OGSA-DAI WSRF Tutorial

NGS Induction Event, Guy Warner, NeSC Training Team

This OGSA-DAI tutorial has two aims:

- to show how access to multiple databases simultaneously is aided by the use of OGSA-DAI.
- to show how to use the OGSA-DAI client toolkit to access a data source and perform multiple actions on the data before retrieving the results.

Please note that all images in this page can be expanded by clicking on them, they are linked to the full size versions.

Accessing Multiple Databases

Initially a database that is running on the host "tc03" is accessed. This database contains an example address book with fictional names and addresses. A copy of the WSRF version of OGSA-DAI (2.1) has been installed and configured to access this database. OGSA-DAI in turn is using the Globus container (the Globus equivalent of Apache Tomcat) as the means of exposing the service on to the internet. The database is running under PostgreSQL.

A second database will then be accessed at the same time as the first database. This database is running on the host "tc05". The database this time has a different schema to the database on "tc03". This database contains a table called stafflist, an example list of staff with (again somewhat) fictional entries. In fact there are several other databases hosted on these servers. These databases are used for providing space for temporary tables and for the exercise at the end of this tutorial.

The first aim of this tutorial is to perform a comparison of both of these main two databases and performing a join of the date .The tutorial then moves on to a more complex scenario where each attendee has to search through three databases for their individual information. This will become clear at that part of the tutorial.

1. Because the tool used in this part of the tutorial often ends up consuming large amounts of memory it is neccessary to spread the load across three nodes of the NeSC training team cluster. You should log on using secure shell (same name/password as usual). to a node according to the below table:
    i. tc03.nesc.ed.ac.uk : user01 - user08
    ii. tc05.nesc.ed.ac.uk : user09 - user15

2. The first part of this tutorial uses the databrowser, a simple tool provided with OGSA-DAI for querying OGSA-DAI services and displaying the results. Real life usage of OGSA-DAI is more likely to involve developing custom code to do the same tasks but then do something more interesting with the data (compared to just displaying a table of results).

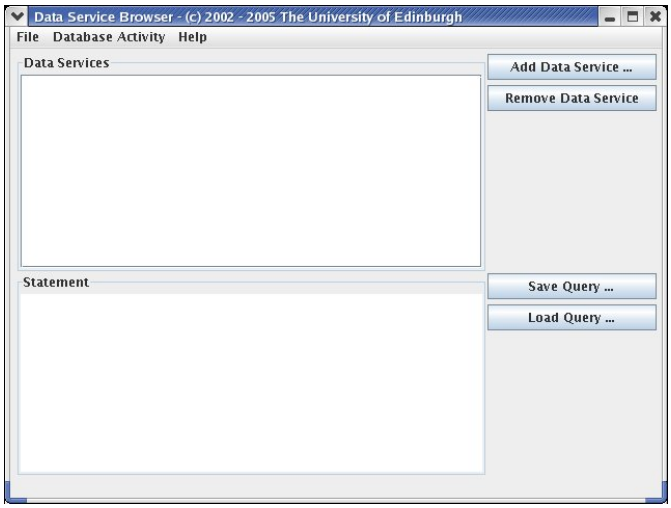
    The environment in which the databrowser will run first needs to be configured. It is neccessary to set the CLASSPATH to point to all of the relevant OGSA-DAI and Globus jar files (java libraries). If you are not familiar with java, the CLASSPATH contains a list of the locations of all the jar files and classes your program depends on. A couple of other standard environmental variables also need to be set. You can configure the correct environment with the command

```
module load java ogsadai_wsrf
```

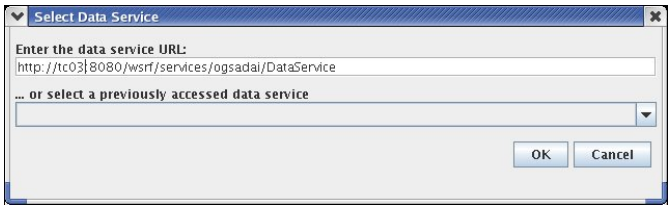
Finally start the databrowser with the command

```
databrowser
```

3. You should now have an empty databrowser window:



The first task is therefore to configure it to point to your local database. Since a single OGSA-DAI data service may provide access to multiple databases the first part of OGSA-DAI that must be accessed is it's list of these databases or more acurately "resources" (since it does not have to be a database, for example it could be an XML file). Click on "Add Data Service...".

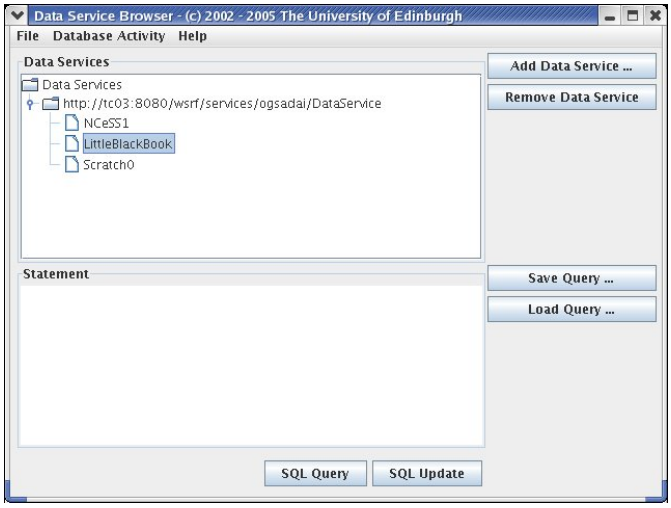


In the dialog that appears enter the url

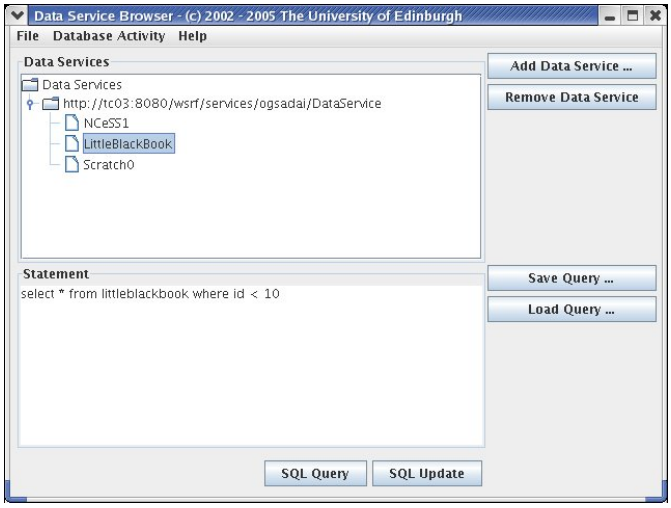
```
http://tc03:8080/wsrf/services/ogsadai/DataService
```

Click "OK" to continue.

4. The data service and its available resources should now appear in the databrowser window.



The next task is to run a query on this database. Select the database you wish to run the query on, in this case "LittleBlackBook". Now enter the query shown in the below screenshot and click "SQL Query".



If all goes well you should have a dialog appear containing a table of the results as shown below:

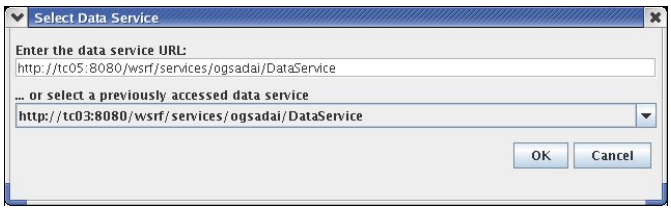
id	name	address	phone
1	Ally Antonioletti	826 Hume Crescent, S...	01670061244
2	Amy Atkinson	583 Atkinson Drive, So...	06312054624
3	Andrew Borley	354 Jackson Road, Edi...	01057075166
4	Charaka Chue Hong	750 Pearson Crescent,...	09945916393
5	Dave Hardman	079 Borley Gardens, ...	06725558505
6	George Hicken	398 Magowan Street, ...	09066873297
7	James Hume	801 Laws Gardens, Ed...	00246566355
8	Malcolm Jackson	743 Krause Lane, Edin...	04333719273
9	Mario Krause	026 Atkinson Gardens...	01922093483

When you close this dialog a message about data being lost will appear. Just click "OK", since this data does not need saving.

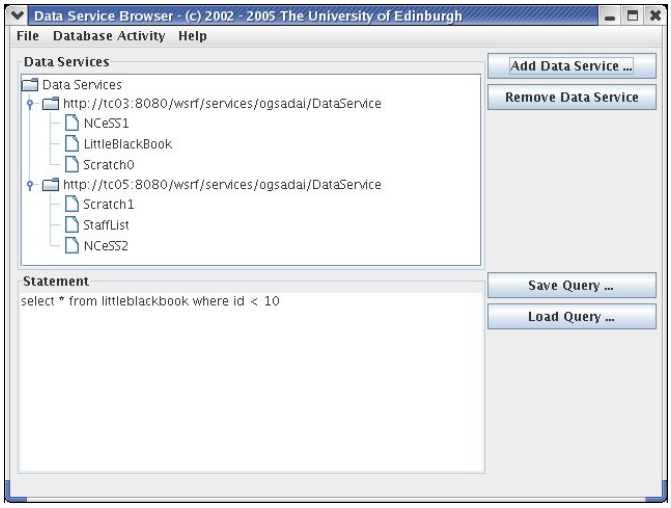
5. Now its time to access the second database. This time add the data service

   `http://tc05:8080/wsrf/services/ogsadai/DataService`

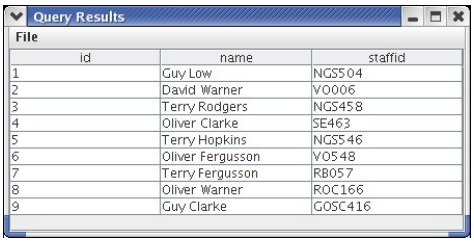
   as shown below:



Your databrowser window should now list two data services and six resources (plus the previous sql query):

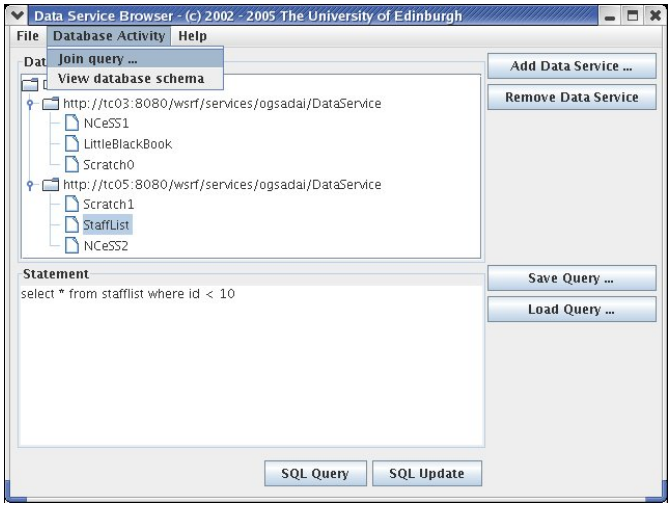


6. Test your access to the StaffList database by modifying your previous query to this time query the table stafflist. Don't forget to select the correct database. On success you should see the below results:

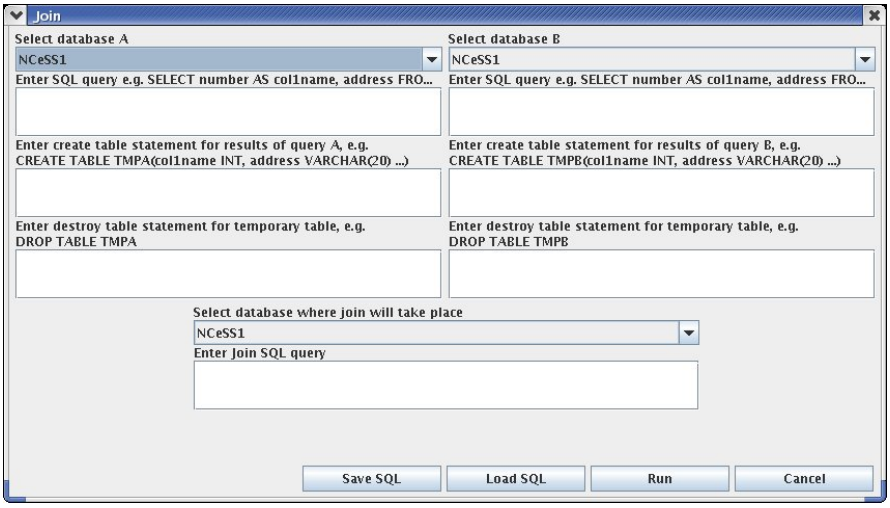


id	name	staffid
1	Guy Low	NGS504
2	David Warner	VO006
3	Terry Rodgers	NGS458
4	Oliver Clarke	SE463
5	Terry Hopkins	NGS546
6	Oliver Fergusson	VO548
7	Terry Fergusson	RB057
8	Oliver Warner	ROC166
9	Guy Clarke	GOSC416

7. The next stage of this tutorial is to 'join' the data of these two tables. From the menu at the top of the databrowser window select "Database Activity" and then "Join Activity ...", as shown below:



The Join dialog now appears. Notice how all of the resource selections point to the first listed resource. If you wish to try saving and loading SQL queries be aware that your resource selection is not saved.

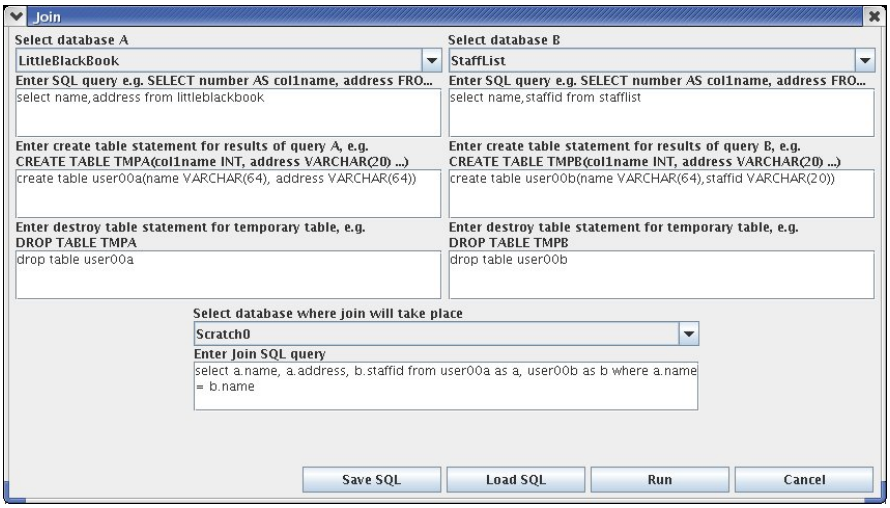


8. To join the data it is necessary for all of the data to be present in the same database. This dialog guides you through the process of selecting data of interest from both databases, copying this to temporary tables, performing the join and then finally cleaning up the temporary tables. When doing so a couple of points should be considered. The first of these is that the better the initial selection of data of the databases, the less data needs to be streamed to the temporary table. The other point is remember not to use the name of a table that already exists, hence the reason OGSA-DAI cannot automatically do this step for you. Using the name of an existing table will create an error.

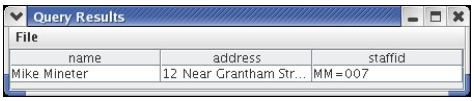
   To avoid creating a table name that conflicts with other trainees and to share the usage across the OGSA-DAI servers please follow the below two rules when creating your temporary tables:

   i. For users user01-user10 it should be contained within the Scratch0 database whilst for users user11-user20 it should be Scratch1 and users user21-user30 it should be Scratch2.
   ii. Use a temporary table name of the format "<username>tmp<some letter(s)>" where <username> is your local username.

   In this case the task is to find the name that appears in both databases and display that persons name, address and staff identity number. Enter the sql queries as shown below and ensure the correct database has been selected at each point. ***Remember to change the table name and scratch database from those show in the below screenshot.***



   If you are successful the below dialog should appear.



When you are finished close down the browser. You will be prompted with an "Exit" confirmation dialog. Click "OK".

Forward to the Client Toolkit

OGSA-DAI WSRF Tutorial: The Client Toolkit

NGS Induction Event,
Guy Warner, NeSC Training Team

This second OGSA-DAI WSRF tutorial shows how to use the OGSA-DAI client toolkit to access a data source and perform multiple actions on the data before retrieving the results. These all involve accessing the LittleBlackBook database via tc03.nesc.ed.ac.uk that was used in the previous tutorial.

This part of the tutorial involves some editing of files. To edit a file with a graphical editor use the command

```
kwrite <filename> &
```

where <filename> is the name of the file you wish to edit (though you may also use the "vi" editor if you would prefer to).

The commands in this tutorial should be run on the same node of the NeSC Training cluster as you have used in the previous tutorial

1. The first step is to just access the LittleBlackBook database and display several entries. If this is a new terminal session from the previous tutorial you will need to run the command

   ```
   module load java ogsadai_wsrf
   ```

2. The java code for running a simple sql query is already in your account. A helper class called "Utilities" that provides some formatting functions, has also been provided (and already compiled). You can view the code with the command

   ```
   cat ogsadai_client/SimpleSQLQueryClient.java
   ```

   Compile this code by typing

   ```
   javac ogsadai_client/SimpleSQLQueryClient.java
   ```

   and then run the program with the command

   ```
   java ogsadai_client.SimpleSQLQueryClient
   ```

   You should get the below output

   ```
   Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
   LittleBlackBook
   Performing SQL query select * from littleblackbook where id<10
   Response:

     1 Ally Antonioletti 826 Hume Crescent, Southampton 01670061244
     2 Amy Atkinson 583 Atkinson Drive, Southampton 06312054624
     3 Andrew Borley 354 Jackson Road, Edinburgh 01057075166
     4 Charaka Chue Hong 750 Pearson Crescent, Southampton 09945916393
     5 Dave Hardman 079 Borley Gardens, Winchester 06725558505
     6 George Hicken 398 Magowan Street, Winchester 09066873297
     7 James Hume 801 Laws Gardens, Edinburgh 00246566355
     8 Malcolm Jackson 743 Krause Lane, Edinburgh 04333719273
     9 Mario Krause 026 Atkinson Gardens, Winchester 01922093483
   ```

3. The previous code displayed the results already formatted. These results were transferred to your code in the form of a "Response Document". This document describes the status of execution of the task you requested of OGSA-DAI and may also contain result data, as was the case in the previous code. When this document contains result data it also contains information describing the format of the data.
   Modify the code used in the previous section to display the received Response document. This can be done by

inserting the line

System.out.println(response.getAsString());

between the lines

System.out.println("Response: ");

and

ResultSet rs = rowset.getResultSet();

Recompile the code and run it again. You should get a result similar to the below output (note that due to the length of the output, parts have been cut out).

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Performing SQL query select * from littleblackbook where id<10
Response:
<?xml version="1.0" encoding="UTF-8"?>
<ns1:response xmlns:ns1="http://ogsadai.org.uk/namespaces/2005/10/types">
    <ns1:session id="session-ogsadai-1098d2e1864"/>
    <ns1:request status="COMPLETED"/>
    <ns1:result name="SQLQuery-ogsadai-109a2b3a400" status="COMPLETED"/>
    <ns1:result name="WebRowSet-ogsadai-109a2b3a402" status="COMPLETED"/>
    <ns1:result name="ogsadai-109a2b3a403" status="COMPLETED"><![CDATA[<webRowSet
xmlns="http://java.sun.com/xml/ns/jdbc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/jdbc http://java.sun.com/xml/ns/jdbc/
webrowset.xsd">
<properties>
<command></command>
<concurrency>1007</concurrency>
...
<columnValue>1</columnValue>
<columnValue>Ally Antonioletti</columnValue>
<columnValue>826 Hume Crescent, Southampton</columnValue>
<columnValue>01670061244</columnValue>
</currentRow><currentRow>
<columnValue>2</columnValue>
<columnValue>Amy Atkinson</columnValue>
<columnValue>583 Atkinson Drive, Southampton</columnValue>
<columnValue>06312054624</columnValue>
</currentRow><currentRow>
...
```

4. The next section of the tutorial looks at more detail into the tasks that OGSA-DAI can achieve. The terminology used in OGSA-DAI for a task is "Activity". In the previous examples a simple activity, in this case an SQLQuery using a WebRowSet to access the data, was constructed, sent to the OGSA-DAI server via a "perform document" (not looked at in this tutorial) and the results received as a response document. Before continuing examine the code to see how these activities were combined and performed.

   An activity in OGSA-DAI can do more than one task at the server end before returning the response to the client. Thus it is possible to construct a workflow of activities acting on the data. The code used this time will perform an sql query, compress the results and finally transfer the compressed results to an ftp site. ***You must modify the code so that the ftp site used is the cluster node you are using.*** This is to avoid problems with security when transferring to a locally running anonymous server.

   After making the edit, compile and run the file ogsadai_client/MultipleActivityClient.java. You should get output similar to the following:

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Performing SQL query select * from littleblackbook where id='1'
Response:
<?xml version="1.0" encoding="UTF-8"?>
<ns1:response xmlns:ns1="http://ogsadai.org.uk/namespaces/2005/10/types">
    <ns1:session id="session-ogsadai-1098d2e1866"/>
    <ns1:request status="COMPLETED"/>
    <ns1:result name="SQLQuery-ogsadai-109a2dc7471" status="COMPLETED"/>
    <ns1:result name="WebRowSet-ogsadai-109a2dc7473" status="COMPLETED"/>
    <ns1:result name="GZIPCompression-ogsadai-109a2dc7475" status="COMPLETED"/>
    <ns1:result name="DeliverToURL-ogsadai-109a2dc7478" status="COMPLETED"/>
    <ns1:result name="ogsadai-109a2dc7477" status="COMPLETED"><![CDATA
[<gzipCompressionMetadata>
  <checksum type="adler32" value="3342833185"/>
</gzipCompressionMetadata>
]]></ns1:result>
</ns1:response>
```

5. To see the output, first run the command

   `mv /var/ftp/incoming/<username>.gz ~/`

   (If you don't move the file, the next time you run the code you will get a failure message because the anonymous ftp server default is set up to prevent over-writing of files).

   You can now view the contents of the response using the command

   `zcat ~/<username>.gz`

6. Moving towards the end of the tutorial, the next exercise demonstrates how a series of activities on one OGSA-DAI exposed resource can exchange data with another resource. These resources may or may not be running on the same OGSA-DAI server. This time the code will use an output datastream to push a series of entries from the LittleBlackBook table to a temporary table in one of the scratch databases. The actvity "SQLBulkLoad" is used for puting the data into the temporary table since this is much more efficient than handling each entry individually. It should be noted that the construction of an output datastream requires the explicit use of sessions which are not covered in this tutorial. To keep the code simple much of the functionality from the previous examples has been incorporated into the class "DataSource" which is already compiled for you.

   Before you can run ***the code it must be modified so that it talks to the scratch resource you used in the previous tutorial and change the name of the temporary table***. This involves modifying the hostname, the resource name and the table name. The name of this temporary table should be (again as in the previous tutorial) in the format "<username>tmp<some letter(s)>".

   Compile the java file "ogsadai_client/BulkLoad.java" and run the class to get output similar to the below output:

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Connecting to DataService for temporary storage
Create table user00
Perform bulk-load from LittleBlackBook to temporary table
Performing SQL query: select * from user00

 Ally Antonioletti       826 Hume Crescent, Southampton
 Amy Atkinson    583 Atkinson Drive, Southampton
 Andrew Borley   354 Jackson Road, Edinburgh
 Charaka Chue Hong       750 Pearson Crescent, Southampton
 Dave Hardman    079 Borley Gardens, Winchester
 George Hicken   398 Magowan Street, Winchester
 James Hume      801 Laws Gardens, Edinburgh
 Malcolm Jackson         743 Krause Lane, Edinburgh
 Mario Krause    026 Atkinson Gardens, Winchester

Drop temporary table
```

7. If there is time left at the end of the tutorial modify the java file "ogsadai/JoinClient.java" in the same way as in the

previous step. Run this class to perform the same join as was performed with the databrowser in step 9 of the previous tutorial.