SOLUTION GUIDE

How a Simple Client/Server Application Reacts When an Oracle RAC Node Fails

By: Justin Richard Bleistein

This Solution Guide is a demonstration of application connections using one of the Oracle database MAA, Maximum Availability Architecture components – RAC. This series will simulate an IT infrastructure failure, i.e. node/server failures, and how an application reacts and is kept highly available with the Oracle database which is protected by RAC.

We will be demonstrating how a simple client/server application written in Java SE, Standard Edition, reacts when the Oracle database instance it is connected to fails in an Oracle RAC environment. RAC stands for Real Applications Cluster. It is Oracle's high availability solution for their database software. In previous releases it was called Oracle Parallel Server. The high level configuration of an Oracle RAC cluster is one database residing on SAN disk which is shared between two or more nodes. In this type of configuration there are multiple Oracle instances connecting to this one Oracle database. The concept behind this configuration is that there will always be at least one node/path available to the database via the instance running on it, so that application will always have somewhere to go when attempting to connect to the database.

I wrote this application using the Java 7 SDK, Software Development Kit, (1.7). I used Eclipse as my IDE, Integrated Development Environment running on my OpenSuSE Linux laptop. I used the JDBC, Java DataBase Connectivity, driver version 6 to connect to Oracle. I embedded the database connection information in the Java code itself. The GUIs of this application were written using the JAVA Swing API. That being the case, I will not be using TAF, FAN or round robin TNS connection configurations. For this blog, I wanted to make it as bare bones and simple as possible. In a real world production environment an enterprise mission critical application written in JAVA would be written using JAVA EE, Enterprise Edition, standards and would be running via an application server software, e.g. IBM Websphere Application server, TomEE(Tomcat), JBOSS, Weblogic, etc..

In our test environment for this blog we have two IBM AIX LPARs running on a POWER 8 server. AIX is IBM’s proprietary version of the UNIX operating system. The specs of the AIX LPARs are below:

- Aixdb1:
    - OS patch level: AIX 7.1 TL 3 SP 4.
    - Processor Entitlement: 0.2
    - Virtual processor: 1
    - 8 Gig of RAM
    - SAN Storage: IBM V7000
    - OS boots off of the SAN
    - IP: 192.168.240.195

- Aixdb2:
    - OS patch level: AIX 7.1 TL 3 SP 4.
    - Processor Entitlement: 0.2
    - Virtual processor: 1

    - 8 Gig of RAM
    - SAN Storage: IBM V7000
    - OS boots off of the SAN
    - IP: 192.168.240.197

We are running Oracle Grid Infrastructure and Oracle database RAC on these AIX LPARs. The database specifications are listed below:

- Database Name: db
- RAC Instance one running on node # 1 (Aixdb1): db1
- RAC Instance two running on node # 2 (Aixdb2): db2
- Oracle Grid Infrastructure software version: 12.1.0.1.0
- Oracle RDBMS software version with RAC option: 11.2.0.1.0
- Oracle Grid Infrastructure software home: /u01/app/12.1.0/grid/
- Oracle RBMS software home: /u05/app/oracle/product/11.2.0/dbhome_1
- RAC SCAN cluster name: db-cluster
- RAC SCAN IP addresses:
    - 192.168.240.179
    - 192.168.240.180
    - 192.168.240.181
- RAC VIP IP addresses:
    - Aixdb1: 192.168.240.60
    - Aixdb2: 192.168.240.83
- RAC private interconnect IP addresses:
    - Aixdb1: 10.1.1.1
    - Aixdb2: 10.1.1.2

The application is a simple two tier thick client. It is running on my Linux laptop which has network connectivity to the AIX database servers.

My JAVA application's file and directory structure consists of the executable oraconnect.jar and a directory named lib. The directory lib is where the JAVA libraries, i.e. jars, that my application needs reside. In this case the standard JAVA runtime library has everything I need except for the Oracle specific database interaction stuff. That is what the ojdbc6.jar file/library in the lib directory will be providing. My application will be referencing methods/functions in that library when it needs to talk to the Oracle database.

```
$ ls
lib  oraconnect.jar

$ ls -l lib
total 2676
-rw-r--r-- 1 justinb users 2739670 Jan 13 18:03 ojdbc6.jar
```

The main application jar file/executable is made up of the compiled JAVA code I wrote. JAVA classes are compiled JAVA executables. JAVA JAR files are also considered executable, they hold JAVA class files. This is one way of packaging and deploying a JAVA application. They can be seen below via the jar –tvf command. Note the jar command is available if you have the JDK installed:

```
$ jar -tvf oraconnect.jar
   102 Tue Jan 13 22:02:32 CET 2015 META-INF/MANIFEST.MF
  6173 Tue Jan 13 19:45:18 CET 2015 oraconnect/get_query_result1.class
  2271 Tue Jan 13 19:45:18 CET 2015 oraconnect/search.class
  5178 Tue Jan 13 21:06:36 CET 2015 oraconnect/create_contact.class
   674 Tue Jan 13 19:45:18 CET 2015 oraconnect/oraconnect.class
  2350 Tue Jan 13 19:45:18 CET 2015 oraconnect/main_menu.class
  1281 Tue Jan 13 22:02:18 CET 2015 oraconnect/db_connect.class
```

Files in JAR file:	Description:
MANIFEST.MF	This is a standard file which tells the JAVA JVM, Java Virtual Machine, what class the main method is stored in and other metadata about the application.
oraconnect.class	This JAVA class is where my program starts executing. This class contains the main method.
mainmenu1.class	This JAVA class is the first GUI form/screen you see.
db_connect.class	This JAVA class tells my program how to connect to the database.
create_contact_class	This JAVA class presents the GUI form/screen you see when you go to create a contact. It also inserts a database record.
search.class	This JAVA class presents the GUI form/screen you see when you go to search for a contact. It also queries the database.
get_query_results1	This JAVA class presents the GUI screen you see to present the results of your contact search. This is the formatted results of the database query/SELECT.

The db_connect.class code uses the following to connect to the Oracle database. I am using a standard Type-4 JDBC thin client with this application, very basic.

jdbc:oracle:thin:@db-cluster:1521/db.ats.local

In database terms when you create a contact my code inserts a record into the CONTACTS table of the APP schema.
When you search for a contact my code queries the database with a SELECT statement against the CONTACTS table of the APP schema.

Create contact:

INSERT INTO APP.CONTACTS VALUES (‘LNAME’,’FNAME’,’AGE’,’ADDRESS’,’CITY’,’ZIP’);

Search for a contact:

SELECT * FROM APP.CONTACTS;

From a database interaction standpoint this application is very simple. It requires a very simple schema, i.e. one table. It is only intended to demonstrate how applications which just do a simple connection to a database perform simple DMLs react to a node failure in a RAC environment.

The first thing we need to check is that our Oracle SCAN is setup correctly. SCAN stands for Single Client Access Name. It was introduced in Oracle 11g. It allows clients/applications to access the database via one DNS resolved network name and then let Oracle figure out which node in the cluster to send the application connection to. SCAN needs to be a DNS entry and needs to resolve to three IP addresses. It needs to be setup in a round robin configuration. To test your SCAN IP configuration, use the nslookup and/or ping command a few times. Each time you run nslookup or ping you should see the three IP addresses of the SCAN name rotating. The SCAN name is basically the network name that applications use to access the Oracle database cluster.

From my Linux PC, i.e. where the JAVA application will run from:

```
# nslookup db-cluster.ats.local
Server:         192.168.240.150
Address:        192.168.240.150#53

Name:   db-cluster.ats.local
Address: 192.168.240.180
Name:   db-cluster.ats.local
Address: 192.168.240.179
```

```
Name:   db-cluster.ats.local
Address: 192.168.240.181

# **nslookup db-cluster.ats.local**
Server:         192.168.240.150
Address:        192.168.240.150#53

Name:   db-cluster.ats.local
Address: 192.168.240.179
Name:   db-cluster.ats.local
Address: 192.168.240.181
Name:   db-cluster.ats.local
Address: 192.168.240.180

# nslookup db-cluster.ats.local
Server:         192.168.240.150
Address:        192.168.240.150#53

Name:   db-cluster.ats.local
Address: 192.168.240.181
Name:   db-cluster.ats.local
Address: 192.168.240.180
Name:   db-cluster.ats.local
Address: 192.168.240.179

oracle@justinbspc1:~> ping db-cluster
PING db-cluster.ats.local (192.168.240.179) 56(84) bytes of data.
64 bytes from db-cluster (192.168.240.179): icmp_seq=1 ttl=255 time=11.6 ms
64 bytes from db-cluster (192.168.240.179): icmp_seq=2 ttl=255 time=13.3 ms
64 bytes from db-cluster (192.168.240.179): icmp_seq=3 ttl=255 time=9.07 ms
^C
--- db-cluster.ats.local ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 9.071/11.368/13.348/1.760 ms

oracle@justinbspc1:~> ping db-cluster
PING db-cluster.ats.local (192.168.240.181) 56(84) bytes of data.
64 bytes from db-cluster.ats.local (192.168.240.181): icmp_seq=1 ttl=255
time=7.66 ms
64 bytes from db-cluster.ats.local (192.168.240.181): icmp_seq=2 ttl=255
time=19.7 ms
^C
--- db-cluster.ats.local ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 7.663/13.709/19.755/6.046 ms

oracle@justinbspc1:~> ping db-cluster
PING db-cluster.ats.local (192.168.240.180) 56(84) bytes of data.
64 bytes from db-cluster.ats.local (192.168.240.180): icmp_seq=1 ttl=255
```

```
time=12.7 ms
64 bytes from db-cluster.ats.local (192.168.240.180): icmp_seq=2 ttl=255
time=1010 ms
64 bytes from db-cluster.ats.local (192.168.240.180): icmp_seq=3 ttl=255
time=12.5 ms
^C
--- db-cluster.ats.local ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 12.555/345.102/1010.028/470.173 ms, pipe 2
```

On the AIX database server Aixdb1 lets start Oracle clusterware:

```
# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)

# hostname
Aixdb1

# uname
AIX

# ps -ef | grep -i smon | grep -iv grep
#

# grep -i grid /etc/oratab
+ASM1:/u01/app/12.1.0/grid:N:           # line added by Agent

# cd /u01/app/12.1.0/grid/bin
# ./crsctl start crs
CRS-4123: Oracle High Availability Services has been started.

# ps -ef | grep -i smon | grep -iv grep
    grid 10092782        1   0 09:26:35      -  0:00 asm_smon_+ASM1
  oracle 26673346        1   0 09:29:27      -  0:00 ora_smon_db1
```

Oracle clusterware is configured to start both our ASM instance and our database instance db1:

```
# su - oracle
ORACLE_SID = [oracle] ? +ASM1
The Oracle base has been set to /u01/app/12.1.0/grid

$ srvctl status database -d db
Instance db1 is running on node Aixdb1
Instance db2 is not running on node Aixdb2

$ srvctl status asm
ASM is running on Aixdb1
```

Let’s now start Oracle clusterware on the second node in our cluster – AIX server Aixdb2:

```
# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)

# hostname
Aixdb2

# uname
AIX

# ps -ef | grep -i smon | grep -iv grep
#

# grep -i grid /etc/oratab
+ASM2:/u01/app/12.1.0/grid:N:           # line added by Agent

# cd /u01/app/12.1.0/grid/bin
# ./crsctl start crs
CRS-4123: Oracle High Availability Services has been started.

# ps -ef | grep -i smon | grep -iv grep
    grid  6684844        1   0 09:53:18      -  0:00 asm_smon_+ASM2
  oracle 12845192        1   0 09:55:29      -  0:00 ora_smon_db2

# su - oracle
ORACLE_SID = [oracle] ? +ASM2
The Oracle base has been set to /u01/app/12.1.0/grid

$ srvctl status database -d db
Instance db1 is running on node Aixdb1
Instance db2 is running on node Aixdb2

$ srvctl status asm
ASM is running on Aixdb1,Aixdb2
```

Back on the first node in the cluster let’s verify that we can connect to the first Oracle database instance via OS authentication and verify that this is indeed a RAC database:

```
$ hostname
Aiixdb1

$ id
uid=206(oracle) gid=202(oinstall)
groups=203(dba),209(asmdba),211(backupdba),212(dgdba),213(kmdba)
```

```
$ . oraenv
ORACLE_SID = [oracle] ? db1
The Oracle base has been set to /u05/app/oracle

$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Thu Jun 11 10:03:01 2015

Copyright (c) 1982, 2009, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Release 11.2.0.1.0 - 64bit Production
With the Real Application Clusters and Automatic Storage Management options

SQL> show user;
USER is "SYS"

SQL> select name from v$database;

NAME
---------
DB

SQL> select instance_name from v$instance;

INSTANCE_NAME
----------------
db1

SQL> select instance_number from v$instance;

INSTANCE_NUMBER
---------------
              1

SQL> select thread# from v$instance;

   THREAD#
----------
         1
```

On node # 2:

```
$ . oraenv
ORACLE_SID = [oracle] ? db2
The Oracle base has been set to /u05/app/oracle
```

```
$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Thu Jun 11 10:03:01 2015

Copyright (c) 1982, 2009, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Release 11.2.0.1.0 - 64bit Production
With the Real Application Clusters and Automatic Storage Management options

SQL> show user;
USER is "SYS"

SQL> select name from v$database;

NAME
---------
DB

SQL> select instance_name from v$instance;

INSTANCE_NAME
----------------
db2

SQL> select instance_number from v$instance;

INSTANCE_NUMBER
---------------
              2

SQL> select thread# from v$instance;

   THREAD#
----------
         2
```

Create the database user that our Java application will be connecting to the database as and create the schema which will own the database objects which our Java application will be using. Note, this can be done from any instance in the cluster. Since you are on the second node in the cluster currently, that will do.

```
SQL> select tablespace_name from dba_tablespaces;

TABLESPACE_NAME
------------------------------
```

```
SYSTEM
SYSAUX
UNDOTBS1
TEMP
USERS
UNDOTBS2

6 rows selected.

SQL> select name from v$datafile;

NAME
-----------------------------------------------------------------------------
---
+DATA/db/datafile/system.269.864599303
+DATA/db/datafile/sysaux.270.864599303
+DATA/db/datafile/undotbs1.271.864599303
+DATA/db/datafile/users.272.864599303
+DATA/db/datafile/undotbs2.277.864599419

SQL> create tablespace app_data01 datafile '+DATA' size 100m;

Tablespace created.

SQL> select tablespace_name from dba_tablespaces where tablespace_name =
'APP_DATA01';

TABLESPACE_NAME
------------------------------
APP_DATA01

SQL> create user app identified by app123 default tablespace app_data01;

User created.
```

Every Oracle database user, human or app, needs to have the CREATE SESSION Oracle system privilege granted, otherwise they won't be able to do anything in Oracle.
We will also grant the Oracle system privilege CREATE TABLE as well as set the quota of the tablespace we created earlier to unlimited. This will allow our app to write to the table in our schema which will reside in the app_data01 tablespace.

```
SQL> grant create session to app;

Grant succeeded.

SQL> grant create table to app;

Grant succeeded.
```

```
SQL> alter user app quota unlimited on app_data01;

User altered.
```

Change users from SYS to user app, our schema owner, and create the schema objects, i.e. table we will use.
Note, in more advanced enterprise applications there will be many more schema objects supporting the application such as views, indexes, sequences, etc…

```
SQL> connect app/app123;
Connected.

SQL> show user;
USER is "APP"

SQL> create table contacts(lname varchar2(20),
  2  fname varchar2(20),
  3  age number(3),
  4  address varchar2(60),
  5  city varchar2(60),
  6  zip number(5));

Table created.


SQL> desc contacts;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------------------
---
 LNAME                                              VARCHAR2(20)
 FNAME                                              VARCHAR2(20)
 AGE                                                NUMBER(3)
 ADDRESS                                            VARCHAR2(60)
 CITY                                               VARCHAR2(60)
 ZIP                                                NUMBER(5)

SQL> select * from contacts;

no rows selected
```

The table our application will need, contacts, has six columns. Each column has a specific datatype. No constraints, remember this is a very simple example.
This one table will hold our contact information. The table represents a person/contact entity and each attribute of the person is represented by columns.
Specific information about the person will be stored in the columns of this table.

Columns of the contacts table:	Description:
LNAME	This column will store the last name of the contact.
FNAME	This column will store the first name of the contact.
AGE	This column will store the age of the contact. The datatype is NUMBER(3) which means that only three numbers max can be inserted here.
ADDRESS	This column will store the address of the contact
CITY	This column will store the city of the contact.
ZIP	This column will store the zip code of the contact's city.

Back on the first node of the cluster, we will see the table we created in the database from node 2. That is because this is the same database accessed by two instances.

```
# su - oracle
ORACLE_SID = [oracle] ? db2
The Oracle base for ORACLE_HOME=/u05/app/oracle/product/11.2.0/dbhome_1 is
/u05/app/oracle

$ id
uid=206(oracle) gid=202(oinstall)
groups=203(dba),209(asmdba),212(dgdba),213(kmdba)

$ hostname
Aixdb1
```

Connect to the database via Sqlplus as the app database user on node # 1:

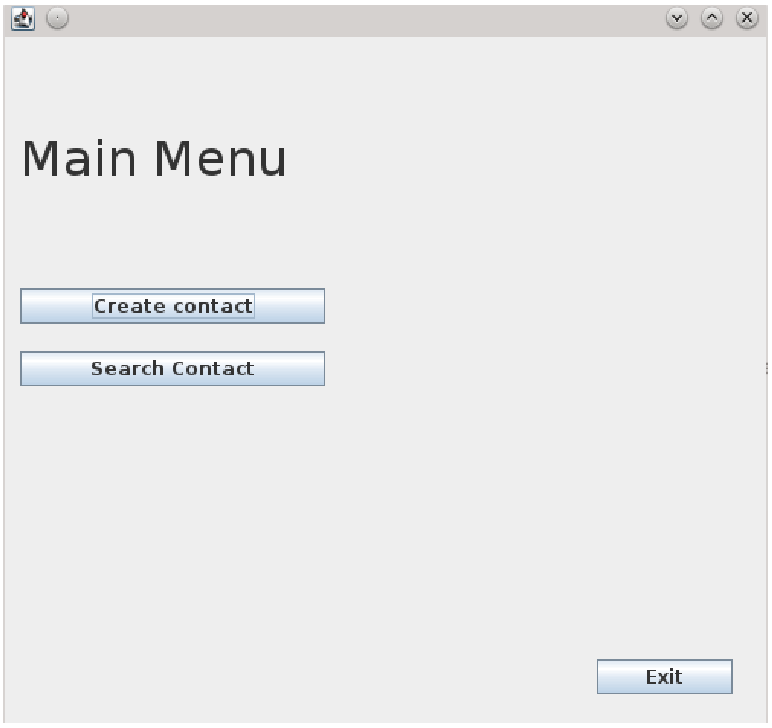
```
$ sqlplus

SQL*Plus: Release 11.2.0.1.0 Production on Thu Jun 11 10:12:43 2015

Copyright (c) 1982, 2009, Oracle.  All rights reserved.

Enter user-name: app
Enter password:

Connected to:
```

Oracle Database 11g Release 11.2.0.1.0 - 64bit Production
With the Real Application Clusters and Automatic Storage Management options

```
SQL> show user;
USER is "APP"

SQL> desc contacts;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------------------
---
 LNAME                                              VARCHAR2(20)
 FNAME                                              VARCHAR2(20)
 AGE                                                NUMBER(3)
 ADDRESS                                            VARCHAR2(60)
 CITY                                               VARCHAR2(60)
 ZIP                                                NUMBER(5)

SQL> select * from contacts;

no rows selected
```

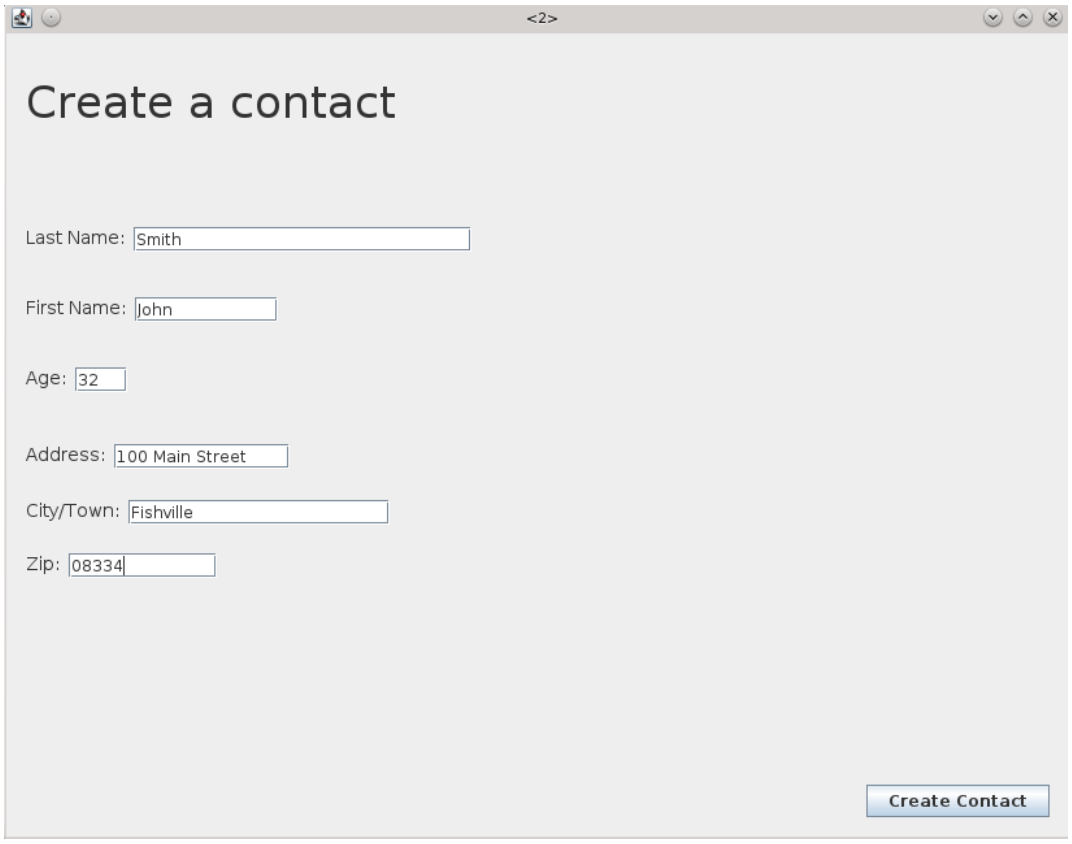
Back on my Linux PC where I have my oraconnect JAVA application "installed" I will run it:

```
$ java -jar oraconnect.jar
Oraconnect Version 1.0 - Justin Bleistein (ATS)
```



On the main menu screen I will click the button “Create contact”. You will then be presented with the “Create a contact” GUI form.

This form provides you with text areas where you can input the information about a contact. When you click the “Create Contact” button, my application uses the values entered in these text areas to generate the INSERT statement which will be sent to the database.



Last Name: Smith

First Name: John

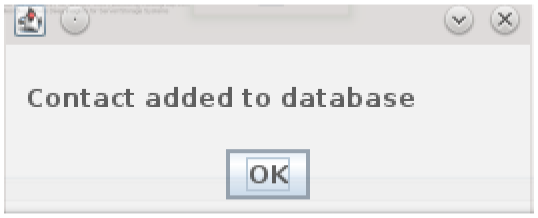
Age: 32

Address: 100 Main Street

City/Town: Fishville

Zip: 08334

Once the contact information is entered click the “Create Contact” button.


Contact added to database

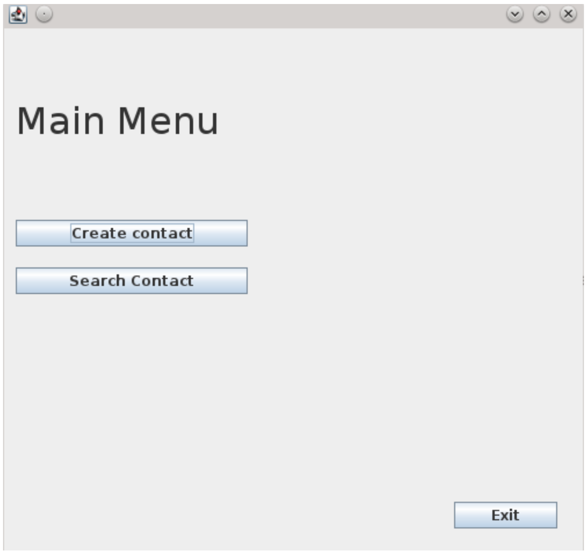
Now connect the database and see that your first contact entry has been added to the CONTACTS table. Again it does not matter which instance you connect to the database from:

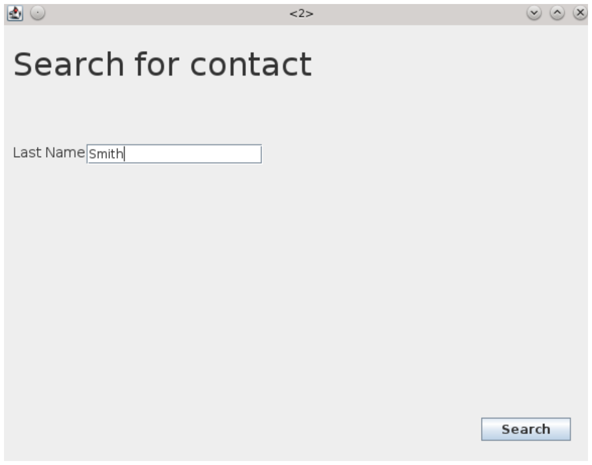
```
SQL> show user;
USER is "SYS"

SQL> select instance_name from v$instance;

INSTANCE_NAME
----------------
db1

SQL> select host_name from v$instance;

HOST_NAME
----------------------------------------------------------------
Aixdb1

SQL> set linesize 300;
SQL> select * from app.contacts;


LNAME                FNAME                           AGE ADDRESS
CITY                                                                   ZIP
-------------------- -------------------- ---------- ------------------------
------------------------------------ ----------------------------------------
-------------------- ----------
Smith                John                             32 100 Main Street
Fishville                                                             08334

SQL>
```

Same data will be seen from the other instance. Again this is the same database; it is just access by two instances.

```
SQL> show user;
USER is "SYS"

SQL> select instance_name from v$instance;

INSTANCE_NAME
----------------
db2

SQL> select host_name from v$instance;

HOST_NAME
----------------------------------------------------------------
Aixdb2

SQL> set linesize 300;
SQL> select * from app.contacts;

LNAME                FNAME                           AGE ADDRESS
CITY                                                                     ZIP
-------------------- -------------------- ---------- ------------------------
------------------------------------ ----------------------------------------
-------------------- ----------
Smith                John                             32 100 Main Street
Fishville                                                               08334
```

Back on the laptop run the application again. This time click the button “Search Contact”:

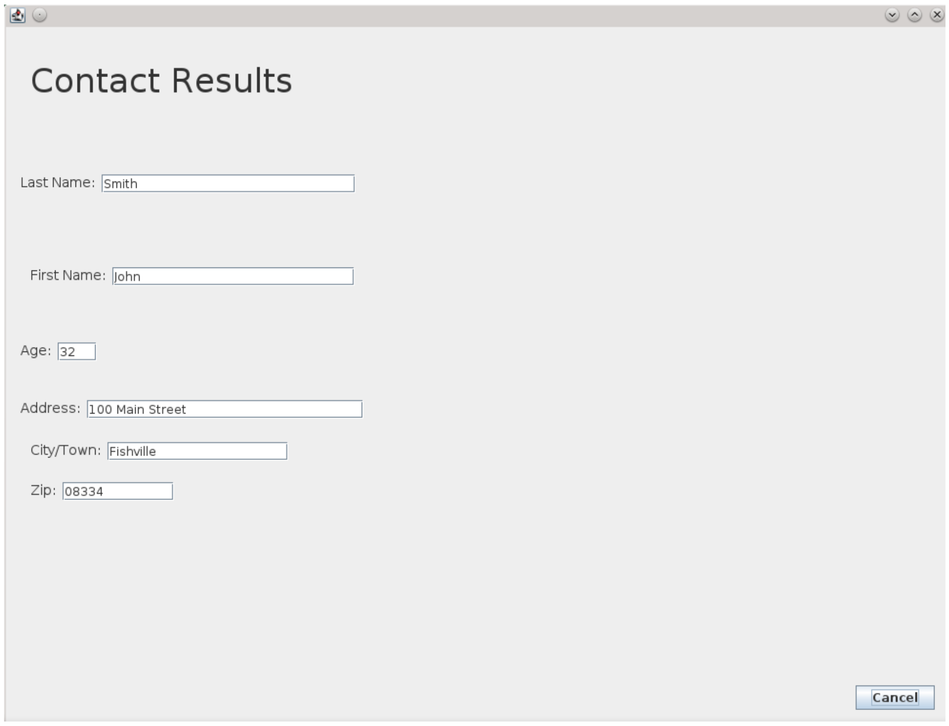




Type in the last name of the contact you entered earlier.

Click the “Search” button and you will then see a results screen. When you click the “Search” button, my application uses the value entered in this text area to generate the SELECT statement which will be sent to the database.

The data set result of the query is shown in the window below:



Contact Results

Last Name: Smith

First Name: John

Age: 32

Address: 100 Main Street

City/Town: Fishville

Zip: 08334

Let’s now simulate a database server/node failure.
Log into node # 1 and verify that the ASM and RDBMS instances are both running and that the TNS listeners are running currently:

```
# hostname
Aixdb1

# ps -ef | grep -i smon | grep -iv grep
    grid  8650790        1   0   Jun 15      -  0:04 asm_smon_+ASM1
  oracle 12058742        1   0   Jun 15      -  0:14 ora_smon_db1

# ps -ef | grep -i tns | grep -iv grep
    grid 10813530        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER -no_crs_notify -inherit
    grid 11141226        1   0   Jun 15      -  0:12
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN1 -no_crs_notify –inherit
```

```
On node # 2:

# hostname
Aixdb2

# ps -ef | grep -i smon | grep -iv grep
    grid 10027058        1   0   Jun 15      -  0:03 asm_smon_+ASM2
  oracle 12910730        1   0   Jun 15      -  0:13 ora_smon_db2

# ps -ef | grep -i tns | grep -iv grep
    grid 11337854        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN2 -no_crs_notify -inherit
    grid 11731046        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN3 -no_crs_notify -inherit
    grid 11993198        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER -no_crs_notify -inherit
#
```

Halt the first node to simulate a system crash:

```
# hostname
Aixdb1

# halt

…
```

Attempt to ping the first node from the laptop. You will receive no response because it is down:

```
$ ping Aixdb1
PING Aixdb1.ats.local (192.168.240.195) 56(84) bytes of data.
…
```

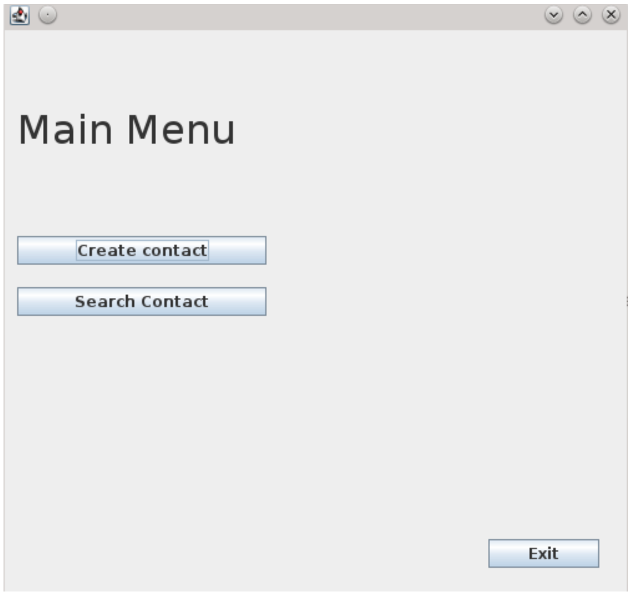
On node # 2, we see that our Oracle processes are still up and running. We also notice that the SCAN1 listener has moved over to node # 2 from node # 1. Oracle does this automatically:

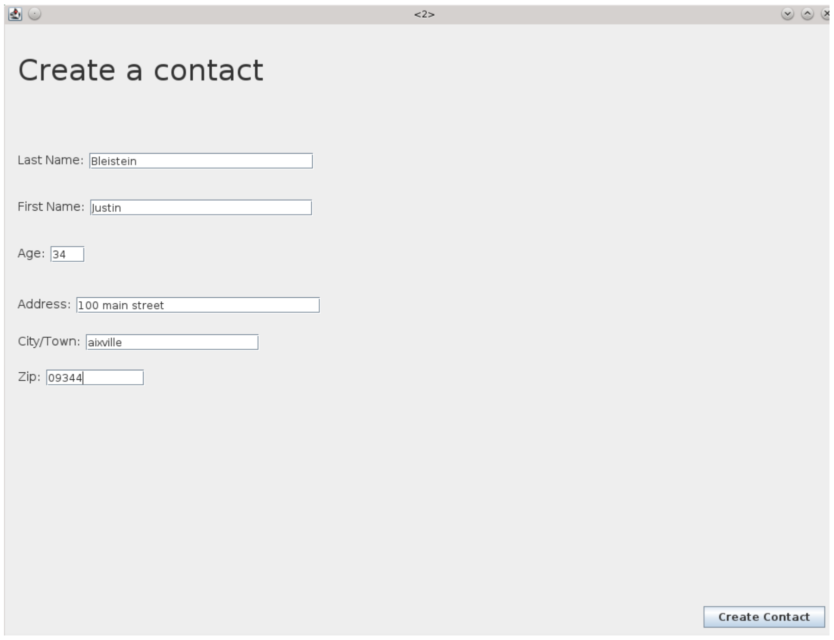
```
# hostname
Aiixdb2

# ps -ef | grep -i smon | grep -iv grep
    grid 10027058        1   0   Jun 15      -  0:03 asm_smon_+ASM2
  oracle 12910730        1   0   Jun 15      -  0:14 ora_smon_db2

# ps -ef | grep -i tns | grep -iv grep
    grid 11337854        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN2 -no_crs_notify -inherit
    grid 11731046        1   0   Jun 15      -  0:11
```

```
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN3 -no_crs_notify -inherit
    grid 11993198        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER -no_crs_notify -inherit
    grid 21561532        1   0 18:04:56      -  0:00
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN1 -no_crs_notify -inherit
```

Run the application again from the laptop:

Click the “Create contact” button:




Create a contact

Last Name: Bleistein

First Name: Justin

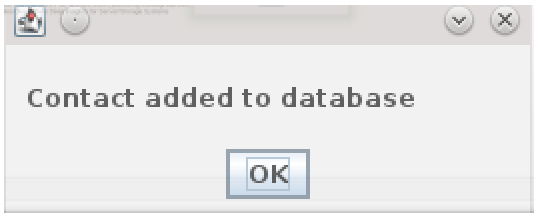
Age: 34

Address: 100 main street

City/Town: aixville

Zip: 09344

Input the contact values and then click the “Create Contact” button.


Contact added to database

The application still worked because the SCAN IP that the application is programmed to use redirected it to a surviving node, Aixdb2, which was running a surviving instance, Db2.
Even this simple client/server application experienced no interruption due to one of the nodes going down. Oracle knew what to do and did it behind the scenes. Note, and I can't stress this enough, given the trivial nature of our test application we have to reconnect to the cluster for the SCAN IP to redirect us to the surviving instance. No session on query state information was moved, a new database connection was just made. This new database connection could result from restarting the application, like we did, or at a certain part in the already running application, it depends how it is coded. This is a good point to note that the Oracle RAC infrastructure can handle the initial connection redirections etc but it is up to the application code to handle the session state redirections.

We will see the data in the CONTACTS table of the APP schema as well:

```
SQL> set linesize 300;
SQL> select * from app.contacts;

LNAME                FNAME                           AGE
ADDRESS
CITY                                                                    ZIP
-------------------- -------------------- ---------- ------------------------------------
------------------------ ------------------------------------------------------------- ---
-----
Bleistein            Justin                           34 100 main
street
aixville                                                               09344
Smith                John                             32 100 Main
Street
Fishville                                                              08334
```

If we didn't use a VIP or a SCAN IP address because your database is running in a single instance non-RAC configuration, then we would probably see a JAVA stack trace error similar to the following. You would have to restart the non-RAC database server and database for your application to regain connectivity to the database.

```
java.sql.SQLRecoverableException: IO Error: The Network Adapter could not
establish the connection
```

```
        at oracle.jdbc.driver.T4CConnection.logon(T4CConnection.java:489)
        at
oracle.jdbc.driver.PhysicalConnection.<init>(PhysicalConnection.java:553)
        at oracle.jdbc.driver.T4CConnection.<init>(T4CConnection.java:254)
        at
oracle.jdbc.driver.T4CDriverExtension.getConnection(T4CDriverExtension.java:32)
        at oracle.jdbc.driver.OracleDriver.connect(OracleDriver.java:528)

...
```

Works the other way as well, now halt the second node in the cluster – node # 2:

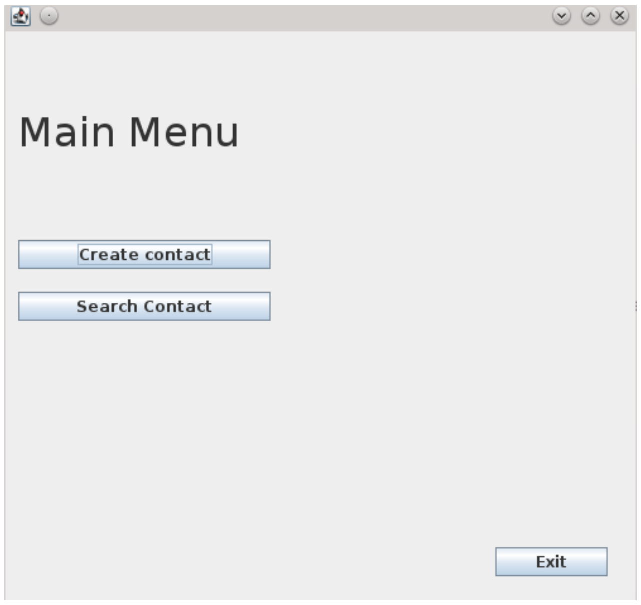
```
# hostname
Aixdb1

# uptime
  06:10PM   up 5 mins,  1 user,  load average: 1.28, 1.17, 0.56

# date
Sun Jun 21 18:10:44 EDT 2015

# who -b
   .        system boot Jun 21 18:05

# ps -ef | grep -i smon | grep -iv grep
    grid  9371684        1   0 18:08:00      -  0:00 asm_smon_+ASM1
  oracle 12910730        1   0 18:09:01      -  0:00 ora_smon_db1

# ps -ef | grep -i tns | grep -iv grep
    grid 10682456        1   0 18:08:29      -  0:00
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER -no_crs_notify -inherit
    grid 10879090        1   0 18:08:30      -  0:00
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN1 -no_crs_notify -inherit


# hostname
Aiixdb2

# ps -ef  |grep -i smon | grep -iv grep
    grid 10027058        1   0   Jun 15      -  0:03 asm_smon_+ASM2
  oracle 12910730        1   0   Jun 15      -  0:14 ora_smon_db2

# ps -ef | grep -i tns | grep -iv grep
    grid 11337854        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN2 -no_crs_notify -inherit
    grid 11731046        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN3 -no_crs_notify -inherit
    grid 11993198        1   0   Jun 15      -  0:11
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER -no_crs_notify -inherit
```

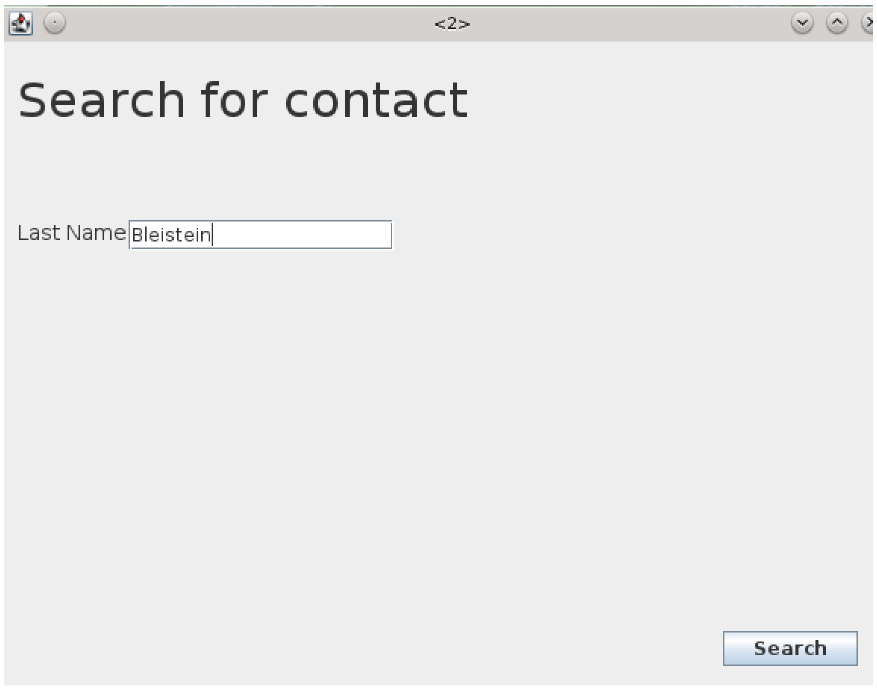
```
# halt
…


# hostname
Aixdb1

# ps -ef | grep -i smon | grep -iv grep
    grid  9371684        1   0 18:08:00      -  0:00 asm_smon_+ASM1
  oracle 12910730        1   0 18:09:01      -  0:00 ora_smon_db1

# ps -ef | grep -i tns | grep -iv grep
    grid 10682456        1   0 18:08:29      -  0:00
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER -no_crs_notify -inherit
    grid 10879186        1   0 18:16:57      -  0:00
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN1 -no_crs_notify -inherit
    grid 15139030        1   0 18:12:21      -  0:00
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN3 -no_crs_notify -inherit
    grid 15335642        1   0 18:12:21      -  0:00
/u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN2 -no_crs_notify -inherit
```

Run the application again from the laptop and this time just search for a contact and you will see that this works as well because once again behind the scenes Oracle RAC saw that node # 2 was down so it automatically redirected the application to the surviving node, Aixdb1, which was running the surviving instance, Db1.
To reiterate yet again, keep in mind that these failovers require you to restart your application or at least go back to a point where the application is reinitializing the database connection via JDBC. A lot of transparent database failure, in regards to application availability, is handled by the application side. As noted earlier this specific topic is out of scope of this blog entry but it is imperative to understand this point. You can think of applications which are programmed in this matter to be – “RAC Aware”.

Start the application again and see that you again still have functionality which the database is facilitating.

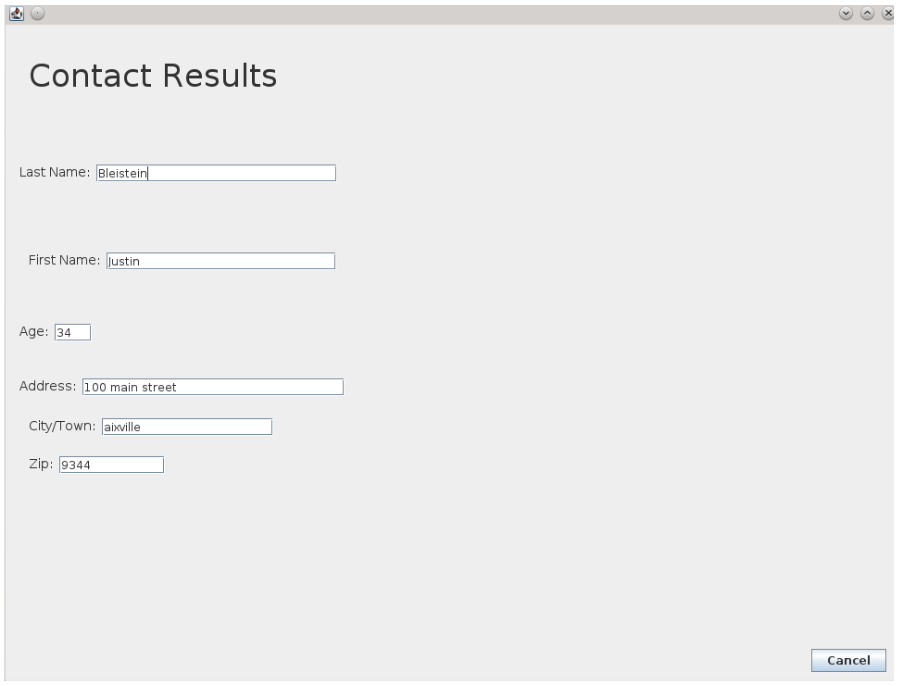
Click the “Create contact” button:





Search for a contact by typing in the last name and then clicking the “Search” button:

View the contact attribute results from the database:


Contact Results

Last Name: Bleistein

First Name: Justin

Age: 34

Address: 100 main street

City/Town: aixville

Zip: 9344

Don’t close the search result window above.

One last test, we will now test to see what happens if we lose a node while the application is “opened”. I put quotes around the word opened because again this is a very primitive client/server application. It being classified as running or opened is very loose in terminology.

After we do a search for a contact our JAVA application makes a connection to the cluster. In my JAVA code I do not close the database connection until the “Cancel” button of the “Contact Results” window is clicked.
The SCAN IP directed the database connection through node # 2:

```
# hostname
Aixdb2

# ps -ef | grep -i "local=no"
    grid 25755772        1   0 13:43:55      -  0:00 oracledb2 (LOCAL=NO)
```

Note, the Oracle SCAN mechanism has an internal algorithm to determine which node the application connection gets sent to. This decision is based on, among other things, the current performance load on a particular instance.

```
# hostname
Aixdb1
```

```
# ps -ef | grep -i "local=no"
#
```

From database db1:

```
SQL> select inst_id from gv$session where machine = 'justinbspc1';

   INST_ID
----------
         2

SQL> select sid from gv$session where machine = 'justinbspc1';

       SID
----------
        99

SQL> SELECT P.SPID, S.SID, S.SERIAL# FROM V$PROCESS P, V$SESSION SWHERE
P.ADDR = S.PADDR AND S.SID =  99;

SPID                            SID    SERIAL#
------------------------ ---------- ----------
25755772                         99      18715
```

PID matches the PID running on node #2.

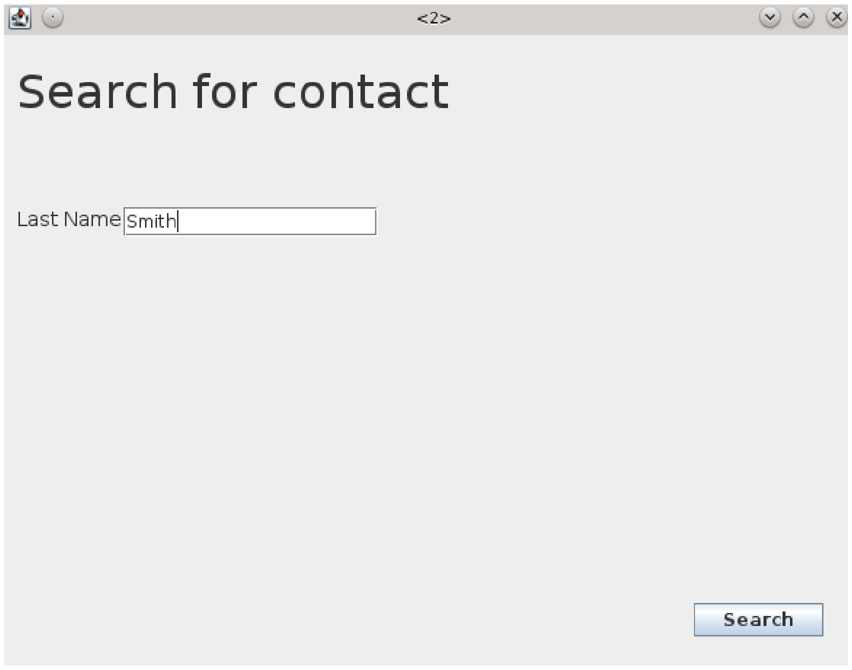
Note, despite the fact that our application is actually connected to the database via the instance Db2 which runs on Aixdb2, we can still see it from any node in the cluster via the global RAC GV$SESSION view.

Now forcefully shutdown node # 2, the node that our application is currently connected to:

```
# hostname
Aixdb2

# reboot
...
```

You won't see an error from the application because again it is opened by not really doing any type of database health checks or anything in the background because it is a primitive application.

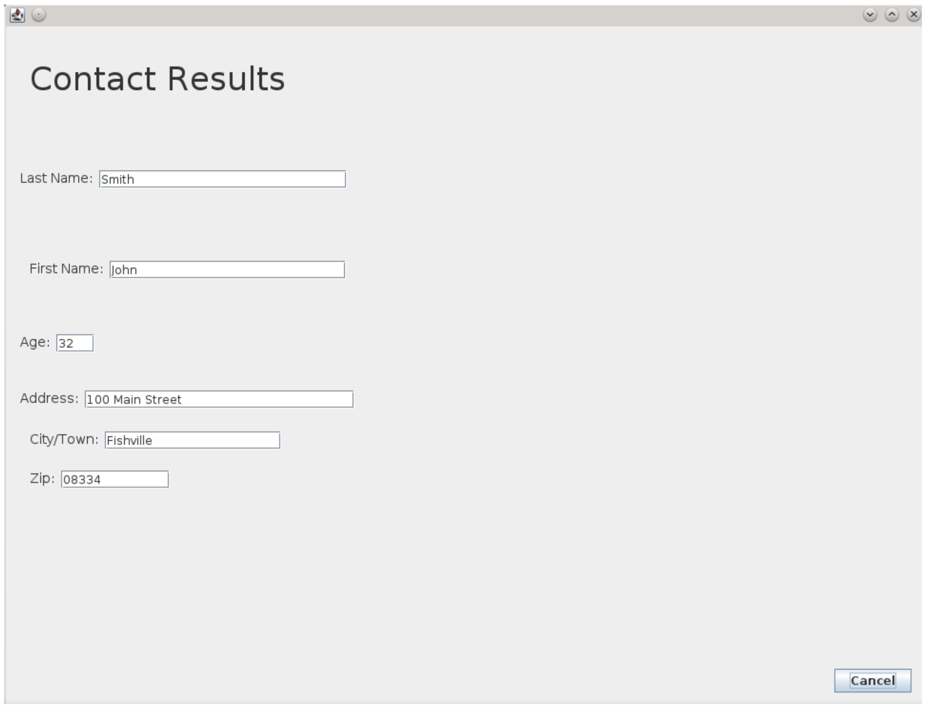
Close the "Contact Results" window by clicking the "Cancel" button. That should drop you back to the "Search for contact" window because in my JAVA code I did not close the "Search for contact" window just because a sub-window off of that window was opened – "Contact Results".



Search for contact

Last Name: Smith

Fill in your search criteria and then click the button “Search”:



Contact Results

Last Name: Smith

First Name: John

Age: 32

Address: 100 Main Street

City/Town: Fishville

Zip: 08334

On node # 1:

```
# hostname
Aiixdb1

# ps -ef  |grep -i "local=no"
    grid 25166052        1   0 14:00:27      -  0:00 oracledb1 (LOCAL=NO)


SQL> select inst_id from gv$session where machine = 'justinbspc1';

   INST_ID
----------
         1

SQL> select sid from gv$session where machine = 'justinbspc1';

       SID
----------
       138

SQL> SELECT P.SPID, S.SID, S.SERIAL# FROM V$PROCESS P, V$SESSION SWHERE
P.ADDR = S.PADDR AND S.SID =  99;

SPID                            SID    SERIAL#
------------------------ ---------- ----------
25166052                         138      89667
```

As you can see, when we clicked the “Search” button on the “Search for contact” window the JAVA code initiated another database connection via the SCAN IP. The SCAN IP directed the application to the surviving instance running on the surviving node – node #1. This is the same behavior we saw earlier, when we would just restart our application after a RAC node failure. So you see? It is all in the JAVA code.