following paragraphs.

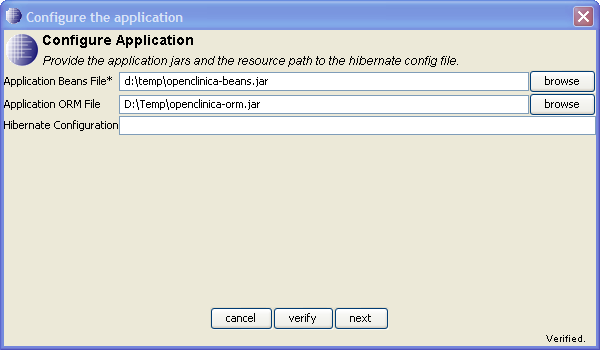
To begin the process of loading the domain model from the .jar files, we click on the **create via application jars** button. The **Configure the Application** dialog appears. It looks like this.



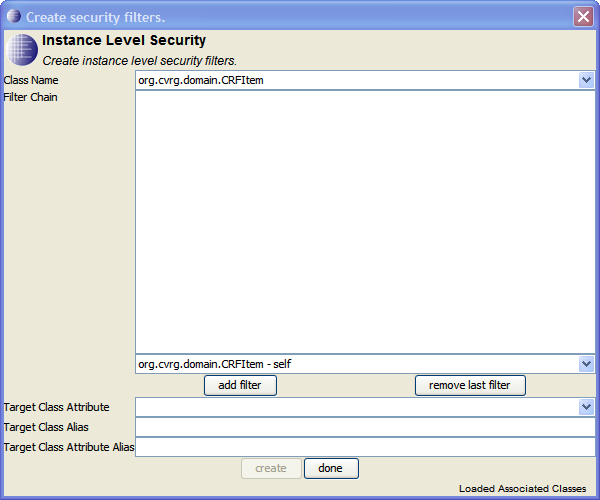
We use the **browse** buttons to fill in the **Application Beans File** field and the **Application ORM File** field with the paths of the two .jar files. We also delete the contents of the **Hibernate Configuration** field, which should be blank. When we are done, it looks like this:



We click on the **verify** button. A progress indicator appears while the UI reads the file and verifies the validity of its contents. When the verification finishes successfully, the progress indicator disappears and is replaced by the word “Verified” in the lower right corner of the dialog. The **next** button is now enabled. The dialog now looks like this:



Click on the **next** button. The **Configure the application** dialog disappears and a **Create security filters** dialog appears. It looks like this:



Before continuing with the configuration process, we need to understand what a CSM security filter is.

A security filter specifies a class in the domain model and an attribute that may be directly or indirectly associated with the class. If the attribute is indirectly associated with a class, then a sequence of one or more associations is specified that determines how to navigate from the filter class to the class that the attribute is directly associated with. The class that the attribute is directly associated with is called the target class.

If there are no filters associated with a class, then no authorization is required to read its instances. If there are filters associated with a class then a user is authorized to read only those instances of the class whose filter attribute has an authorized value. The authorized values are determined by what protection elements a user is authorized to read. Protection elements are described in the documentation for CSM.

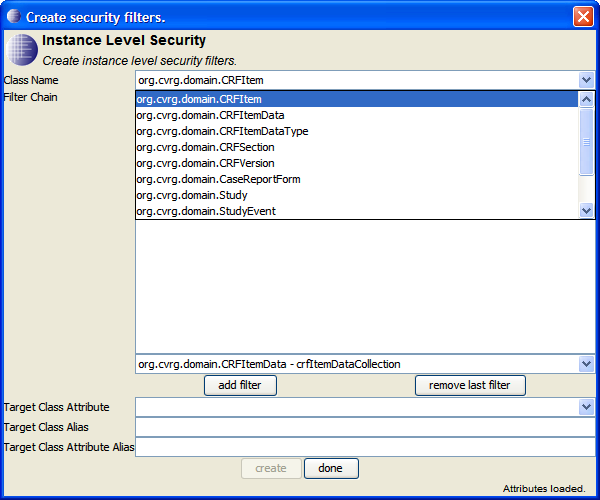
To determine what values a user is authorized to read through a CSM filter, we look for protection elements whose Object ID field has a value that equals the name of the filter’s target class and whose Attribute Name field has a value that equals the name of the filter’s attribute. If the user is authorized to read the protection element, then the user is authorized to read instances of the filtered class whose associated attribute equals the value of the protection element’s Attribute Value field.

Having explained this, we will continue by working through an example. Suppose that we are creating CSM authorization information for a data service that accesses data in an OpenClinica database. This database contains data about clinical studies. Each study in the database will be associated with subjects who participate in the study. A subject may be associated with more than one study.

There are two authorization policies that we what to implement using CSM:

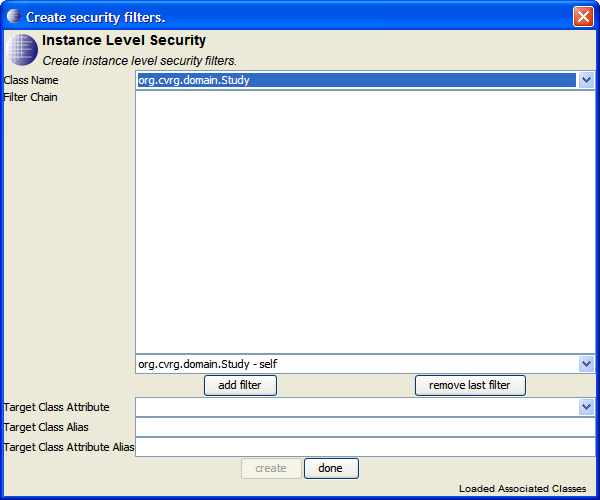
* Users can only get study objects for studies that they have been explicitly authorized for.
* Users can only get subject objects of subject who are participating in a study that the user is authorized for.

Bearing all this in mind, we come back to the **Create security filters** dialog and look at the choices we have to select from for the **Class Name** field:



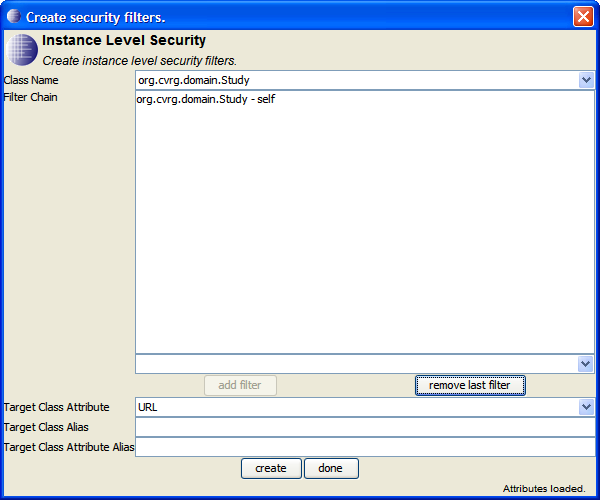
The choices that we have to choose from are the classes in the domain model.

We begin creating our first filter by selecting org.cvrg.domain.Study. The **Create security filters** dialog now looks like this:



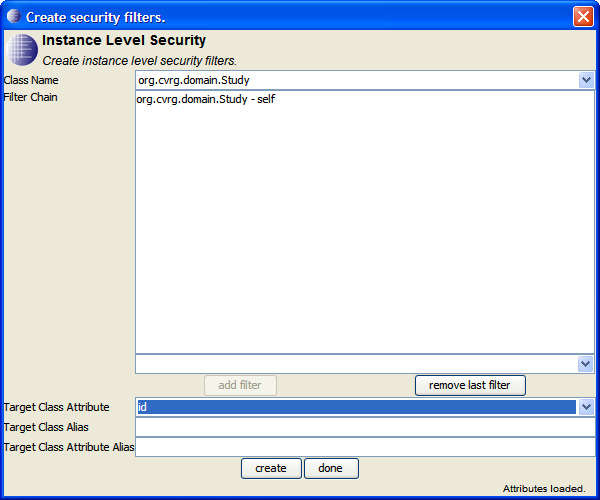
The **Filter Chain** field will contain the sequence of associations from the filter class to the target class. This is currently empty. The unlabeled drop-down list under the **Filter Chain** field allows us to select associations to add to the list. The format of the choices in this drop-down list is *targetClassName ‑ AssociationName*. In addition to the associations defined in the domain model, CSM pretends that every class has an association named “self” that associates the class with itself. CSM also requires that the filter chain have at least one association in it, so we will use the “self” association.

Clicking the **add filter** button adds the association selected in the drop-down list to the **Filter Chain** field. After we click on the **add filter** button, the dialog looks like this:



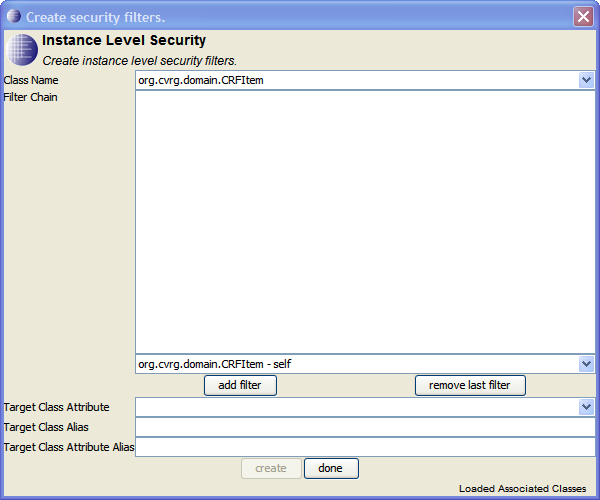
If we make a mistake and add the wrong association, we can remove the last association we added by clicking on the **remove last filter** button.

Now that we have identified a target class, there is a list of target class attributes for us to select from. The org.cvrg.domain.Study class has an attribute with the name id that contains a number value that uniquely identifies the study. We will select id as the value of the **Target Class Attribute** field.

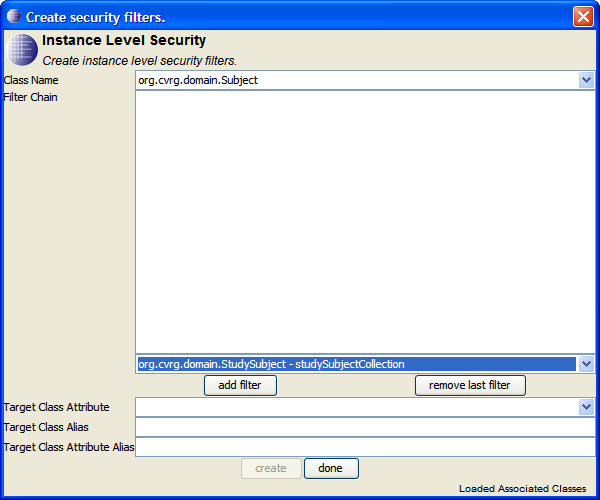


We are done specifying a filter that requires uses be explicitly authorized to read Study objects. We always leave the Target Class Alias and Target Class Attribute Alias fields blank.

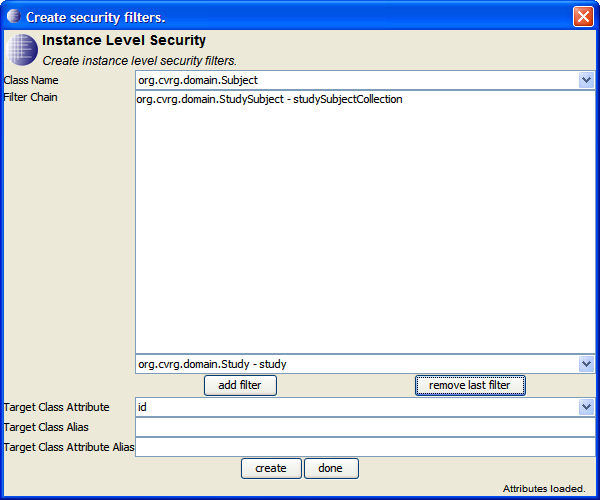
We click on the **create** button. We are now ready to enter another filter.



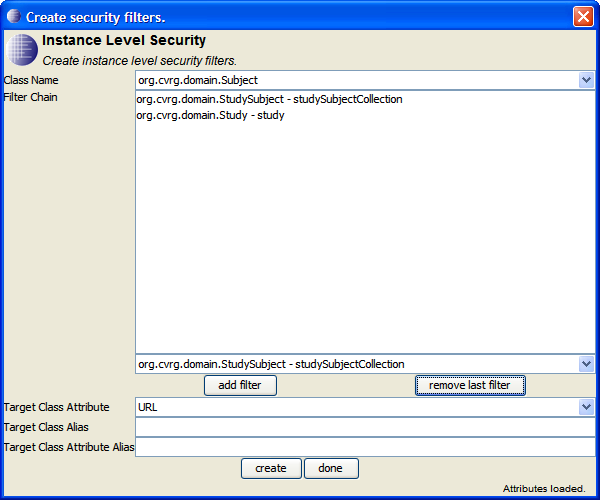
The next filter we create will be associated with the Subject class. Its target class with by Study and the target attribute will be the Study class’s id attribute. We begin by selecting the org.cvrg.domain.Subject class in the **Class Name** field.



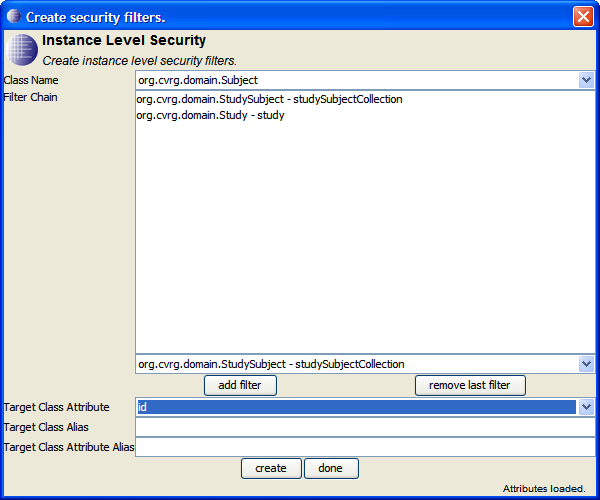
In the domain model, the Subject and Study classes are not directly associated. Instead, they are association through a third class: StudySubject. We select the association from Subject to Study Subject and click on the **add filter** button.



We next select the association from StudySubject to Study and click on the **add filter** button again.

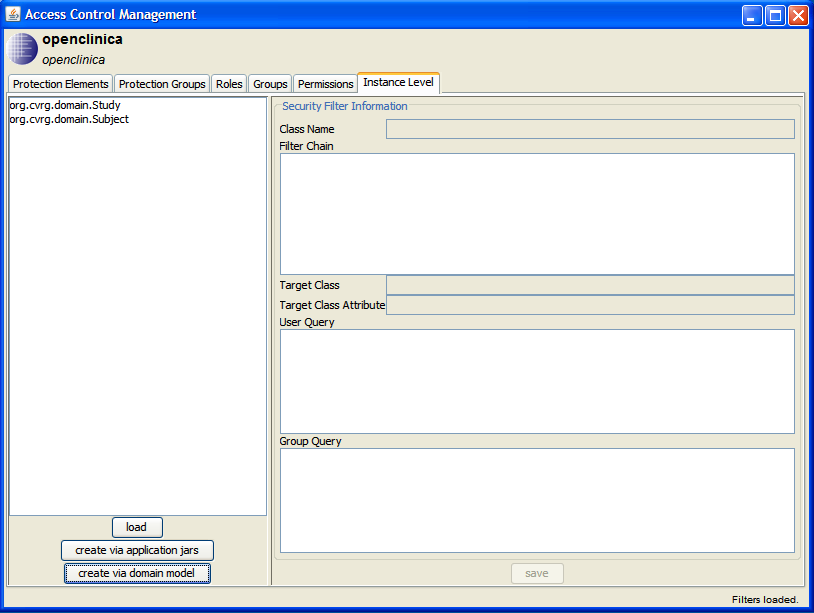


The target class is now Study. Since we used the Study class’s id attribute to identify studies for the other filter, we will do the same for this filter.

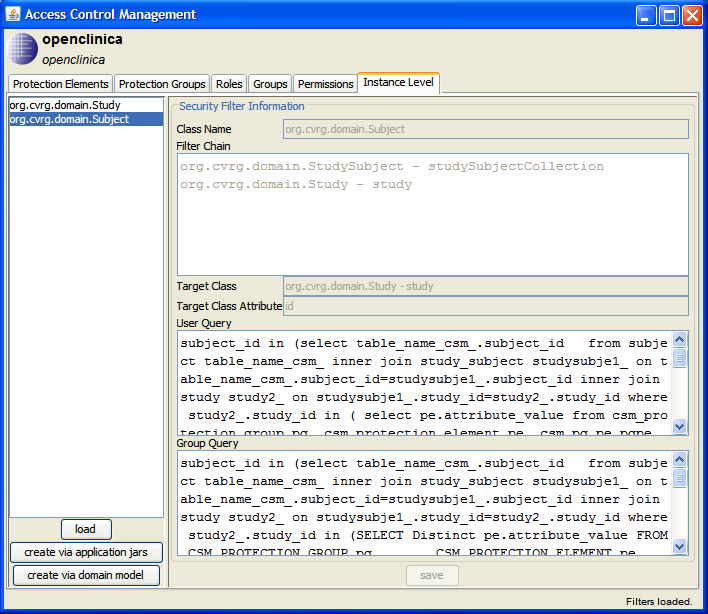


We click on the **create** button to create the second filter. Since we have created all the filters we need, we click on the **done** button. The **Create security filters** dialog disappears.

At this point the **Access Control Management** window does not reflect that new filters that we just added. To get it to update itself, click on the **load** button.



The filters are listed by their associated class. Selecting a filter shows its details.



The administration tool generates two pieces of SQL from these two fields to generate two pieces of SQL.[[3]](#footnote-3) These are Boolean expressions that CQL\_CSL will incorporate into the where clause of any query that references study objects. These expressions are “OR”ed together with each other and the result is “AND”ed with the other constraints in the where clause.

One of the SQL expressions is to determine if the user who issued a query is authorized to read specific study objects. The other SQL expression is to determine if the user who issued a query is a member of a group that is authorized to read specific study objects. Since the SQL expressions are too big to fit on the screen, here is what the SQL for the group part of the constraint looks like:

study\_id in  
 (select table\_name\_csm\_.study\_id from study table\_name\_csm\_  
 where table\_name\_csm\_.study\_id in  
 (SELECT Distinct pe.attribute\_value   
 FROM CSM\_PROTECTION\_GROUP pg, CSM\_PROTECTION\_ELEMENT pe, CSM\_PG\_PE pgpe,  
 CSM\_USER\_GROUP\_ROLE\_PG ugrpg, CSM\_GROUP g, CSM\_ROLE\_PRIVILEGE rp,  
 CSM\_ROLE r, CSM\_PRIVILEGE p  
 WHERE ugrpg.role\_id = r.role\_id  
 AND ugrpg.group\_id = g.group\_id  
 AND ugrpg.protection\_group\_id  
 = ANY (select pg1.protection\_group\_id  
 from csm\_protection\_group pg1  
 where pg1.protection\_group\_id = pg.protection\_group\_id  
 OR pg1.protection\_group\_id  
 = (select pg2.parent\_protection\_group\_id  
 from csm\_protection\_group pg2  
 where pg2.protection\_group\_id  
 = pg.protection\_group\_id))  
 AND pg.protection\_group\_id = pgpe.protection\_group\_id  
 AND pgpe.protection\_element\_id = pe.protection\_element\_id  
 AND r.role\_id = rp.role\_id  
 AND rp.privilege\_id = p.privilege\_id  
 AND pe.object\_id= 'org.cvrg.domain.Study'  
 AND p.privilege\_name='READ'  
 AND g.group\_name IN (:GROUP\_NAMES )  
 AND pe.application\_id=:APPLICATION\_ID))

When CQL\_CSM inserts this in a where clause, it substitutes for “:GROUP\_NAMES” a comma separated list of the names of groups that the user who issued the CQL query is known to be a member of. It substitutes for “:APPLICATION\_ID” the identifier it uses internally for the OpenClinica application.

The SQL expression for the user part of the constraint is similar to the expression for the group. The difference between the two is at the end. Here is what the end of the user SQL expression looks like:

and p.privilege\_name='READ'  
 and u.login\_name=:USER\_NAME  
 and pe.application\_id=:APPLICATION\_ID))

CQL\_CSM substitutes for “:USER\_NAME” the identity of the user that issued the CQL query.

Multiple filters may be specified for one object. CQL\_CSM ANDs them together.

Not shown in the preceding screen shot is the instance level information for filtering subjects. For filtering subjects, the value of the Target Class field is org.cvrg.domain.Study – study. This indicates that subjects should be filtered by an attribute of the study object they are associated with through the association role named study. The value of the Target Class Attribute field is id, indicating that it is the value of the study object’s id field that will be used to filter subjects.

For comparison, the group SQL expression generated for filtering subjects is shown:

subject\_id in  
 (select table\_name\_csm\_.subject\_id  
 from subject table\_name\_csm\_  
 inner join study\_subject studysubje1\_  
 on table\_name\_csm\_.subject\_id=studysubje1\_.subject\_id  
 inner join study study2\_ on studysubje1\_.study\_id=study2\_.study\_id  
 where study2\_.study\_id in  
 (SELECT Distinct pe.attribute\_value  
 FROM CSM\_PROTECTION\_GROUP pg, CSM\_PROTECTION\_ELEMENT pe,  
 CSM\_PG\_PE pgpe, CSM\_USER\_GROUP\_ROLE\_PG ugrpg, CSM\_GROUP g,  
 CSM\_ROLE\_PRIVILEGE rp, CSM\_ROLE r, CSM\_PRIVILEGE p  
 WHERE ugrpg.role\_id = r.role\_id  
 AND ugrpg.group\_id = g.group\_id  
 AND ugrpg.protection\_group\_id  
 = ANY (select pg1.protection\_group\_id  
 from csm\_protection\_group pg1  
 where pg1.protection\_group\_id  
 = pg.protection\_group\_id  
 OR pg1.protection\_group\_id  
 = (select pg2.parent\_protection\_group\_id  
 from csm\_protection\_group pg2  
 where pg2.protection\_group\_id  
 = pg.protection\_group\_id) )  
 AND pg.protection\_group\_id = pgpe.protection\_group\_id  
 AND pgpe.protection\_element\_id = pe.protection\_element\_id  
 AND r.role\_id = rp.role\_id  
 AND rp.privilege\_id = p.privilege\_id  
 AND pe.object\_id= 'org.cvrg.domain.Study'  
 AND p.privilege\_name='READ'  
 AND g.group\_name IN (:GROUP\_NAMES )  
 AND pe.application\_id=:APPLICATION\_ID))

Both of types of generated SQL expressions begin with the identifier “study\_id”. CQL\_CSM expects and requires that these SQL expressions begin with an identifier that consists of the class name followed by “\_id”. Before inserting one of these expressions into a where clause, CQL\_CSM prepends the alias it is using for the table whose contents are being filtered.

The SQL determines what values are acceptable for the attribute they are filtering on based on which protection elements are directly or indirectly associated with a combination of the user that issued the CQL query and a role that has the READ privilege. From this set of protection elements, the SQL expression uses the protection elements whose object ID field matches the name of the class whose attribute is being used as the basis for a filter.

The following screen shot shows such a protection element for filtering based on the id attribute of study objects.



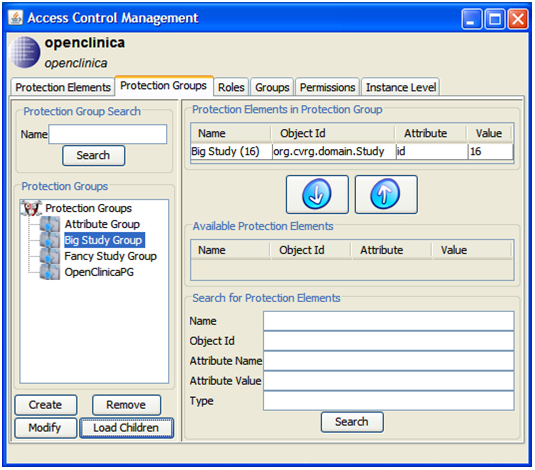
The protection element in the above screen shot allows filters based on the id attribute of study objects to match study objects that have 16 as the value of their id attribute. Here are descriptions of the fields shown in the above screen shot:

* Id  
  This is the internal id supplied by the CSM database. It is a read-only field.
* Name  
  This is the name of the protection element. This has no significance to filters. It should be a name that is signification to administrators. In this case, the name “Big Study” is used to indicate to administrators that the permission element is used to authorize people to read data from a study known as “Big Study”.
* Object Id  
  This must exactly match the qualified class name of the class whose instances the protection element will be used to filter.
* Attribute Name  
  The name of the attribute of the named class that this protection element will be used to match.
* Attribute Value  
  The value of the named attribute that this protection element will match. The exact nature of the comparison (lexical or numeric) is determined by the type of the attribute.
* Type  
  This field is not used.
* Last Update  
  A read-only field that contains the data that the details of the protection element were last updated.
* Description  
  A human readable description of the protection element.

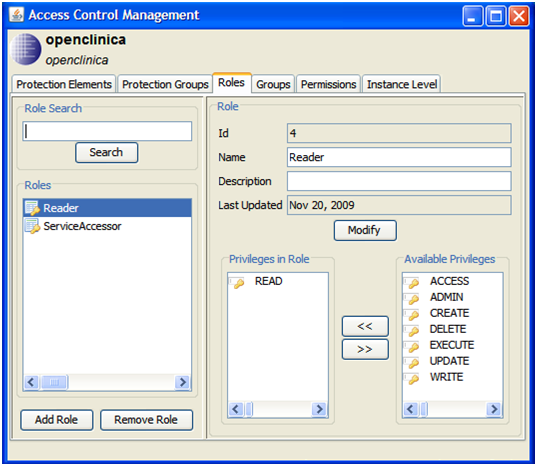
Protection elements are not directly associated with groups or permissions. This association is made indirectly through protection groups.

A protection group can be associated with multiple groups. Each protection group has a name that administrators use to identify the protection group. In this example, there is a protection group named “Big Study Group”. This group contains the protection element named “Big Study”.

Here is the screen used to administer protection groups:



Permission groups are indirectly associated with a combination of privileges and users. The association between protection groups and privileges is indirect. Permission groups are associated with roles which are associated with privileges. For this example we define a role named “Reader” that has the “READ” privilege.[[4]](#footnote-4) The screen used to administer this is shown below:



Combinations of individual protection groups and individual roles are usually associated with users through groups. Groups are most easily managed by telling the caGrid CSM service to synchronize with group definitions maintained on a gridGrouper service. The definition of such a service looks like this:



Shown in the preceding screen shot is the definition of a group named “Big Study User Group”. We want members of the “Big Study User Group” group to be able to read the object that describes the “Big Study” study and also to be able to read subject objects that are associated with subjects participating in the “Big Study” study.

The “Big Study” study object is a remote group, as indicated on the right side of the screen. Remote groups are groups that are managed by a gridGrouper service.

The upper part of right side of the screen shows the URL of the gridGrouper service that manages the group. It also shows the name that the gridGrouper service uses to identify the group. The lower part of the right side of the screen shows a list of the group’s members.

The last piece of the configuration puzzle is associating a group with pairings of a protection group and a role. This is shown in the following screen shot:



The preceding screen shot shows that the “Big Study User Group” (user) group is associated with the “Big Study Group” protection group and the “Reader” Role.

This completes the CSM configuration part of our example.

# OpenClinica Data

Before we look at an example of what CQL\_CSM does when it receives a CQL query, it is helpful to look at a small portion of the data in the OpenClinica database so that we will have an idea of what the results of CQL queries will be. We will look at a few relevant columns and rows of three tables.

Here is some of what is in the STUDY table that corresponds to the study class.  


The preceding listing shows three studies in the table. One of the studies is the “Big Study” study that we have been discussing.

Here is some of what is in the SUBJECT table that corresponds to the subject class.  


Here is some of what is in the STUDY\_SUBJECT table that corresponds to the association between the study class and the subject class.  


The information in the STUDY\_SUBJECT table indicates that the subjects Papa Bear, Mama Bear, Baby Bear and Mother Hubbard are subjects in the “Big Study” study. Mother Hubbard is also a subject in the “Fancy Study” study.

# Processing CQL Queries

In this section we discuss some CQL queries. We will assume that they are all submitted by the user we saw in the preceding example of CSM configuration: /O=CVRG/OU=LOA1/OU=Dorian/CN=bogus2

The first CQL query we will consider is

<CQLQuery xmlns="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Target name="org.cvrg.domain.Study"></Target>

<QueryModifier countOnly="true"></QueryModifier>  
</CQLQuery>

This query asks how many studies there are in the OpenClinica database. Even though we know that there are at least three studies in the database, the expected answer is one, because there is only one study that the user submitting the query is allowed to see.

Here is the SQL that CQL\_CSM generates for this CQL query:

select count(distinct this\_.study\_id) as y0\_   
from study this\_   
where ((this\_.study\_id  
 in (select F1.study\_id  
 from study F1   
 where F1.study\_id   
 in (select pe.attribute\_value  
 from csm\_protection\_group pg, csm\_protection\_element pe, csm\_pg\_pe pgpe,  
 csm\_user\_group\_role\_pg ugrpg, csm\_user u, csm\_role\_privilege rp,  
 csm\_role r, csm\_privilege p  
 where ugrpg.role\_id = r.role\_id  
 and ugrpg.user\_id = u.user\_id  
 and ugrpg.protection\_group\_id  
 = ANY (select pg1.protection\_group\_id  
 from csm\_protection\_group pg1  
 where pg1.protection\_group\_id = pg.protection\_group\_id  
 or pg1.protection\_group\_id  
 = (select pg2.parent\_protection\_group\_id  
 from csm\_protection\_group pg2  
 where pg2.protection\_group\_id  
 = pg.protection\_group\_id))  
 and pg.protection\_group\_id = pgpe.protection\_group\_id  
 and pgpe.protection\_element\_id = pe.protection\_element\_id  
 and r.role\_id = rp.role\_id  
 and rp.privilege\_id = p.privilege\_id  
 and pe.object\_id= 'org.cvrg.domain.Study'  
 and pe.attribute='id'  
 and p.privilege\_name='READ'  
 and u.login\_name='/O=CVRG/OU=LOA1/OU=Dorian/CN=bogus2'  
 and pe.application\_id=2))  
 or this\_.study\_id  
 in (select F2.study\_id  
 from study F2  
 where F2.study\_id  
 in (SELECT Distinct pe.attribute\_value  
 FROM CSM\_PROTECTION\_GROUP pg, CSM\_PROTECTION\_ELEMENT pe, CSM\_PG\_PE pgpe,  
 CSM\_USER\_GROUP\_ROLE\_PG ugrpg, CSM\_GROUP g, CSM\_ROLE\_PRIVILEGE rp,  
 CSM\_ROLE r, CSM\_PRIVILEGE p  
 WHERE ugrpg.role\_id = r.role\_id  
 AND ugrpg.group\_id = g.group\_id  
 AND ugrpg.protection\_group\_id  
 = ANY (select pg1.protection\_group\_id  
 from csm\_protection\_group pg1  
 where pg1.protection\_group\_id = pg.protection\_group\_id  
 OR pg1.protection\_group\_id  
 = (select pg2.parent\_protection\_group\_id  
 from csm\_protection\_group pg2  
 where pg2.protection\_group\_id  
 = pg.protection\_group\_id))  
 AND pg.protection\_group\_id = pgpe.protection\_group\_id  
 AND pgpe.protection\_element\_id = pe.protection\_element\_id  
 AND r.role\_id = rp.role\_id  
 AND rp.privilege\_id = p.privilege\_id  
 AND pe.object\_id= 'org.cvrg.domain.Study'  
 AND p.privilege\_name='READ'  
 AND g.group\_name IN ('Big Study User Group' )  
 AND pe.application\_id=2))))

Since the CQL contained no constraints, the WHERE clause portion of this select consists mostly of the group authorization and user authorization expressions specified in the CSM configuration. All that has been added is an OR and some parentheses.

The result returned from the query is

<CQLQueryResults targetClassname="org.cvrg.domain.Study"  
 xsi:type="ns1:CQLQueryResults"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:ns1="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLResultSet">  
 <ns1:CountResult count="1" xsi:type="ns1:CQLCountResult"/>  
</CQLQueryResults>

We now consider a CQL query to retrieve subjects that have the name Mama Bear or Papa Bear. This is the CQL query:

<CQLQuery xmlns="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
 <Target name="org.cvrg.domain.Subject">  
 <Group logicRelation="OR">  
 <Attribute name="uniqueIdentifier" value="Mama Bear"/>

<Attribute name="uniqueIdentifier" value="Papa Bear"/>

</Group>

</Target>  
 <QueryModifier>

<AttributeNames>uniqueIdentifier</AttributeNames>  
 </QueryModifier>  
</CQLQuery>

This query includes its own constraints that will be appear as part of the generated SQL WHERE clause along with the more complex expressions for filtering subjects through their association with studies. Here is the resulting SQL:

select this\_.unique\_identifier as y0\_ from subject this\_  
where ((this\_.unique\_identifier=Mama Bear or this\_.unique\_identifier=Papa Bear)  
 and (this\_.subject\_id  
 in (select F1.subject\_id  
 from subject F1 inner join study\_subject studysubje1\_  
 on F1.subject\_id=studysubje1\_.subject\_id  
 inner join study study2\_  
 on studysubje1\_.study\_id=study2\_.study\_id  
 where study2\_.study\_id  
 in (select pe.attribute\_value  
 from csm\_protection\_group pg, csm\_protection\_element pe, csm\_pg\_pe pgpe,  
 csm\_user\_group\_role\_pg ugrpg, csm\_user u, csm\_role\_privilege rp,  
 csm\_role r, csm\_privilege p  
 where ugrpg.role\_id = r.role\_id  
 and ugrpg.user\_id = u.user\_id  
 and ugrpg.protection\_group\_id  
 = ANY (select pg1.protection\_group\_id  
 from csm\_protection\_group pg1  
 where pg1.protection\_group\_id = pg.protection\_group\_id  
 or pg1.protection\_group\_id  
 = (select pg2.parent\_protection\_group\_id  
 from csm\_protection\_group pg2  
 where pg2.protection\_group\_id  
 = pg.protection\_group\_id))  
 and pg.protection\_group\_id = pgpe.protection\_group\_id  
 and pgpe.protection\_element\_id = pe.protection\_element\_id  
 and r.role\_id = rp.role\_id  
 and rp.privilege\_id = p.privilege\_id  
 and pe.object\_id= 'org.cvrg.domain.Study'  
 and pe.attribute='id'  
 and p.privilege\_name='READ'  
 and u.login\_name='/O=CVRG/OU=LOA1/OU=Dorian/CN=bogus2'  
 and pe.application\_id=2))  
 or this\_.subject\_id  
 in (select F2.subject\_id  
 from subject F2 inner join study\_subject studysubje1\_  
 on F2.subject\_id=studysubje1\_.subject\_id  
 inner join study study2\_ on studysubje1\_.study\_id=study2\_.study\_id  
 where study2\_.study\_id  
 in (SELECT Distinct pe.attribute\_value  
 FROM CSM\_PROTECTION\_GROUP pg, CSM\_PROTECTION\_ELEMENT pe, CSM\_PG\_PE pgpe,  
 CSM\_USER\_GROUP\_ROLE\_PG ugrpg, CSM\_GROUP g, CSM\_ROLE\_PRIVILEGE rp,  
 CSM\_ROLE r, CSM\_PRIVILEGE p  
 WHERE ugrpg.role\_id = r.role\_id  
 AND ugrpg.group\_id = g.group\_id  
 AND ugrpg.protection\_group\_id  
 = ANY (select pg1.protection\_group\_id  
 from csm\_protection\_group pg1  
 where pg1.protection\_group\_id = pg.protection\_group\_id  
 OR pg1.protection\_group\_id  
 = (select pg2.parent\_protection\_group\_id  
 from csm\_protection\_group pg2  
 where pg2.protection\_group\_id  
 = pg.protection\_group\_id))  
 AND pg.protection\_group\_id = pgpe.protection\_group\_id  
 AND pgpe.protection\_element\_id = pe.protection\_element\_id  
 AND r.role\_id = rp.role\_id  
 AND rp.privilege\_id = p.privilege\_id  
 AND pe.object\_id= 'org.cvrg.domain.Study'  
 AND p.privilege\_name='READ'  
 AND g.group\_name IN ('Big Study User Group' )  
 AND pe.application\_id=2))))

Finally here is the result returned from the CQL query:

<CQLQueryResults targetClassname="org.cvrg.domain.Subject"  
 xsi:type="ns1:CQLQueryResults"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:ns1="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLResultSet">

<ns1:AttributeResult xsi:type="ns1:CQLAttributeResult">  
 <ns1:Attribute name="uniqueIdentifier" value="Papa Bear"  
 xsi:type="ns1:TargetAttribute"/>  
 </ns1:AttributeResult>  
 <ns1:AttributeResult xsi:type="ns1:CQLAttributeResult">

<ns1:Attribute name="uniqueIdentifier" value="Mama Bear"  
 xsi:type="ns1:TargetAttribute"/>  
 </ns1:AttributeResult>

</CQLQueryResults>

1. The list of CMS services that appears in the **Manage Access Control** window is manually configured during installation. If you want to work with a CSM service that is not on the list, you can edit the list of CSM services by [**modifying the CMS preferences**](https://www.cagrid.org/display/csm131/Installation#Installation-AddingaCSMServiceURL) for the GAARDS UI. [↑](#footnote-ref-1)
2. he credential list always contains "Globus Default Proxy", which is a convenient default. If you are logged into the grid with more than one identity, "Globus Default Proxy" might not be the identity that you think it is. If you are logged into the grid with more than one identity, selecting the actual identity rather than "Globus Default Proxy" can avoid mysterious problems later on. [↑](#footnote-ref-2)
3. The SQL is generated only when the domain model was loaded from .jar files. When the domain model is loaded from .jar files, it is assumed that the data service that will consume the authorization policy information serves information from a caCORE application and uses the replacement CQL processor, which needs the SQL.

   When the domain model is loaded from a domain model file, the CSM service has no information about a database that it can use as the basis for any SQL. This is not a problem, since the assumption is that when the domain model is loaded from a domain model file, the authorization policy information will be consumed by the CQL pre-processor, which does not need any SQL. [↑](#footnote-ref-3)
4. If you don’t see READ as one of the available privileges, then it was not added by the installation process. There is another windows available to add privileges.

   Click on the **Access Control** menu. Click on the **Manage Privileges** menu item. The Manage Privileges window appears. You can click on the **Search** button to see a list of the available privileges. Click on the **Add** button to add a privilege. The Create Privilege dialog appears. Enter READ in the **Privilege Name** field. Click on the **Create** button.

   You should now see the READ privilege in the list of privileges. Close the **Manage Privileges** window and you are done. [↑](#footnote-ref-4)