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.

### Accessing Multiple Databases

Initially a database that is running on your local machine is accessed. In the screenshots that follow the local machines address has been obscured and replaced with `<local>`. The local address of your classroom machine will appear instead of `<local>`. This database is running on your local machine using the mysql database server. This database is available for read and write operations and contains a table called `littleblackbook`, an example address book with (somewhat) fictional entries. A local copy of OGSA-DAI has been installed and configured to access this database. OGSA-DAI in turn is using Apache Tomcat as the means of exposing the service on to the internet.

A second database will then be accessed at the same time as the first database. This database is running on a separate server on the network using the postgreSQL database server. The database this time has a different schema to your local database. This database is only available for read operations and contains a table called `stafflist`, a list of (again somewhat) fictional staff and their staff identity numbers. In the screenshots below, the address of this remote machine has also been obscured and replaced with `<remote>`.

The first aim of the tutorial is to perform a comparison of both of these databases. The tutorial will then finish by looking at accessing an ogsadai service directly from java in a way that requires no knowledge of java programming.

Note: All of the images below have been reduced in size so as to make this tutorial easier to read. To see a full size image click on the image. The images were all taken on an earlier version of OGSA-DAI as a result the url's referred to have slightly changed. Please use the url's from the text and not the images. Any text that must be typed appears like this.

1. To start the tutorial you will need to startup Tomcat on your local machine. This is done by double clicking the “Start Tomcat” shortcut on your desktop. Next launch the OGSA-DAI databrowser by double clicking the “Databrowser” shortcut on your desktop. The databrowser is 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). A simple example appears at the end of the tutorial.

2. 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 instance of OGSA-DAI may provide access to multiple databases (not done in this tutorial for simplicities sake), the first part of OGSA-DAI that must be accessed is its Registry of these services. Click on "Add Registry...".

In the dialog that appears enter the url

```
http://localhost:8080/ogsas/services/ogsadai/DAISGR
```

(or just select it from the menu that appears in the dialog). Click "OK" to continue.

3. The Registry should now appear in the databrowser window. A list of the available databases should also have appeared.

The next task is to run a query on this database. Select the database you wish to run the query on (in this case there is only one available, however it must still be explicitly selected). Now enter the query shown in the below screenshot.
If all goes well you should have a dialog appear containing a table of the results as shown below:

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

4. Now its time to access the second database. This time add the registry

   http://pub-234:8080/ogsa/services/ogsadai/DAISGR

as shown below:

Your databrowser window should now list two registries and two databases (plus the previous sql query):
5. Test your access to the remote 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:

6. The final stage to 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 database selections point to the first selected database. If you wish to try saving and loading SQL queries be aware that your database selection is not saved.
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 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. 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.

7. Try to improve the join so that it runs quicker by using the id number field (of type INT(11)).

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

9. Move on to the second half of the tutorial on the next page.
The Client Toolkit

The second half of this 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. In the first and second examples this is done by accessing the OGSA-DAI service running on the NGS at Manchester and using a table called littleblackbook. In the final case an OGSA-DAI server is running on pub-234 (referred to in the code as localhost). This server is using a table called stafflist. This second OGSA-DAI server is used so as to allow for different security considerations (more details about this are in that section of the tutorial).

This part of the tutorial involves some editing of files. To use a graphical editor you must first launch Exceed - part of Hummingbird Connectivity. After the splash screen no further screen will appear - this is what is meant to happen. To edit a file use the command

```
 kwrite <filename> &
```

where `<filename>` is the name of the file you wish to edit. You can safely ignore any messages that appear in your terminal window. When you exit you may also see a message about "This may end your X session" - just click "Ok".

1. The first step is to just access an ogsadai database. The database being used is hosted on the NGS at Manchester and contains the littleblackbook table. The connection requires GSI authentication and hence the java code is run under Linux. Open a Putty (ssh) connection to pub-234, log in and ensure you have a running proxy (see [here](#) for help).

2. The environment in which your program will run now needs to be configured. The default environment in your account on pub-234 is set up for the Globus Toolkit 2.4. OGSA-DAI 4.0 however uses Globus Toolkit 3. The environmental variables that store this information therefore need to be modified for this session. It is also necessary to set the CLASSPATH to point to all of the relevant ogsadai 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. A script to do this task has already been created, just type the command

```
 source /usr/local/ogsadai/etc/ogsadai_env.sh
```

You can test your environment with the commands

```
 echo $GLOBUS_LOCATION
 echo $CLASSPATH
```

The following table gives an example of the output from all the commands in this section (the CLASSPATH has had to be truncated for reasons you will understand when you have run the above commands)
3. 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/SimpleSQLGSIQueryClient.java

Compile this code by typing

     javac ogsadai_client/SimpleSQLGSIQueryClient.java

and then run the program with the command

     java ogsadai_client.SimpleSQLGSIQueryClient

You should get the below output

        Ready to connect to service at: http://grid-data-node10.man.ac.uk:8080/ogsa/services/ogsadai/GridDataServiceFactory
        Created Grid Data Service at: http://grid-data-node10.man.ac.uk:8080/ogsa/services/ogsadai/GridDataServiceFactory/hash-12539221-1118421055150

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

Destroyed service.

4. 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(Utilities.responseToString(response));

between the lines
Response response = service.perform(query);

and

ResultSet rs = query.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).

```
Ready to connect to service at:
Created Grid Data Service at:
http://grid-data-node10.man.ac.uk:8080/ogsa/services/ogsadai/GridDataServiceFactory/hash-
30866355-1120148932561

Performing SQL query: select * from littleblackbook where id<10

Response:
---------
<?xml version="1.0" encoding="UTF-8"?>
<gridDataServiceResponse xmlns="http://ogsadai.org.uk/namespaces/2003/07/gds/types">
  <request status="COMPLETED"/>
  <result name="SQLQuery-ogsadai-104ce178420" status="COMPLETED"/>
  <result name="ogsadai-104ce178421" status="COMPLETED"><?xml version="1.0" encoding="UTF-
8"?>
ns/jdbc/webrowset.xsd">
      <properties>
        ....
      </properties>
      <currentRow>
        <columnValue>9</columnValue>
        <columnValue>Mario Krause</columnValue>
        <columnValue>026 Atkinson Gardens, Winchester</columnValue>
        <columnValue>01922093483</columnValue>
      </currentRow>
    </data>
  </result>
</gridDataServiceResponse>
```

5. The final 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 single activity, in this case an SQLQuery, 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. This occured in the below two lines

```
SQLQuery query = new SQLQuery(sql);
Response response = service.perform(query);
```

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. This time a locally running OGSA-DAI server is used. This is to avoid problems with security when transferring to a locally running anonymous server.

This time compile and run the file ogsadai_client/MultipleActivityClient.java. You should get output similar to the following:
Ready to connect to service at:
http://localhost:8080/ogsa/services/ogsadai/PostgresGDSF
Created Grid Data Service at:

Performing request

Response:
----------
<?xml version="1.0" encoding="UTF-8"?>
<gridDataServiceResponse xmlns="http://ogsadai.org.uk/namespaces/2003/07/gds/types">
  <request status="COMPLETED"/>
  <result name="SQLQuery-ogsadai-104d2732388" status="COMPLETED"/>
  <result name="GZIPCompression-ogsadai-104d273238a" status="COMPLETED"/>
  <result name="DeliverToURL-ogsadai-104d273238d" status="COMPLETED"/>
  <result name="ogsadai-104d273238c" status="COMPLETED"><![$gzipCompressionMetadata>
    <checksum type="adler32" value="2010661078"/>
  </gzipCompressionMetadata>
</result>
</gridDataServiceResponse>

Destroyed service.

6. To see the output, first run the command (where <username> is your username on pub-234)

    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

7. If you have time at the end try editing the code so that the response is delivered uncompressed.