OpenEye
Scientific Software

OpenEye Oracle Cartridge

Release 1.0.0

OpenEye Scientific Software, Inc.

October 29, 2012

CONTENTS

CHAPTER

ONE

FRONT MATTER

CHAPTER

TWO

INTRODUCTION

The *OpenEye Oracle Cartridge* imbues Oracle databases with the ability to perform substructure and similarity searches on molecules and reactions. In addition, the cartridge is capable of searching molecular shapes on the same timescale as graph similarity searches. For OpenEye toolkit programmers, the OpenEye Oracle Cartridge provides a critical function for building chemical information systems.

CHAPTER

THREE

BACKGROUND

3.1 Structure Indexing

The cartridge provides domain indexes for columns that contain structures encoded in Daylight SMILES, MDL MOL (or SDF) blocks; reactions encoded as Daylight SMIRKS or MDL RXN blocks; and multi-conformer molecules encoded in OEBinary arrays. This is achieved by using an external Java RMI service to perform substructure, similarity and exact match chemical searches. The RMI service uses the Java wrappers of the OEChem toolkit to assist in SMILES parsing and chemical structure handling. The Oracle JVM is used to call out to the RMI service. A domain index can be created for any table column containing structural information: a varchar2 column containing SMILES or SMIRKS, a clob column containing a MDL MOL or RXN block, or a blob column containing an OEBinary byte array. This index can be uniquely named such that index_name = owner.table.column. On index building the RMI server connects to oracle and extracts structures by Oracle ROWID. The external index is built as a hash keyed by ROWID and with a data structure value which contains a fingerprint for the structure and the SMILES. The index is then cached locally as a serialized Java object. Although structure indexes can be created for a variety of structure encodings, index creation and manipulation is much faster for indexes built on SMILES columns.

The index tracks edits to the base table using a change log (or journal table) that contains ROWIDs and new values. Before performing searches the domain index loads new entries from the change log into the index. An index rebuild saves these changes and empties the log table. Substructure search is achieved by fingerprinting a query SMARTs pattern (or MDL Mol or Rxn query block) and screening out against fingerprints in the index. Those structures which test against the fingerprint are searched exhaustively for a match. Similarity search is done by fingerprinting a molecule and determining Tanimoto coefficients for target structures. Database molecules that exceed a minimum similarity are returned. Exact match search is done by filtering structures by fingerprint similarity first, and then comparing the canonical SMILES of the query and target structures.

The substructure search requires the creation of OEMol structures. These can be cached on the RMI server in a LRU (least-recently used) cache. They are not currently stored in Oracle and must be regenerated each time the RMI server starts. If enabled, the molecule cache can retain OEMols that it generates for search or all molecules in an index can be added to the cache. A performance benefit will be realized if the server has sufficient memory to cache all the structures in the database.

3.2 Fingerprints

The fingerprints used in the data cartridge are an implementation of the MACCS 166 keys. These are used for both screenout during substructure searches and in the calculation of molecular similarity for similarity searches.

When performing a similarity search, the query molecule is fingerprinted using the full set of MACCS 166 keys and the similarity of each database molecule to the query molecule is calculated by comparing their fingerprints using the Tanimoto equation.

When performing a substructure search, the query substructure is fingerprinted using a subset of the MACCS 166 keys. Each database molecule fingerprint is compared to the query substructure fingerprint to determine whether the query substructure is possibly contained within the database molecule. If it is, then a more time consuming atom-by-atom comparison is carried out to either confirm or deny the match. A subset of the keys is used to fingerprint substructure queries as there are many keys in the MACCS 166 keyset that represent features that cannot be assigned unambiguously in a substructure - as opposed to a complete molecule (e.g. feature no. 34 A=CH2).

A separate index type is available for performing similarity searches using OpenEye Graphsim fingerprints. Each index of this type consists of a memory resident fingerprint database. Tanimoto and Tversky similarity are computed from fingerprints using the following formulas:

Formula: $Sim_{Tanimoto}(A,B) = \frac{bothAB}{onlyA+onlyB+bothAB}$

and

Formula: $Sim_{Tversky}(A,B) = \frac{bothAB}{\alpha*onlyA+\beta*onlyB+bothAB}$

3.3 Shape

An additional index type is available for performing similarity searches on FastROCS shape databases derived from structure and conformer columns in database tables. When an index is created, the RMI server creates an external file of persistent multi-conformer OEMols. If the RMI server is running on a machine with GPU hardware and configured to run FastROCS, the external file is loaded into a shape database. For 2D columns (varchar2 columns containing SMILES or clob columns containing SDF entries), multiple OMEGA jobs are used to create the persistent muti-conformer OEMols. Since creating multi-conformer molecules using OMEGA is time-consuming, on most machines, any FastROCS index created from 2D compounds will likely be limited to a smaller number of compounds. For example, using a quad core workstation with Intel Xeon 3GHz processors, it took 2hrs 18m to build a FastROCS index of 9M conformers on a table of 100,000 SMILES and 11hrs 9min to build a FastROCS index of 44 conformers on a table of 500,000 SMILES structures. Multi-conformer molecules may be stored in Oracle as an OEBinary byte array in a blob column. For FastROCS indexes built on 3D columns the external index type uses the multi-conformer compounds stored in the database table and thus the index can support any size of table. For 2D columns on update new values are added to the FastROCS server by creating a multi-conformer OEMol using the OMEGA toolkit. For 3D columns the column value can be added unmodified to the FastROCS server. In order to save space and improve performance, it is recommended that OEMols be stored in rotor compressed format.

CHAPTER

FOUR

INSTALLATION AND TESTING

4.1 Quick Start

To install and test the cartridge on a clean Oracle database complete these steps:

1. *Check the target platform requirements*
2. *Run the Cartridge installer GUI*
3. *Test the installation*

4.2 Requirements

- Linux (RHEL5, RHEL6, Unbreakable) or Windows XP/Vista/7.
- Oracle 11G. Standard Edition One or higher. Oracle XE is not supported (as it lacks the Oracle Server JVM)
- Java JDK or JRE 1.7.
- Oracle JDBC driver
- Python cx_Oracle module (if using Python examples/scripts)
- A GPU server with FastROCS installed and configured (optional for shape searches)
- Memory requirements for the RMI server:
  - RMI in-memory search index requires aproximately 60 Mb per 100K molecule
  - RMI index for molecules compressed in OEDBMol format requires approximately 80 Mb per 100K molecules
- OMEGA2 and FastROCS executables and licenses if using shape indexes.
- Remote RMI servers must have all ports (bidirectional) open to the Oracle server.
- The Oracle instance and tnslsnr must be running prior to installation. (Please refer to Oracle documentation)

4.3 GUI-based Installation

1. *Select Installation Type*
2. *Enter Oracle Information*
3. *Install Cartridge*

4. *Grant Permissions*
5. *Configure FastROCS Server*
6. *Configure RMI Server*
7. *Load Test Data and Build Index*

A graphical user interface is provided to facilitate installation. The cartridge may be installed on a single server, or distributed across multiple servers with the Oracle and RMI servers on different machines. Oracle components must be installed on the Oracle server, but no such restriction exists for RMI servers.

The following information is necessary for successful installation

1. Oracle SID. This is the unique per-machine identifier of an Oracle database and is used by JDBC drivers to connect to Oracle.
2. Oracle database name (as defined in TNS_NAMES.ORA). This is the database name that SQL*Plus uses to connect to Oracle. The Oracle SID and database name are normally the same, athough the Oracle database name may contain an additional domain name. The SID can be no longer than 8 characters (an Oracle database gorgonzola.cheese.com would be truncated to an SID of gorgonzo).
3. Password for the Oracle SYS account.
4. A list of existing users to whom cartridge access shall be granted.
5. The maximum amount of RAM to allow the RMI server (60 Mb/100K molecules) and whether or not to cache molecules in memory.
6. Location of the OpenEye license file.
7. Locations of the FastROCS and OMEGA executables, if shape indexes are to be built.

Prior to installing remote RMI servers, the following tasks must be completed:

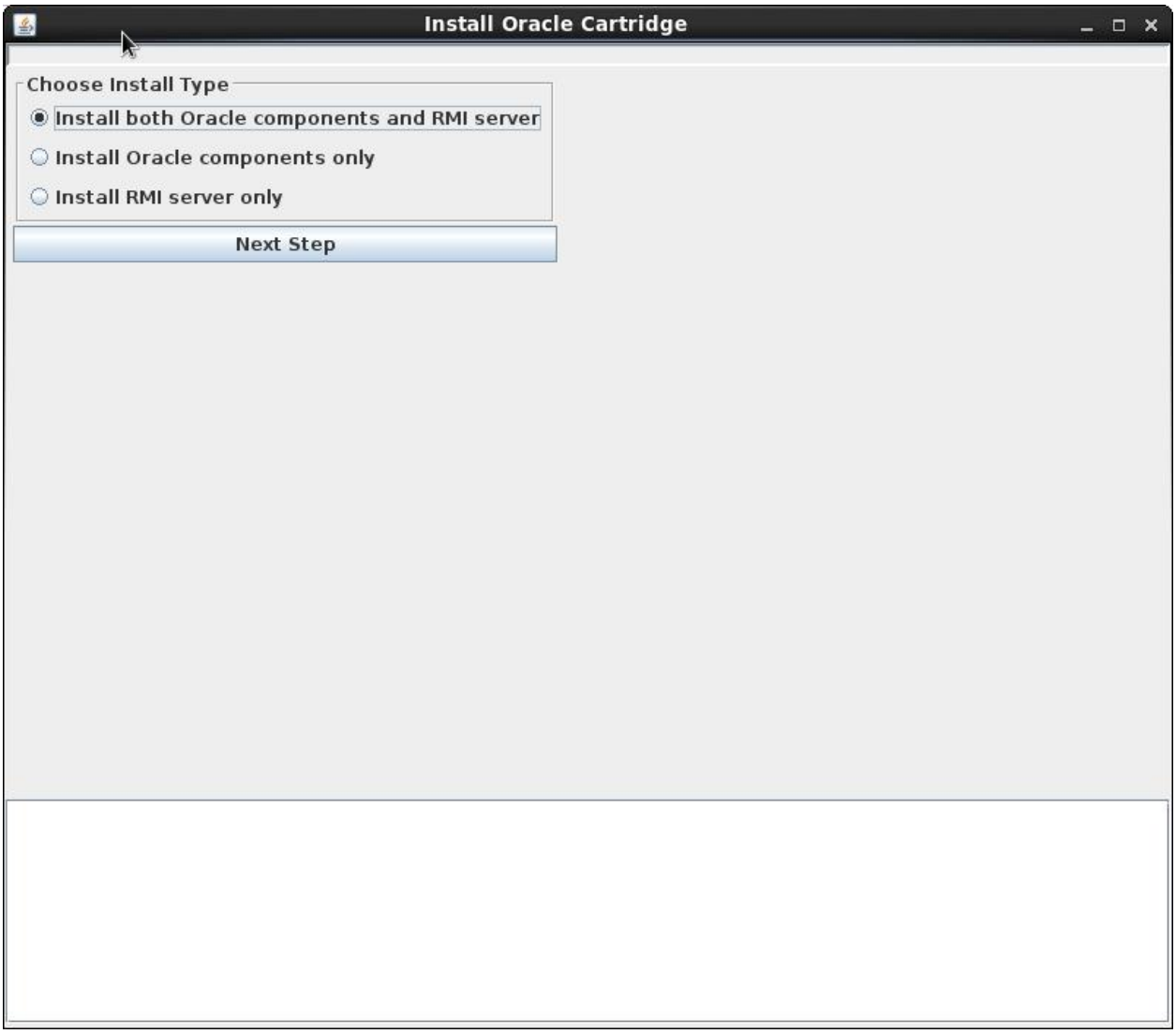
1. The Oracle components must be installed on the Oracle server.
2. The jdbc6.jar file needs to be copied to the lib directory of the unpacked cartridge distribution from which the RMI server will be installed. This file is located in the $ORACLE_HOME/jdbc/lib in any Oracle installation.

The installer logs messages in the directory log/install/<install date>. If installation fails, the install logs may help diagnose the problem.

To run the installer navigate to the bin subdirectory of the OpenEye cartridge distribution and run install.bat (Windows) or install.sh (Linux). The installer will not work properly if started from a working directory other than bin subdirectory. On Windows, the install.bat script should be run as Administrator. Installation of the cartridge service may fail if the installation is not performed from the Administrator account.

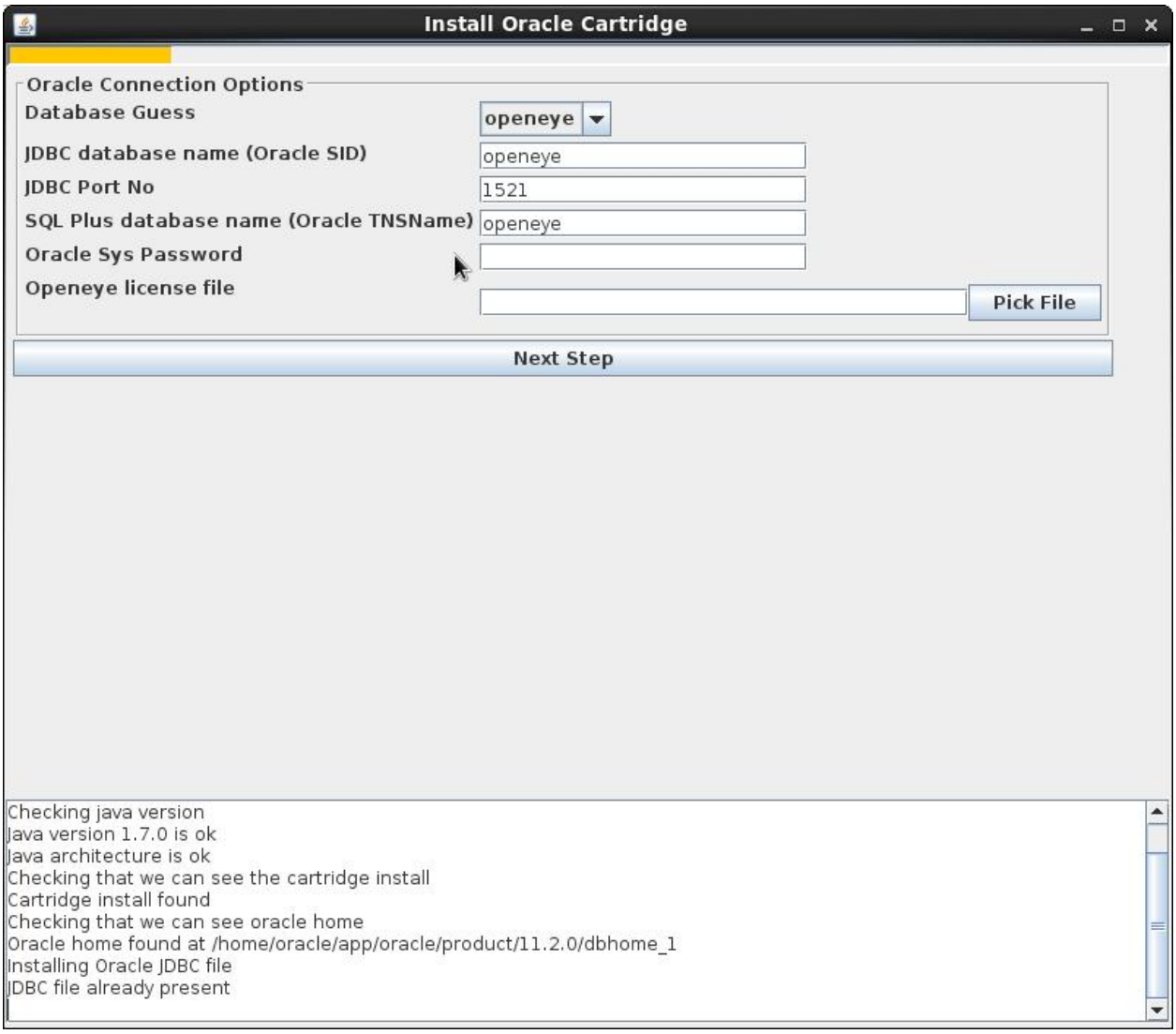
4.3.1 Select Installation Type

On the first screen select the type of installation to be performed. When installing on the Oracle server, it is standard to install both the Oracle components and RMI server. The *Install RMI server only* option should be used for a FastROCS server that is physically separate from the Oracle server.



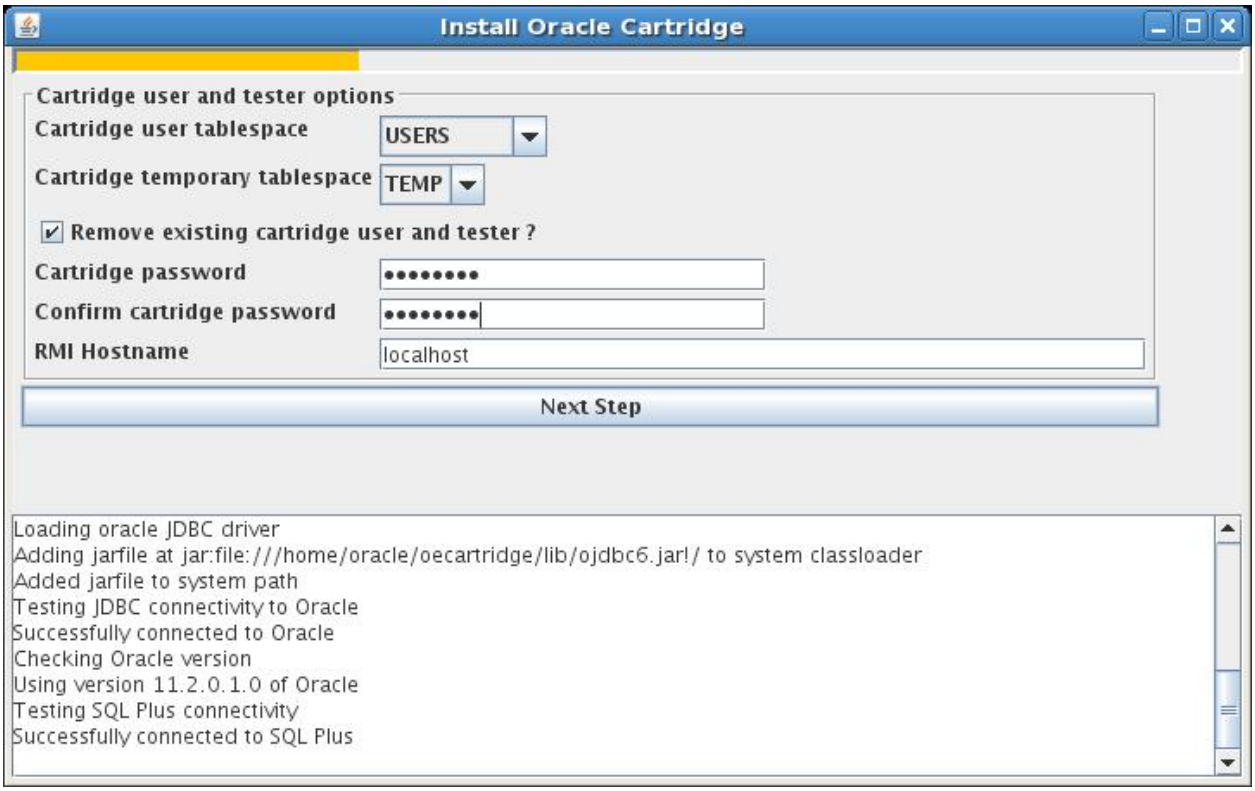
4.3.2 Enter Oracle Information

On the second screen enter the Oracle database information, and the location of the OpenEye Toolkit location and Cartridge license file.



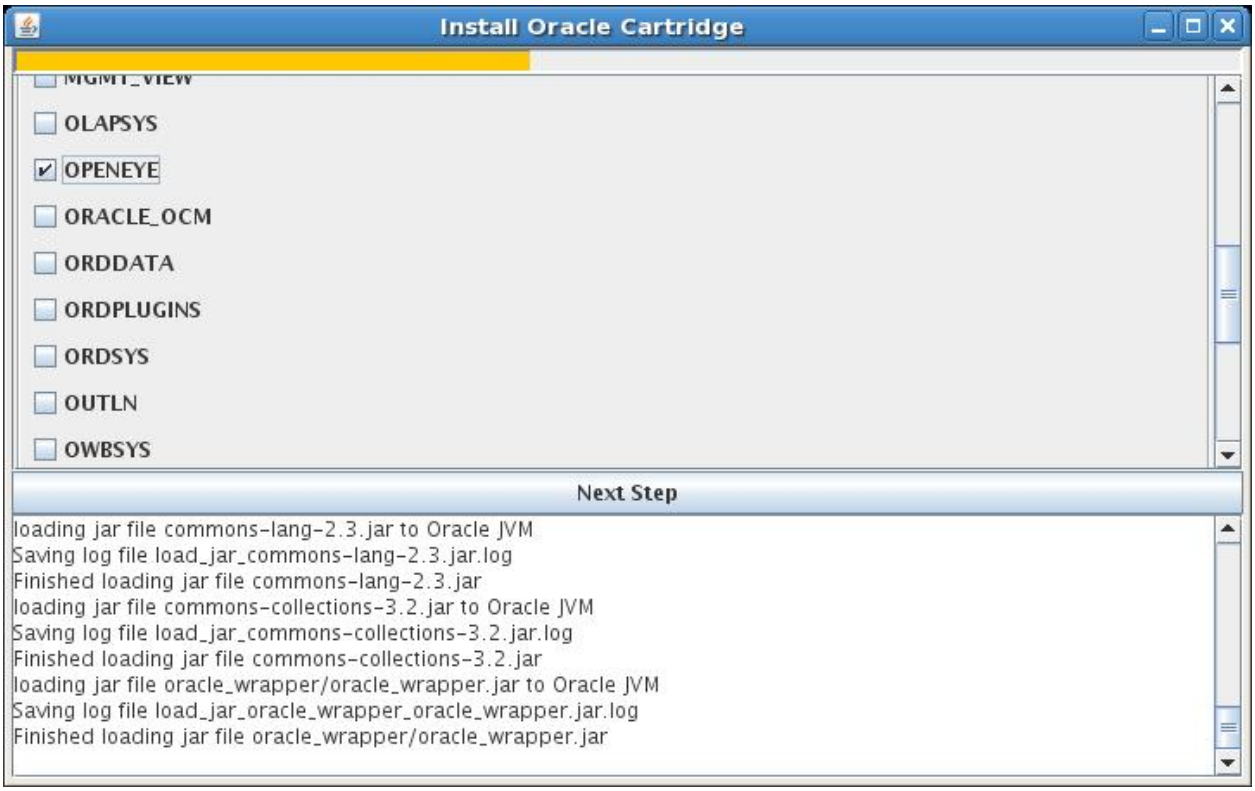
4.3.3 Install Cartridge

The tablespaces that the Cartridge user and tester will use may be selected on the next screen. Note that the installer does not create new tablespaces. Custom tablespaces for these users must be created beforehand. A shared password will need to specified for these users. Previous installations of Oracle components will have to be removed before proceeding. The name 'localhost' can be used as the RMI hostname for most installations where the RMI server is being installed on the same machine as the Oracle server. A particular hostname may need to be specified for servers that contain multiple network interface cards.



4.3.4 Grant Permissions

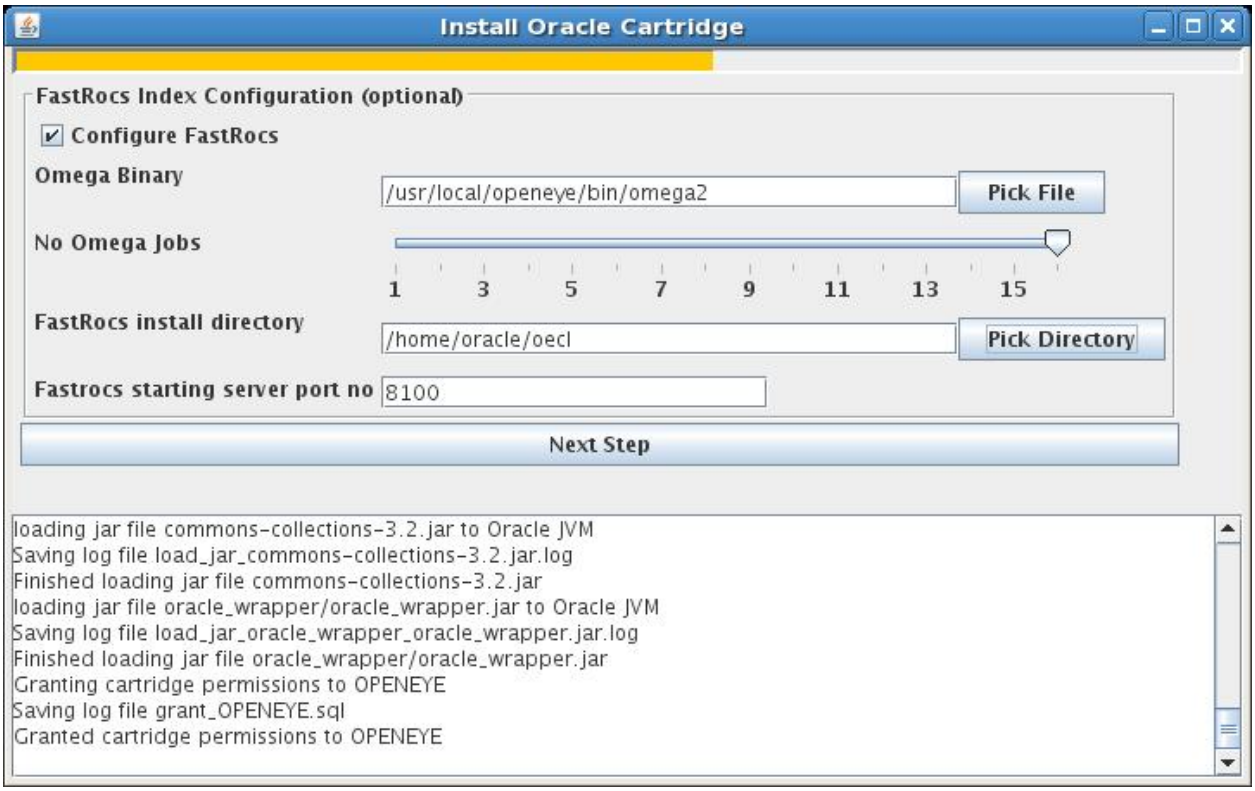
On the next screen, select from the list of existing users that will be granted access to the cartridge functionality. After cartridge installation, the SQL script DIR/sql/grant.sql may be used to grant permissions to additonal users. The test user ARNACHM1_TEST us automatically granted permission to use the cartridge. Note that Oracle comes with a large number of built-in users.



4.3.5 Configure FastROCS Server

If FastROCS domain indexes are to be used in conjunction with the cartridge, the 'Configure FastROCS' checkbox will enable configuration of the RMI server to interoperate with FastROCS. Note that this configuration should only be done on the FastROCS server. If the Oracle and FastROCS servers are on separate machines, install the cartridge on the Oracle server first, and then install an RMI server on the FastROCS server.

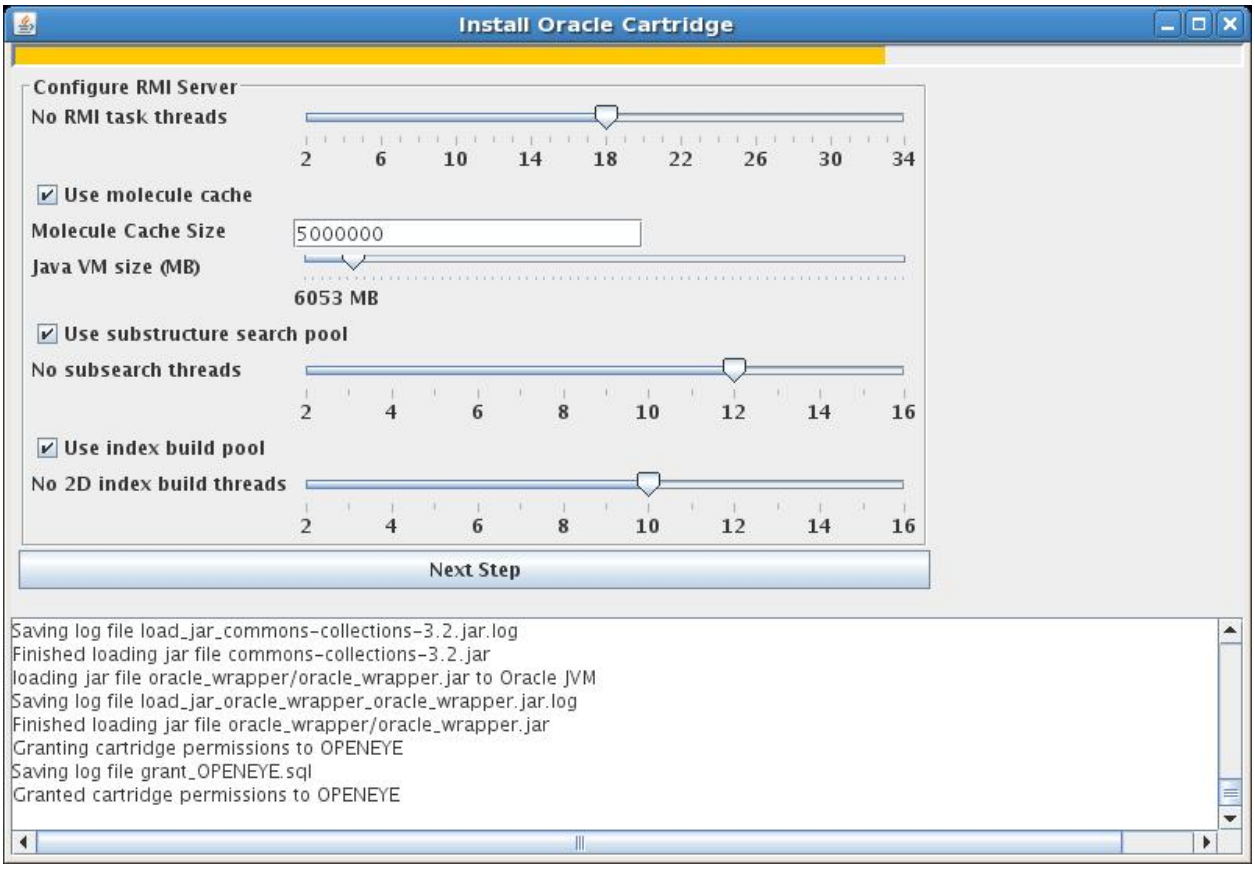
Specify the full path to the OMEGA binary. OMEGA is used to generate conformers when building or rebuilding indexes on SMILES or MOL file columns. The maximum number of parallel OMEGA processes to be used in building indexes may be specified with a slider that reflects the number of processes and cores on the FastROCS server. Next, the full path to the FastROCS installation directory must be provided, and the port number on which the RMI server will communicate with the FastROCS server. Note that the RMI server will spawn its own instance of the FastROCS server, so the port number should be chosen to avoid ports used by non-cartridge instances of the FastROCS server.



4.3.6 Configure RMI Server

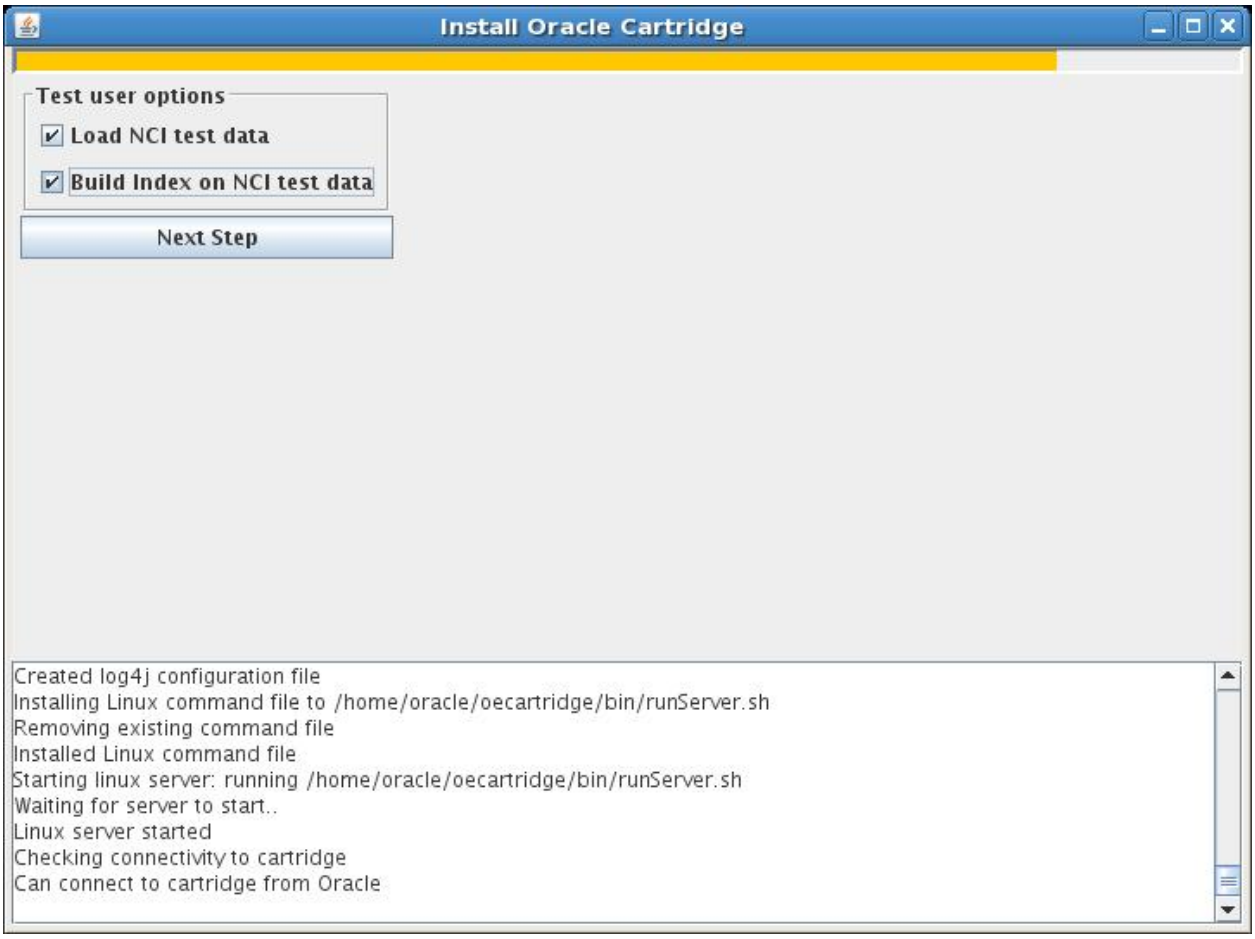
For the penultimate step select the number of RMI task threads (or the maximum number of concurrent requests the RMI server can service); the maximum memory usage for the Java RMI process; whether to enable the cache of OpenEye molecules and (if more than one processing core is present) whether to enable a dedicated thread pool for structure index creation. These settings are described in *configuring the RMI server*. Once this screen is completed the RMI server will be started and the connection between the test user and data cartridge validated. The server configuration is stored in a file called 'server.properties' in the bin subdirectory. Messages from the cartridge are logged in the file log/rmi/rmi_server.log (relative to the cartridge root). On a Windows machine Apache Procrun is used to install the RMI server as a Windows service (see http://commons.apache.org/daemon/procrun.html). The service can be edited and controlled from a GUI started using the batch file monitorService.bat created in the bin subdirectory. If running the RMI server as a process instead of a service is preferred, a batch file runServer.bat is created in the bin subdirectory of the cartridge installation, and may be started manually. Linux installations include a shell script that may be used to start the RMI server process manually. The aforementioned runServer.sh script is located in the bin subdirectory of the cartridge installation.

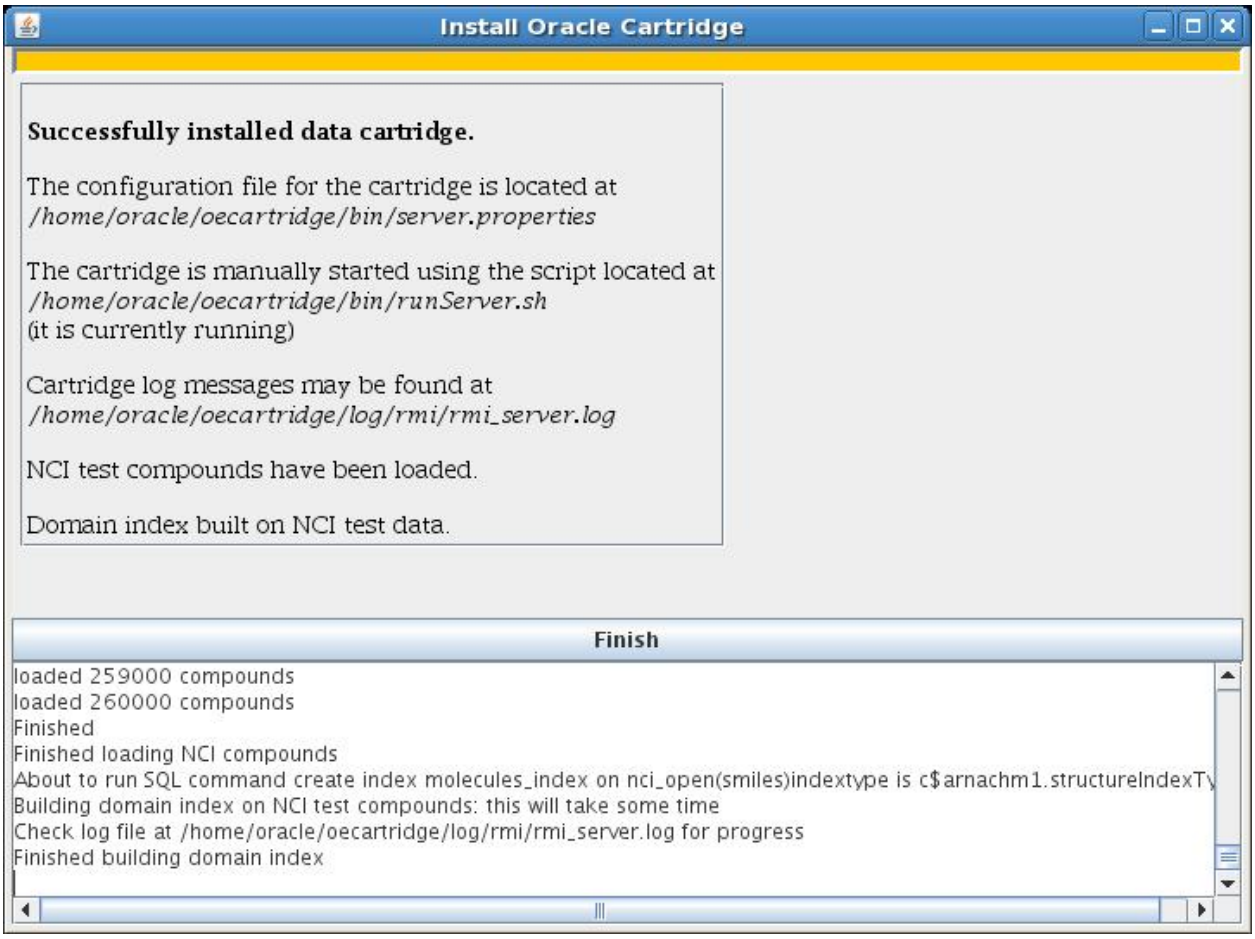
After cartridge installation, the RMI server may be altered to *manually load the indexes and cache on startup.* Whenever the configuration file is edited, the Java RMI process must be restarted.



4.3.7 Load Test Data and Build Index

In this optional step, the test data may be loaded into the cartridge tester's (ARNACHM1_TEST) schema. This loads 260K structure from the NCI open database into a table called nci_open. After loading the test data, a structure domain index may be built for the table. Once the structures are loaded and the index built, example commands may be run from SQL*Plus while connected as ARNACHM1_TEST user. Once this step is complete, a final screen shows a summary of the install.


Install Oracle Cartridge
Test user options
Load NCI test data
Build Index on NCI test data
Next Step
Created log4j configuration file
Installing Linux command file to /home/oracle/oecartridge/bin/runServer.sh
Removing existing command file
Installed Linux command file
Starting linux server: running /home/oracle/oecartridge/bin/runServer.sh
Waiting for server to start..
Linux server started
Checking connectivity to cartridge
Can connect to cartridge from Oracle



4.4 Starting and Stopping the RMI Server

4.4.1 Linux

The server can be stopped using the UNIX kill command. If FastROCS indexes have been created, it may be necessary to stop shape database servers and OMEGA jobs with the UNIX kill command. The server can be started using the runServer.sh script located in the distribution bin directory.

4.4.2 Windows

The RMI server is installed as a service called cartridgeService using Apache procrun. The server my be stopped or started using the Windows management console. Alternatively, an Apache service monitor GUI is supplied in the distribution that can be used to start, stop or edit the service. This GUI is accessed from the windows Task Bar. If it is not present it can be started by running monitorService.bat in the distribution bin directory. For more information about Apache procrun see http://commons.apache.org/daemon/procrun.html. In the cartridge distribution procrun.exe has been renamed to cartridgeService32.exe and cartidgeService64.exe while prunmgr.exe is cartridgeServerW.exe.

4.4.3 Monitoring

The cartridge may be monitored by observing the log file located in oecartridge/log/rmi/rmi_server.log.

4.5 Configuration File

The RMI server is configured using a Java properties file (this file may contain comments, which begin with "#"). The server.properties.template file in the bin subdirectory needs to be copied to server.properties and modified to reflect the desired setup. These strings to must be edited to according to the path of the Cartridge installation: "<install_directory>", "<rmi_hostname>", "<use_cache>", "<cache_size>", "<password>", "<database>" and "<database_host>". The cartridge.install_directory setting is required. It should point to the root of the cartridge distribution. Note that the "/" path separator must be used on both Windows and Linux.

```
cartridge.install_directory = C:/packages/rmi
```

If multiple interfaces are being used or the RMI host name must be specified, use the optional setting java.rmi.server.hostname. This is the hostname that the Oracle client needs to communicate back to the RMI server . Setting this incorrectly will result in an error on the client. If commented out then the Java API call InetAddress.getLocalHost().getHostName() will be used to get the hostname.

```
# java.rmi.server.hostname = localhost
```

Set task_manager.n_threads to the size of the thread pools used to service tasks and batch jobs. Note that because of other concurrency constraints the cartridge may not be able to service this number of tasks simultaneously.

```
task_manager.n_threads = 4
```

Set structure_search.use_molecule cache to true to use the molecule cache for substructure search.

```
structure_search.use_molecule_cache = true
```

OE molecules can be cached as compressed OEDBMols or uncompressed OEMiniMols. Set molecule_cache.use_compression = true to use compressed OEDBMols for structures (the recommended setting). OEMiniMols take up much space than OEDBMols, but they do store reaction information. Even with compression set OEMiniMols will be used for reactions.

```
molecule_cache.use_compression = true
```

The size of the LRU molecule cache is set in molecule_cache.cache_size. Using a size that exceeds physical memory of the server will degrade performace. Note that cache size chosen must account for the additional memory size of the system and other processes.

```
molecule_cache.cache_size = 500000
```

A thread pool dedicated to substructure search may be enabled using the following parameter:

```
structure_search.use_substructure_thread_pool = true
```

The number of threads in the aforementioned substructure search thread pool is set by the following entry. The number of threads in the substructure search thread pool should be at least 2 and no more than the number of processor cores on the RMI machine. Atom by atom matches in substructure search are then distributed among all the threads in the pool. So if an RMI server has 4 processors and a substructure search thread pool of size 4, non-concurrent substructure searches will run about 4 times as fast as when the pool is not enabled (assuming that there are no other intensive processes running). Note that the pool is shared between all current substructure searches, so if two searches are running concurrently on a substructure search thread pool of size 4, each search will access on average two threads and run twice as fast as when the pool is not enabled.

```
substructure_search_thread_pool.n_threads = 4
```

A similar thread pool is available for the creation of structure indexes. It is enabled by the following parameter:

table_index.use_index_build_thread_pool = true

In this case fingerprint creation is distributed among all the threads in the pool. This strategy is effective for SMILES columnns where a thread pool size of 4 will typically reduce the index creation by a factor of 4. For columns containing MDL MOL or OEBinary entries, a performance boost is not normally observed since the RMI server is limited by the rate at which the Oracle server can send data. Note that this thread pool is shared between all current structure index builds.

index_build_thread_pool.n_threads = 4

The next three keys specify the database and password for the C$ARNACHM1 user. If the host setting is commented out the server will attempt to load the OCI driver- otherwise the thin driver will be used. The thin driver is marginally slower than the OCI driver. The database setting for the thin driver is the Oracle SID of the database, while the database setting for the OCI driver is the database name defined in the Oracle TNS listener.

```
credentials.password = secret
credentials.database = DB_NAME
credentials.host = db_name.company.com
credentials.port = 1521
```

The RMI server can pre-load indexes on startup (otherwise indexes will not be loaded until a search is requested or an alter index command is issued). The key table_index.load_tables is used to specify indexes to be loaded into memory from their saved Oracle state. Any index which doesn't currently exist will be ignored.

```
table_index.load_tables = arnachm1_test.nci_open.smiles
```

For a pre-loaded index all molecules can be added to the molecule_cache using the key table_index.cache.<index_name>. The cache size should be sufficient to accommodate all the compounds in the index.

```
table_index.cache.arnachm1_test.nci_open.smiles = true
```

If the RMI server is not on the same host as Oracle then any firewall on the RMI server should be disabled. RMI connections occur on the first available port.

4.6 Test Script

A demonstration script is provided in DIR/test/test.sql. Once compounds have been loaded as described in *Load Test Data and Build Index* or *Loading the Test Table*, the test script can be run:

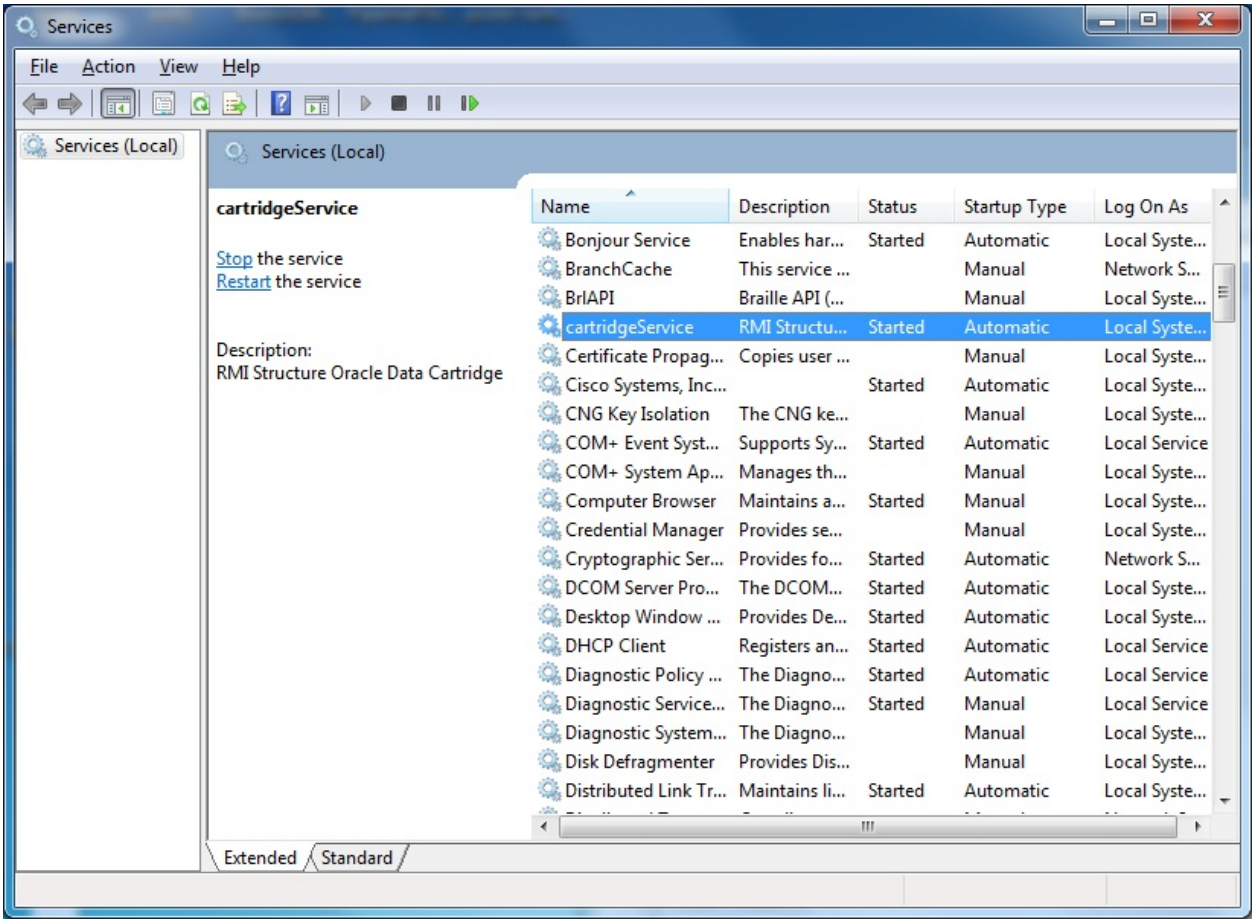
```
$ sqlplus /nolog @test.sql

SQL*Plus: Release 11.2.0.1.0 Production on Mon Apr 9 08:44:13 2012

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

Test user database orcl
Test user password
Connected.
```

```
Call completed.

dropping any existing index

Index dropped.

NCI compound table
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER
 SMILES                                             VARCHAR2(1000)

size of NCI table:
select count(*) "No Compounds" from nci_open;

No Compounds
------------
      260039
```

The test script demonstrates the three domain search operators, index rebuild and SQL filtering and can also be used to fully validate a cartridge install.

4.7 Uninstalling the Cartridge

Existing installations of the Cartridge do not need to be removed prior to installing a new version of the cartridge. Simply stop the RMI server and install any new version on top of the existing cartridge. After a new install any structure indexes will need to be dropped and recreated. To uninstall the cartridge, first stop the RMI server. On Windows, left-click on the Start menu and type "services" in the search box and hit return. This will open the Microsoft Management Console. In the window that opens, right click the "cartridgeService" entry and select "stop" to stop the service.



Next, (Windows only) remove the RMI service:

```
C:\> cd DIR\bin
C:\DIR\bin> cartridgeService32.exe //DS//cartridgeService
```

Use cartridgeService64.exe on 64-bit windows.

Next, connect to Oracle as sys and drop the cartridge user and tester.

```
SQL> drop user C$ARNACHM1 cascade;
User dropped.
SQL> drop user ARNACHM1_TEST cascade;
User dropped.
```

Any structure indexes created by other users will still be present, though obviously it will no longer be possible to perform structural searches on them. While harmless, they can easily be removed using the drop index command.

Finally, the cartridge distribution can be deleted.

4.8 Licenses

The OpenEye Cartridge requires a license that may be obtained from OpenEye Scientific Software (business@eyesopen.com).

The cartridge does not use the OE_LICENSE environment variable. The license should be copied to a file named oe_license.txt in the top level directory of the cartridge distribution.

CHAPTER

FIVE

ADMINISTRATION GUIDE

5.1 Usage

5.1.1 Creating and Manipulating Domain Indexes

To create an index for a varchar2 column containing molecules represented as SMILES strings use:

```
create index molecules_index on nci_open(smiles) indextype is c$arnachm1.structureIndexType;
```

To create an index for a CLOB column containing molecules represented as MDL MOL or SDF blocks use:

```
create index molecules_index2 on nci_open(sdf) indextype is c$arnachm1.structureIndexType;
```

To create an index for a BLOB column containing molecules encoded as OEMol byte arrays use:

```
create index molecules_index3 on nci_open(oemol) indextype is c$arnachm1.structureIndexType;
```

Creating an index may require several minutes per 100K structrues. To run all the commands in this section, the index must be created beforehand. The option exists for this index to be created during the automated install process. To commit the change log table into the serialized Java object (emptying the change log table for the index) use:

```
alter index molecules_index rebuild;
```

To completely rebuild the index (deleting the current change log table and serialized Java object then rebuilding that object) use:

```
alter index molecules_index rebuild parameters('full');
```

To load the index into the RMI server memory use:

```
alter index molecules_index parameters('load');
```

To unload/remove the index from the memory of the RMI server use:

```
alter index molecules_index parameters('unload');
```

To add all molecules in the index to the RMI server OEMol structure cache use:

```
alter index molecules_index parameters('add_to_cache');
```

This command will do nothing if the cache is not enabled.

The index will automatically adjust during column renames, table renames and table truncates.

To delete the index use:

```
drop index molecules_index;
```

To illustrate the use of the domain index we will consider user ARNACHM1_TEST, table NCI_OPEN with column SMILES containing structure. The *test data must be loaded* prior to running the commands in this section. The commands should be executed from the command line prompt after logging in to SQL*Plus as the ARNACHM1_TEST user.

5.1.2 Using Domain Index Operators

To perform a substructure search use the substructure(<column_name>, <query>, <max_hits>) operator, where <query> is a SMARTS pattern or MDL query block and <max_hits> is the maximum number of hits. Set <max_hits> to -1 to retrieve all rows that match. In the case where <query> is a SMARTS pattern, <query> must be a varchar2 string. In the case where <query> is an MDL query block, <query> must be a CLOB. The selection of apprpriate data types through a programming API is demonstrated in the *examples* section. An example SMARTS query:

```
select id
  from nci_open
 where c$arnachm1.substructure(smiles, '[#6]c2nc1ccccc1o2', -1) = 1;
```

The mdl_substructure operator may be used to perform a substructure search using an MDL query block represented by a string, although the operator has been deprecated. The varchar2 data type has a limit of 4000 characters, and as such may not be safe to use in some instances. It is preferable to use the substructure operator with the MDL query block represented by a CLOB data type. The query can use '|' in place of newlines when executing queries pl/sql, although this is unnecessary in java or python.

```
  select id from nci_open
   where c$arnachm1.mdl_substructure(SMILES, 'Cc2nc1ccccc1o2
JME 2002.05 Tue Mar 17 15:39:01 PDT 2009

 10 11                             V2000
    5.9792    1.4000    0.0000 C
    0.0000    2.1000    0.0000 C
    0.0000    0.7000    0.0000 C
    1.2124    2.8000    0.0000 C
    1.2124    0.0000    0.0000 C
    3.7564    2.5326    0.0000 N
    3.7564    0.2674    0.0000 O
    4.5792    1.4000    0.0000 C
    2.4249    2.1000    0.0000 C
    2.4249    0.7000    0.0000 C
  1  8  1  0
  2  3  2  0
  2  4  1  0
  3  5  1  0
  4  9  2  0
  5 10  2  0
  6  8  2  0
  6  9  1  0
  7  8  1  0
  7 10  1  0
```

```
  9 10  1  0
M  END
', 10000) = 1;
```

For exact match use the exactMatch(<column_name>, <query>, <max_hits>) where <query> is a smiles structure. For example,

```
select id
  from nci_open
 where c$arnachm1.exactMatch(smiles,
      'O=C1C(=CC(=O)C=C1)C', -1) = 1;
```

The similarity(<column_name>, <query>, <min_similarity>, <max_hits>) operator retrieves all rows that have Tanimoto similarity of at least <min_similarity> to the query structure.

```
select id from nci_open
 where c$arnachm1.similarity(smiles,
         'CCCc1ccc(cc1)S(=O)(=O)Nc2cc(on2)C', 0.80, -1) = 1;
```

An ancillary operator smililarityScore is provided to retrieve similarity scores:

```
select c$arnachm1.similarityScore(1), id
  from nci_open
 where c$arnachm1.similarity(smiles,
       'CCCc1ccc(cc1)S(=O)(=O)Nc2cc(on2)C', 0.60, -1, 1) = 1;
```

The integer argument in the ancillary operator must match the extra integer argument in the similarity function (this is an Oracle requirement). In this case matching ROWIDs are returned directly in a PL/SQL table (from tableIndexSubstructSqlFilter) and are cast to a SQL table using the table operator.

5.1.3 Domain Index Operators Caveats

The domain index always does a full table scan and returns all hits. When using compound query that contains a common substructure, the full table scan will be inefficient. Consider the following:

```
select count(id)
  from nci_open
 where c$arnachm1.substructure(smiles, 'c1ccccc1', -1) = 1
   and id between 170000 and 171000;
```

The domain index search returns almost 200000 hits and these are filtered by the id clause to give about 600 hits. It would be more efficient to get approximately 1000 hits from the id clause first and then perform substructure search on those. Unfortunately, the Oracle cartridge implementation allows only a full table scan or a single row test. Each single row test would require a call to the RMI server and is impractical for 1000 rows.

To obtain better performance, use SQL that returns ROWIDS to the domain index using *tableIndexSubstructSqlFilter*:

```
select count(id)
   from nci_open m
  where rowid in
       (select * from
        table(c$arnachm1.chem_structure.tableIndexSubstructSqlFilter
              ('ARNACHM1_TEST', 'NCI_OPEN', 'SMILES',
               'select rowid from arnachm1_test.nci_open where id between 170000 and 171000',
                      'c1ccccc1', -1)));
```

There is an analogous function *tableIndexSimilaritySqlFilter* for similarity searches.

5.1.4 Functional Operators

The domain index operators substructure, exactMatch and similarity are not available as functional operators that can be applied to any text column. While this is easily implemented, a design decision was taken to not provide this functionality. Because each row tested using a functional operator will necessitate a call-out to the RMI server, a search on an un-indexed table is likely to overwhelm the RMI server.

5.2 CHEM_STRUCTURE Functions and Procedures

5.2.1 Utility Functions

These are general structure handling functions that do not require domain indexes. In general each function call requires an RMI call-out. Single functions such as molecularWeight or translateStructure should be avoided in select statements that process many rows. If there is interest the single value function can be converted to a domain index operator or a batch function.

MolecularWeight(<smiles>) takes a SMILES argument and returns the molecular weight.

```
select c$arnachm1.chem_structure.molecularWeight('c1ccccc1')
  from dual;
```

CanonicalizeSmiles(<smiles>) takes a SMILES string and returns its canonical isomeric form. It will return a SMILES even if normalization fails.

```
select c$arnachm1.chem_structure.canonicalizeSmiles('Cc2nc1ccccc1o2')
  from dual;
```

5.2.2 Index Procedures

These procedures operate on domain indexes and return PL/SQL tables. The *test data* must already have been loaded into Oracle to run the commands shown in this section. TableIndexSubstructSqlFilter(<owner>, <table>, <column>, <sql_filter>, <query>, [<max_hits>], [<query_type>]) returns a PL/SQL table of ROWIDS for the index owner.table.column for structures which match both the SMARTS substructure query and the sql_filter. The sql_fiter query should return a single column of ROWIDS, which are then tested against the substructure query. Set query_type = 'mdl' to use an MDL mol block as a query (use the 'I' character to represent newline in the query). Note that the SQL in sql_filter will be run by the C$ARNACHM1 user so full schema names of tables are required.

```
select id
   from nci_open m
  where rowid in
   (select * from
     table(
      C$ARNACHM1.chem_structure.tableIndexSubstructSqlFilter
       ('ARNACHM1_TEST', 'NCI_OPEN', 'SMILES',
           'select rowid from ARNACHM1_TEST.nci_open where rownum < 30000',
           '[#6]c2nc1ccccc1o2', -1)));
```

TableIndexSimilaritySqlFilter(<owner>, <table>, <column>, <sql_filter>, <query>, <max_hits>) returns a PL/SQL table of ROWIDS and smilarity scores for the index owner.table.column for structures which both pass the sql_filter and have at least min_similarity similarity with the query structure. The sql_fiter query should return a single column of ROWIDS, which are then tested against the similarity query. Note that the SQL in sql_filter will be run by the C$ARNACHM1 user so full schema names of tables are required.

```
select id, score
  from table(
    C$ARNACHM1.chem_structure.tableIndexSimilaritySqlFilter
     ('ARNACHM1_TEST', 'NCI_OPEN', 'SMILES',
      'select rowid from arnachm1_test.nci_open where rownum < 30000',
      'CCCc1ccc(cc1)S(=O)(=O)Nc2cc(on2)C',
      0.7, -1)) s,
    nci_open m
 where m.rowid = s.hit_rowid;
```

5.3 Implementation Notes

5.3.1 Reactions

The cartridge does support reactions. However, reaction information cannot be stored in compressed OEDBMols. If use_compression is set to true in the configuration file and a reaction is stored in the molecule cache then the reaction will be stored as an OEMiniMol.

To use reactions just use the functions and domain operators as described above. For structures, substitute SMIRKS for SMILES. For specifying substructure queries substitute reaction SMARTS for SMARTS and MDL Rxn blocks for MDL Mol blocks.

5.3.2 Row Level Functions

The Oracle cartridge interface allows for domain index operators to do full index searches or row level searches. The cartridge must implement both. In this cartridge, both implementations do a full table search using the RMI server (even for the row level search a few hundred RMI call-outs are going to take longer than a complete search).

It appears that the optimizer always chooses a full index scan (though this could change in the future). Dbms_output messages indicate which implementation is used when a domain operator is used.

5.3.3 Similarity Ancillary Operator

The similarityScore ancillary function also processes hits by matching one row at a time. It therefore suffers from the same issues as the row-level functional operator implementation described above. For this reason tableIndexSimilaritySearch will normally out-perform the similarity operator with similarityScore ancillary operator.

5.3.4 Transactions

Be wary of performing domain searching while editing the underlying table. Since the domain index searches as another user in a separate connection it cannot be aware of uncommitted edits. When a row is edited and then the domain index is searched prior to a commit, that row will be searched using the incorrect structure. If a row is deleted and a search is performed prior to the commit, the row is returned as a hit will generate an Oracle internal error.

5.3.5 Logging

Logging is enabled on the RMI server using Apache log4j. The logger is configured in bin/log4j.properties. The default configuration from the GUI installer creates one appender which is a rolling file log in log/rmi/rmi_server.log. The RMI server does not output by default to the console. This is easily changed by adding a console appender in the log4j configuration file. There is another available appender which logs messages of level INFO or higher to the Oracle table MESSAGE_TABLE (though connection errors obviously can't be logged). This appender is defined within the code and is not user-configurable. It is disabled by default and can be enabled by editing the setting database.use_database_appender in the configuration file.

If the RMI server is being run as a service, standard output and standard error will be in the procrun log directory (logprocrun). If the runServer.sh command is used on Linux to start the RMI server, output will be directed to the file nohup.out.

Currently, Java in the Oracle JVM does not use log4j (though the classes have been loaded- future improvements may include log4j Oracle JVM logging).

Messages from the Oracle JVM can be found in the Oracle trace file. Alternatively, the SQL command:

```
CALL dbms_java.set_output(200000);
```

will direct Java output to sqlplus.

The PL/SQL functions and procedures print out various messages. To see them use:

```
set serveroutput on size 1000000
```

Note that Java and PL/SQL messages are buffered within SQL*Plus and are not available until an SQL command has completed.

5.3.6 RMI Networking

The external RMI server and the Oracle JVM must use the same hostname/IP address in order for RMI to work . This is particularly important when the server has multiple interfaces.

The server name can be set in server.properties. The client name is set on install and can be changed by updating the table c$arnachm1.rmi_hostname (this table has a single row and column). To set the hostname use (as c$arnachm1):

```
update rmi_hostname set rmi_hostname = 'rmi_server.company.com';
```

5.3.7 PL/SQL Data Structures

This section summarizes the main PL/SQL packages and types used by the cartridge:

- CHEM_STRUCTURE this package provides utility functions and communicates with the RMI server though java functions and procedures.
- STRUCTURE_IND_OBJ this type implements the oracle extensible index interface. Includes all domain index searches.
- INDEX_UTIL this package assists STRUCTURE_IND_OBJ. Includes row-level searches and functional searches.

5.4 Graphsim Index Type

5.4.1 Creating and Manipulating Graphsim Domain Indexes

To create a graphsim path fingerprint index on a varchar2 column containing SMILES use:

```
create index graphsim_index on nci_open(smiles)
       indextype is c$arnachm1.graphsimIndexType;
```

As shown for structure indexes, graphsim indexes may also be built for CLOB columns containing MDL MOL blocks and BLOB columns containing OEBinary arrays.

To create an index with lingo, maccs166 and path based fingerprints use:

```
create index graphsim_index on nci_open(smiles)
       indextype is c$arnachm1.graphsimIndexType parameters ('lingo maccs166 path');
```

To create an index on a remote RMI server use:

```
create index graphsim_index on nci_open(smiles)
       indextype is c$arnachm1.graphsimIndexType parameters ('rmi_hostname:name_of_remote_host');
```

Any of the OEFPType fingerprints may be used in the parameters string. The first fingerprint listed is the default fingerprint.

Creating an index may require minutes for 260K structures. The index will have to be created prior to running the following commands.

To commit the change log table (emptying the change log table for the index) use:

```
alter index graphsim_index rebuild;
```

To completely rebuild the index (deleting the current change log table and cached fingerprints then rebuilding those fingerprints) use:

```
alter index graphsim_index rebuild parameters('full');
```

To load the index into the RMI server memory use:

```
alter index graphsim_index parameters('load');
```

To unload/remove the index from the memory of the RMI server use:

```
alter index graphsim_index parameters('unload');
```

The index will automatically adjust during column renames, table renames and table truncates. To delete the index use:

```
drop index graphsim_index;
```

5.4.2 Using Graphsim Domain Operators

To perform Tanimoto search on the default fingerprint in the index use the graphsim(<column_name>, <query>, <min_similarity>, <max_hits>) operator. For example:

```
select id from nci_open
where c$arnachm1.graphsim(smiles,
      'ClC1CCC2(C3CCC4(C(CCC4C3CC(C2(C1)Br)Br)C(C)CCCC(C)C)C)C',
      0.50, -1) = 1;
```

In order to extract similarity scores for the graphsim operators, the ancillary operator graphsim_score is available: graphsim(<column_name>, <fingerprint>, <search_method>, <query>, <min_similarity>, <max_hits>, <arg1>, <arg2>). Fingerprint is one of the fingerprint types in the index, <search_method> can be either Tanimoto or Tversky and arg1 and arg2 are used to set the alpha and beta parameters for Tversky search (where beta can be null).

```
select id, smiles from nci_open
 where c$arnachm1.graphsim(smiles, 'maccs166', 'tversky',
       'ClC1CCC2(C3CCC4(C(CCC4C3CC(C2(C1)Br)Br)C(C)CCCC(C)C)C)C',
       0.7, -1, 0.95, null) = 1;
```

In order to extract similarity scores for the graphsim operators two ancillary operators are available:

```
select id, c$arnachm1.graphsim_score(1) from nci_open
 where c$arnachm1.graphsim(smiles,
       'ClC1CCC2(C3CCC4(C(CCC4C3CC(C2(C1)Br)Br)C(C)CCCC(C)C)C)C',
       0.50, -1, 1) = 1;
```

5.4.3 Graphsim Implementation Notes

The graphsim functions uses a random access file to cache OEGraph molecules with attached fingerprint data. A delete of a molecule results in an empty space "hole" in the random access file. Thus as the underlying table data is edited the ramdom access file will contain more and more unused space. This is solved by performing a full rebuild. The normal rebuild empties the change log table but does not completely reconstruct the random access cache file.

5.5 Shape Similarity Searches

The cartridge is able to perform real-time shape similarity searches by working in concert with a FastROCS server. Once an RMI server is installed on a GPU capable machine with a pre-existing FastROCS installation, shape searches are performed using FastROCS domain operators.

5.5.1 Shape Indexes

To create a FastROCS index for shape searches use:

```
create index fastrocs_index on struct_table(conformers)
       indextype is c$arnachm1.fastrocsIndexType;
```

It is recommended to build FastROCS indexes on columns of pre-built multi-conformer molecules for large data tables. Building an index on a column of SMILES or SDF strings may require a prohibitive amount of compute time for a large data table.

To change the OMEGA parameters maxconfs and maxtime from their default values use:

```
create index fastrocs_index on nci_open(smiles)
       indextype is c$arnachm1.fastrocsIndexType parameters ('max_conf:50 max_time:10.0');
```

To change from the default RMI server for the index build use:

```
create index fastrocs_index on nci_open(smiles)
       indextype is c$arnachm1.fastrocsIndexType parameters ('rmi_hostname:name_of_remote_host');
```

The name_of_remote_host may either be specified as a fully qualified domain name or IP address.

To defragment the external index (remove "holes" that are created when compounds are removed or edited) use:

```
alter index fastrocs_index rebuild;
```

Note that for the FastROCS index the change log table is committed whenever the table is read.

To completely rebuild the index (deleting the current change log table and serialized Java object then rebuilding that object) use:

```
alter index fastrocs_index rebuild parameters('full');
```

To load the index into the RMI server memory use:

```
alter index fastrocs_index parameters('load');
```

To unload/remove the index from the memory of the RMI server use:

```
alter index fastrocs_index parameters('unload');
```

To validate the structure of the external file of persisted multi-conformer OEMols use:

```
alter index fastrocs_index parameters('validate');
```

The index will automatically adjust during column renames, table renames and table truncates. To delete the index use:

```
drop index fastrocs_index;
```

5.5.2 Shape Search

To perform a FastROCS search on an indexed column use:

```
select * from table_name where c$arnachm1.fastrocs(column_name,
       query, minimum_similarity, max_hits) = 1;
```

for example:

```
select * from mol_table where c$arnachm1.fastrocs(conformers,
       'COc1ccc(cc1)c2nnc(o2)SCC(=O)c3cc(ccc3F)F',1.2, 5000) = 1;
```

Queries may be submitted as a varchar2 containing a SMILES query or a clob containing a 3D conformer in MDL MOL/SDF format. If the query is submitted as a SMILES string, an OMEGA toolkit license will be required on the RMI server on which the index was built. The *minimum_similarity* may be used to filter the returned hits to those exceeding the minimum TanimotoCombo similarity threshold. The maximum number of hits can be used to limit the maximum number of rows returned by the query.

Similarity scores for returned rows my be obtained by the ancillary operator fastrocs_score. As for all ancillary operators, fastrocs_score takes a single integer argument that must match the extra argument passed to the FastROCS domain index operator. For example:

```
select id, c$arnachm1.fastrocs_score(1) from mol_table where
       c$arnachm1.fastrocs(conformers,
       'COc1ccc(cc1)c2nnc(o2)SCC(=O)c3cc(ccc3F)F',1.2, 5000, 1) = 1;
```

Matching conformations superimposed to the coordinates of a 3D query may be obtained by the ancillary operator *fastrocs_conformation*. To obtain the superimposed conformations (returned as CLOBs), use:

```
select id, dbms_lob.getlength(c$arnachm1.fastrocs_conformation(1))
       from mol_table where c$arnachm1.fastrocs(conformers,
       'COc1ccc(cc1)c2nnc(o2)SCC(=O)c3cc(ccc3F)F',1.2, 5000, 1) = 1;
```

Note that the preceding SQL example shows the length of the matching conformation instead of the conformation itself. This is due to the 4000 character limit for varchar2 strings that exists in SQLPLUS. Exceeding this limit results in an error. The full conformations are best retrieved using a programming API as illustrated in the *examples*.

In cases where the query is submitted as a SMILES string, the conformation used in the shape search may be obtained by the function *smiles_to_sdf_conformer_clob*. This function may be called prior to a search to obtain the query conformation, and subsequently passed to the search. An example of the correct way to write this:

```
var conformation_clob clob;
exec :conformation_clob :=
c$arnachm1.chem_structure.smiles_to_sdf_conformer_clob(
    'COc1ccc(cc1)c2nnc(o2)SCC(=O)c3cc(ccc3F)F');
select id from mol_table where
       c$arnachm1.fastrocs(conformers, :conformation_clob, 1.2, 5000) = 1;
```

Care should be used in writing the SQL statement to convert the query to 3D prior to the search. In the following example, the 3D conversion will be performed once for every row returned:

```
select id from mol_table where c$arnachm1.fastrocs(conformers,
       c$arnachm1.chem_structure.smiles_to_sdf_conformer_clob(
       'CCCc1ccc(cc1)S(=O)(=O)Nc2cc(on2)C'), 0.60, 5000) = 1;
```

This is a limitation from the Oracle implementation of extensible indexes.

5.5.3 Notes

The RMI server spawns external processes that may continue running if the RMI server crashes or is not shut down properly. While building a FastROCS index on a column, if the RMI server process is terminated, external OMEGA processes used to create 3D conformations may be left running. This will need to be terminated manually. It is not possible to stop OMEGA external processes from within Java.

When a FastROCS index is loaded into memory, a ShapeDatabaseServer.py process is created. The FastROCS proceses will be terminated upon normal shutdown of the RMI server (i.e. with the UNIX kill signal). However, if the RMI server is terminated abruptly (i.e. UNIX term signal) the ShapeDatabaseServer.py processes will continue running and will need to be stopped manually.

RMI communication as implemented in the cartridge is insecure. Any distributed system is thus best run on a secure intranet or virtual private network. The RMI server must leave port 1521 open to the Oracle server.

Besides the obvious scenario where the cartridge is not sufficiently resourced in terms of memory this section documents other cases where the cartridge may break or appear to break.

5.6 Troubleshooting

5.6.1 Cartridge is not Responsive

Check all outputs for errors as described in the *Logging* section. In the default configuration, log4j messages will be written to log/rmi/rmi_server.log. Under Windows running as a service standard error and standard output will be redirected to time stamped files in log/procrun. In a Linux install standard error and output will end up in nohup.out in the bin directory. Note that many cartridge functions are time consuming and currently all operations block and are applied in a first in, first out queue. If there are no obvious error causes it may be worth restarting the cartridge.

5.6.2 Manually Cleaning an Index

If an index becomes corrupt and if the drop command fails, these steps will clean it completely. Replace the owner, table and column as appropriate for the given index. First, drop the index with the force option:

```
drop index molecules_index force;
```

Next connect as C$ARNACHM1 and remove any stored Java object:

```
delete from java_objects
 where name = 'ARNACHM1_TEST.NCI_OPEN.SMILES_fingerprintLookup';
```

Then see if an entry for a change table exists in the index lookup.

```
select change_table_name from index_lookup
 where index_key = 'ARNACHM1_TEST.NCI_OPEN.SMILES';
CHANGE_TABLE_NAME
------------------------------
change_log_62
```

If one does, drop the table and delete the entry.

```
drop table change_log_62;
delete from index_lookup
 where index_key = 'ARNACHM1_TEST.NCI_OPEN.SMILES';
```

Lastly, commit

```
commit;
```

5.6.3 Oracle Internal Errors

If an index operation returns an internal error this may be due to an invalid row id. For example, see the section regarding *Transactions*. A commit in the current session will normally correct the error. If the error is not corrected, rebuild the index and verify that the index is working correctly.

```
 select id
       from nci_open
      where c$arnachm1.exactMatch(smiles, 'Cl', -1) = 1;
  2    3  select id
*
ERROR at line 1:
ORA-00600: internal error code, arguments: [12406], [], [], [], [], [], [], []
```

5.6.4 No RMI log Messages Created

If the process running the RMI server cannot write to the log directory then no log output will be seen. This can happen on Windows where the system user running the cartridge service may not have the same permissions as the user installing the cartridge. To fix this make sure the log directory is writable by the system user or edit the service (using the Windows management console or the service monitor) so that the installing user runs the service, then restart the service.

CHAPTER

SIX

EXAMPLE CODE

6.1 Example Code

6.1.1 Python Examples

- *Molecule loader*
- *Exact match*
- *Similarity search*
- *SMARTS substructure search*
- *MOL file substructure search*
- *Conformer loader*
- *Shape search*

The following example scripts require that cx_Oracle has been installed. The search examples require that test data has been loaded, and domain indexes have been built. An example molecule loader:

```
#!/usr/bin/env python

from openeye.oechem import *
import os
import sys
import cx_Oracle
import getpass

ORACLE_SID = None
if os.environ['ORACLE_SID']:
    ORACLE_SID = os.environ['ORACLE_SID']

def HandleException(exc):
    error, = exc.args
    print >> sys.stderr, "Oracle-Error-Code:", error.code
    print >> sys.stderr, "Oracle-Error-Message:", error.message


def LoadMolecules(ifs,password):
    conn = None
    cursor = None

    try:
```

```
        conn = cx_Oracle.connect("ARNACHM1_TEST", password, ORACLE_SID)

        # First create the test table.
        cursor = conn.cursor()
        cursor.execute("create table test (title varchar2(100), smiles varchar2(4000) not null)")
        cursor.close()
        print "Created test table"

        # Next load the molecules.
        cursor = conn.cursor()
        sql = "insert into test(title, smiles) values (:title, :smiles)"

        count = 0
        for mol in ifs.GetOEGraphMols():
            title = mol.GetTitle()
            ism = OECreateIsoSmiString(mol)
            cursor.execute(sql, title=title, smiles=ism)
            count += 1
            if count % 1000 == 0:
                conn.commit()
                sys.stdout.write("Loaded %s molecules (so far)\r" % count)
                sys.stdout.flush()

        print "Loaded", count, "molecules (complete)"

        cursor.close()
        conn.commit()

        # Finally create the domain index.
        cursor = conn.cursor()
        cursor.execute(
            "create index test_structure_idx \
            on test(smiles) \
            indextype is c$arnachm1.structureIndexType")
        print "Created index test_structure_idx"

    except cx_Oracle.DatabaseError, exc:
        HandleException(exc)

    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()


InterfaceData = """
!BRIEF [-i] <input> [-p password]
!PARAMETER -i 1
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -p 2
  !TYPE string
  !REQUIRED false
```

```
  !BRIEF oracle password
!END
"""

def main(argv=[__name__]):

    if not ORACLE_SID:
        print >> sys.stderr, "ORACLE_SID not defined.  Please set \
                              the ORACLE_SID environment variable"
        sys.exit(1)

    itf = OEInterface(InterfaceData, argv)

    password = None
    if itf.HasString('-p'):
        password = itf.GetString('-p')
    elif not password:
        password = getpass.getpass()

    ifs = oemolistream()
    filename = itf.GetString("-i")
    if not ifs.open(filename):
        OEThrow.Fatal("Unable to open %s for reading" % filename)

    LoadMolecules(ifs,password)


if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

An example of an exact match search:

```
#!/usr/bin/env python

from openeye.oechem import *
import cx_Oracle
import os
import sys
import getpass

ORACLE_SID = None
if os.environ['ORACLE_SID']:
    ORACLE_SID = os.environ['ORACLE_SID']

def HandleException(exc):
    error, = exc.args
    print >> sys.stderr, "Oracle-Error-Code:", error.code
    print >> sys.stderr, "Oracle-Error-Message:", error.message


def ExactMatchSearch(ifs,ofs,password,tableName,columnName):

    hitCount = 0
    queryCount = 0
    outmol = OEGraphMol()
    conn = None
    cursor = None
```

```
    try:
        conn = cx_Oracle.connect("ARNACHM1_TEST", password, ORACLE_SID)
        cursor = conn.cursor()

        sql = "select :column, title from %s \
               where c$arnachm1.exactMatch(:column, :query, -1) = 1"\
               % tableName

        for mol in ifs.GetOEGraphMols():
            queryCount += 1
            query = OEMolToSmiles(mol)

            cursor.execute(sql, query=query,column=columnName)

            while True:
                rows = cursor.fetchmany();
                if rows == []:
                    break
                for data in rows:
                    hitCount += 1
                    if ofs.GetFormat() == OEFormat_SMI:  #write smiles
                        line = "%s\t%s\n" % (data[0],data[1])
                        ofs.write(line,len(line))
                    else:
                        smi = data[0]
                        id = str(data[1])
                        OESmilesToMol(outmol,smi)
                        outmol.SetTitle(id)
                        OEWriteMolecule(ofs, outmol)

    except cx_Oracle.DatabaseError, exc:
        HandleException(exc)

    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()

    print "Found", hitCount, "exact matches for", queryCount, "queries"
    ifs.close()
    ofs.close()


InterfaceData = """
!BRIEF [-i] <input> [[-o] <output>] [-p password]
!PARAMETER -i 1
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -p 2
  !TYPE string
  !REQUIRED false
  !BRIEF oracle password
!END
```

```
!PARAMETER -o 3
  !ALIAS -out
  !TYPE string
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -table_name 4
  !ALIAS -tname
  !TYPE string
  !DEFAULT test
  !REQUIRED false
  !BRIEF name of Oracle table to be searched
!END
!PARAMETER -column_name 5
  !ALIAS -cname
  !TYPE string
  !DEFAULT smiles
  !REQUIRED false
  !BRIEF name of Oracle column to be searched
!END
"""


def main(argv=[__name__]):

    if not ORACLE_SID:
        print >> sys.stderr, "ORACLE_SID not defined.  Please set \
                              the ORACLE_SID environment variable"
        sys.exit(1)

    itf = OEInterface(InterfaceData, argv)

    ifs = oemolistream()
    filename = itf.GetString("-i")
    if not ifs.open(filename):
        OEThrow.Fatal("Unable to open %s for reading" % filename)

    ofs = oemolostream()
    filename = itf.GetString("-o")
    if not ofs.open(filename):
        OEThrow.Fatal("Unable to open %s for writing" % filename)

    password = None
    if itf.HasString('-p'):
        password = itf.GetString('-p')
    elif not password:
        password = getpass.getpass()

    tableName = itf.GetString('-tname')
    columnName = itf.GetString('-cname')

    ExactMatchSearch(ifs,ofs,password,tableName,columnName)


if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

An example of a similarity search:

```
#!/usr/bin/env python

from openeye.oechem import *
import os
import sys
import cx_Oracle
import getpass

ORACLE_SID = None
if os.environ['ORACLE_SID']:
    ORACLE_SID = os.environ['ORACLE_SID']

def HandleException(exc):
    error, = exc.args
    print >> sys.stderr, "Oracle-Error-Code:", error.code
    print >> sys.stderr, "Oracle-Error-Message:", error.message


def SimilaritySearch(ifs, cutoff, password, tableName = 'test'):
    conn = None
    cursor = None

    try:
        conn = cx_Oracle.connect("ARNACHM1_TEST", password, ORACLE_SID)
        cursor = conn.cursor()

        sql = "select smiles, title, c$arnachm1.similarityScore(1) from %s \
                where c$arnachm1.similarity(smiles, :query, :cutoff, -1, 1) = 1" % tableName


        count = 1
        work_dir = os.getcwd()
        outmol = OEGraphMol()

        for mol in ifs.GetOEGraphMols():
            query = OEMolToSmiles(mol)
            cursor.execute(sql, query=query, cutoff=cutoff)
            output_filename = os.path.join(work_dir,
                                           "sim-hits-%03d.sdf" % count)
            oms = oemolostream(output_filename)
            hit_count = 0;

            while True:
                rows = cursor.fetchmany();
                if rows == []:
                    break
                for data in rows:
                    smi = data[0]
                    title = data[1]
                    if title == None:
                        title = "Hit %i" % (hit_count+1)
                    sim = data[2]
                    outmol.Clear()
                    OESmilesToMol(outmol,smi)
                    outmol.SetTitle(title)
                    OESetSDData(outmol, "SIM", "%.2f" % sim)
                    OEWriteMolecule(oms, outmol)
```

```
                    hit_count += 1

            print "Found", hit_count, "molecules with sim >=", cutoff, \
                  "for query", query, "output to", output_filename
            oms.close()
            count += 1

    except cx_Oracle.DatabaseError, exc:
        HandleException(exc)

    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()


InterfaceData = """
!BRIEF [-i] <input> [-c] <cutoff> [-p password]
!PARAMETER -i 1
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -c 2
  !ALIAS -cutoff
  !TYPE double
  !REQUIRED true
  !BRIEF Tanimoto similarity cutoff
  !KEYLESS 2
!END
!PARAMETER -p 3
  !TYPE string
  !REQUIRED false
  !BRIEF oracle password
  !KEYLESS 3
!END
"""


def main(argv=[__name__]):

    if not ORACLE_SID:
        print >> sys.stderr, "ORACLE_SID not defined.  Please set \
                              the ORACLE_SID environment variable"
        sys.exit(1)

    itf = OEInterface(InterfaceData, argv)

    ifs = oemolistream()
    filename = itf.GetString("-i")
    if not ifs.open(filename):
        OEThrow.Fatal("Unable to open %s for reading" % filename)

    password = None
    if itf.HasString('-p'):
```

```
        password = itf.GetString('-p')
    elif not password:
        password = getpass.getpass()

    cutoff = itf.GetDouble('-c')

    SimilaritySearch(ifs,cutoff,password)


if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

A substructure search using a SMARTS query:

```
#!/usr/bin/env python

from openeye.oechem import *
import os
import sys
import cx_Oracle
import getpass

ORACLE_SID = None
if os.environ['ORACLE_SID']:
    ORACLE_SID = os.environ['ORACLE_SID']


def HandleException(exc):
    error, = exc.args
    print >> sys.stderr, "Oracle-Error-Code:", error.code
    print >> sys.stderr, "Oracle-Error-Message:", error.message


def SmartsSubstructureSearch(smarts,ofs,password,tableName='test'):

    hitCount = 0
    outmol = OEGraphMol()
    conn = None
    cursor = None

    try:
        conn = cx_Oracle.connect("ARNACHM1_TEST", password, ORACLE_SID)
        cursor = conn.cursor()

        sql = "select smiles, title from %s \
               where c$arnachm1.substructure(smiles, :smarts, -1) = 1" \
               % tableName
        cursor.execute(sql, smarts=smarts)

        while True:
            rows = cursor.fetchmany();
            if rows == []:
                break
            for data in rows:
                hitCount += 1
                if ofs.GetFormat() == OEFormat_SMI:  #write smiles
                    line = "%s\t%s\n" % (data[0],data[1])
                    ofs.write(line,len(line))
```

```
                else:
                    smi = data[0]
                    id = str(data[1])
                    OESmilesToMol(outmol,smi)
                    outmol.SetTitle(id)
                    OEWriteMolecule(ofs, outmol)

        print "Found", hitCount, "matches"

    except cx_Oracle.DatabaseError, exc:
        HandleException(exc)

    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
        ofs.close()


InterfaceData = """
!BRIEF [-s] <SMARTS> [[-o] <output>] [-p password]
!PARAMETER -s 1
  !TYPE string
  !REQUIRED true
  !BRIEF SMARTS pattern
  !KEYLESS 1
!END
!PARAMETER -o 2
  !ALIAS -out
  !TYPE string
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -p 3
  !TYPE string
  !REQUIRED false
  !BRIEF oracle password
!END
"""


def main(argv=[__name__]):

    if not ORACLE_SID:
        print >> sys.stderr, "ORACLE_SID not defined.  Please set \
                              the ORACLE_SID environment variable"
        sys.exit(1)

    itf = OEInterface(InterfaceData, argv)

    ofs = oemolostream()
    filename = itf.GetString("-o")
    if not ofs.open(filename):
        OEThrow.Fatal("Unable to open %s for writing" % filename)

    password = None
    if itf.HasString('-p'):
```

```
        password = itf.GetString('-p')
    elif not password:
        password = getpass.getpass()

    smarts = itf.GetString("-s")

    SmartsSubstructureSearch(smarts,ofs,password)


if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

A substructure search using a MOL file query:

```
#!/usr/bin/env python

from openeye.oechem import *
import os
import sys
import cx_Oracle
import getpass

ORACLE_SID = None
if os.environ['ORACLE_SID']:
    ORACLE_SID = os.environ['ORACLE_SID']

def HandleException(exc):
    error, = exc.args
    print >> sys.stderr, "Oracle-Error-Code:", error.code
    print >> sys.stderr, "Oracle-Error-Message:", error.message


def MDLSubstructureSearch(mdl,ofs,password,tableName='test'):

    hitCount = 0
    outmol = OEGraphMol()
    conn = None
    cursor = None

    try:
        conn = cx_Oracle.connect("ARNACHM1_TEST", password, ORACLE_SID)
        cursor = conn.cursor()

        sql = "select smiles, title from %s where \
               c$arnachm1.substructure(smiles, :mdl_clob, -1) = 1" \
               % tableName
        clob = cursor.var(cx_Oracle.CLOB)
        clob.setvalue(0, mdl)
        cursor.execute(sql, mdl_clob=clob)
        while True:
            rows = cursor.fetchmany();
            if rows == []:
                break
            for data in rows:
                hitCount += 1
                if ofs.GetFormat() == OEFormat_SMI:  #write smiles
                    line = "%s\t%s\n" % (data[0],data[1])
                    ofs.write(line,len(line))
```

```
                else:
                    smi = data[0]
                    id = str(data[1])
                    outmol.Clear()
                    OESmilesToMol(outmol,smi)
                    outmol.SetTitle(id)
                    OEWriteMolecule(ofs, outmol)

        print "Found", hitCount, "matches"

    except cx_Oracle.DatabaseError, exc:
        HandleException(exc)

    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
        ofs.close()


InterfaceData = """
!BRIEF [-i] <input> [-o] <output> [-p password]
!PARAMETER -i 1
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -o 2
  !ALIAS -out
  !TYPE string
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -p 3
  !TYPE string
  !REQUIRED false
  !BRIEF oracle password
!END
"""


def main(argv=[__name__]):

    if not ORACLE_SID:
        print >> sys.stderr, "ORACLE_SID not defined.  Please set \
                              the ORACLE_SID environment variable"
        sys.exit(1)

    itf = OEInterface(InterfaceData, argv)

    ifs = oemolistream()
    filename = itf.GetString("-i")
    fp = open(filename, 'r')
    if not fp:
        OEThrow.Fatal("Unable to open %s for reading" % filename)
```

```
    mdl = fp.read()
    fp.close()

    ofs = oemolostream()
    filename = itf.GetString("-o")
    if not ofs.open(filename):
        OEThrow.Fatal("Unable to open %s for writing" % filename)

    password = None
    if itf.HasString('-p'):
        password = itf.GetString('-p')
    elif not password:
        password = getpass.getpass()

    MDLSubstructureSearch(mdl,ofs,password)


if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

An example of populating a blob column with conformers stored in an OEBinary array:

```
#!/usr/bin/env python

from openeye.oechem import *
import os,sys
import cx_Oracle
import getpass

ORACLE_SID = None
if os.environ['ORACLE_SID']:
    ORACLE_SID = os.environ['ORACLE_SID']

def HandleException(exc):
    error, = exc.args
    print >> sys.stderr, "Oracle-Error-Code:", error.code
    print >> sys.stderr, "Oracle-Error-Message:", error.message


def GetSingleResult(cursor):
    rows = cursor.fetchone()
    if rows:
        for row in rows:
            return row

    return None


def LoadConformers(ifs,password,tableName,columnName):
    conn = None
    cursor = None

    try:
        conn = cx_Oracle.connect("ARNACHM1_TEST", password, ORACLE_SID)

        cursor = conn.cursor()

        ofs = oemolostream()
```

```
        ofs.SetFormat(OEFormat_OEB)
        count = 0
        mol = OEMol()
        while OEReadMolecule(ifs,mol):

            ofs.openstring()
            OEWriteMolecule(ofs,mol)
            ofs.close()

            title = mol.GetTitle()
            cursor.execute("select title from %s where title = :title"\
                           % tableName,
                           title=title)
            molID = GetSingleResult(cursor)

            blobvar = cursor.var(cx_Oracle.BLOB)
            cursor.execute("""update %s set %s = empty_blob() where \
                              title = :title returning conf into :blobvar"""\
                           % (tableName,columnName),
                           title=title,
                           blobvar=blobvar)

            blobvar.getvalue().write(ofs.GetString())

            count += 1
            if count % 1000 == 0:
                conn.commit()
                sys.stdout.write("Loaded %s molecules (so far)\r" % count)
                sys.stdout.flush()

        print "Loaded", count, "molecules (complete)"

        conn.commit()

    except cx_Oracle.DatabaseError, exc:
        HandleException(exc)

    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()


InterfaceData = """
!BRIEF [-i] <input> [-p password]
!PARAMETER -i 1
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -p 2
  !TYPE string
  !REQUIRED false
  !BRIEF oracle password
!END
```

```
!PARAMETER -table_name 3
  !ALIAS -tname
  !TYPE string
  !DEFAULT test
  !REQUIRED false
  !BRIEF name of Oracle table to be searched
!END
!PARAMETER -column_name 4
  !ALIAS -cname
  !TYPE string
  !DEFAULT conf
  !REQUIRED false
  !BRIEF name of Oracle column to be searched
!END
"""

def main(argv=[__name__]):

    if not ORACLE_SID:
        print >> sys.stderr, "ORACLE_SID not defined.  Please set \
                              the ORACLE_SID environment variable"
        sys.exit(1)

    itf = OEInterface(InterfaceData, argv)

    password = None
    if itf.HasString('-p'):
        password = itf.GetString('-p')
    elif not password:
        password = getpass.getpass()

    ifs = oemolistream()
    filename = itf.GetString("-i")
    if not ifs.open(filename):
        OEThrow.Fatal("Unable to open %s for reading" % filename)
    hand = ifs.GetBinaryIOHandler()
    hand.Clear()
    OEInitHandler(hand,OEBRotCompressOpts(),OEBDefaultOpts())

    tableName = itf.GetString('-tname')
    columnName = itf.GetString('-cname')

    LoadConformers(ifs,password,tableName,columnName)

if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

A shape similarity search example:

```
#!/usr/bin/env python

from openeye.oechem import *
import os
import sys
import cx_Oracle
import getpass

ORACLE_SID = None
```

```
if os.environ['ORACLE_SID']:
    ORACLE_SID = os.environ['ORACLE_SID']

def HandleException(exc):
    error, = exc.args
    print >> sys.stderr, "Oracle-Error-Code:", error.code
    print >> sys.stderr, "Oracle-Error-Message:", error.message


def ShapeSearch(query,ofs,password,tableName,columnName,tanCutoff,maxHits):

    hitCount = 0
    outmol = OEMol()
    conn = None
    cursor = None

    dimension = query.GetDimension()
    molstr = ""
    if dimension == 3:
        qfs = oemolostream()
        qfs.SetFormat(OEFormat_SDF)
        qfs.openstring()
        OEWriteMolecule(qfs,query)
        qfs.close()
        molstr = qfs.GetString()
    else:
        molstr = OEMolToSmiles(query)

    try:
        conn = cx_Oracle.connect("ARNACHM1_TEST", password, ORACLE_SID)
        cursor = conn.cursor()

        sql = "select c$arnachm1.fastrocs_conformation(1) from %s where \
              c$arnachm1.fastrocs(%s, :molstr, %f, %d, 1) = 1" \
              % (tableName,columnName,tanCutoff,maxHits)

        if dimension == 3:
            clob = cursor.var(cx_Oracle.CLOB)
            clob.setvalue(0, molstr)
            cursor.execute(sql, molstr=clob)
        else:
            cursor.execute(sql, molstr=molstr)

        while True:
            rows = cursor.fetchmany();
            if rows == []:
                break
            for data in rows:
                hitCount += 1
                resfs = oemolistream()
                resfs.SetFormat(OEFormat_SDF)
                resfs.openstring(data[0].read())
                if OEReadMolecule(resfs,outmol):
                    OEWriteMolecule(ofs, outmol)

        print "Found", hitCount, "matches"

    except cx_Oracle.DatabaseError, exc:
```

```
        HandleException(exc)

    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
        ofs.close()


InterfaceData = """
!BRIEF [-i] <input> [-o] <output> [-p password] [-max_hits max_hits] [-tname table_name] []
!PARAMETER -i 1
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
!END
!PARAMETER -o 2
  !ALIAS -out
  !TYPE string
  !REQUIRED true
  !BRIEF Output file name
!END
!PARAMETER -p 3
  !TYPE string
  !REQUIRED false
  !BRIEF oracle password
!END
!PARAMETER -table_name 4
  !ALIAS -tname
  !TYPE string
  !DEFAULT test
  !REQUIRED false
  !BRIEF name of Oracle table to be searched
!END
!PARAMETER -column_name 5
  !ALIAS -cname
  !TYPE string
  !DEFAULT conf
  !REQUIRED false
  !BRIEF name of Oracle column to be searched
!END
!PARAMETER -max_hits 6
  !TYPE int
  !REQUIRED false
  !DEFAULT 1000
  !BRIEF maximum number of hits to return
!END
!PARAMETER -tanimoto_cutoff 7
  !ALIAS -tcut
  !TYPE float
  !DEFAULT 1.2
  !REQUIRED false
  !BRIEF maximum number of hits to return
!END
"""
```

```
def main(argv=[__name__]):

    if not ORACLE_SID:
        print >> sys.stderr, "ORACLE_SID not defined.  Please set \
                              the ORACLE_SID environment variable"
        sys.exit(1)

    itf = OEInterface(InterfaceData, argv)

    ofs = oemolostream()
    filename = itf.GetString("-o")
    if not ofs.open(filename):
        OEThrow.Fatal("Unable to open %s for writing" % filename)

    password = None
    if itf.HasString('-p'):
        password = itf.GetString('-p')
    elif not password:
        password = getpass.getpass()

    filename = itf.GetString("-i")
    ifs = oemolistream()
    if not ifs.open(filename):
        OEThrow.Fatal("Unable to open %s for reading" % filename)

    tableName = itf.GetString('-tname')
    columnName = itf.GetString('-cname')
    maxHits = itf.GetInt('-max_hits')
    tanCutoff = itf.GetFloat('-tcut')

    mol = OEGraphMol()
    while OEReadMolecule(ifs,mol):
        ShapeSearch(mol,ofs,password,tableName,columnName,maxHits,tanCutoff)


if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

6.1.2 Java Examples

- *Molecule loader*
- *Exact match*
- *Similarity search*
- *SMARTS substructure search*
- *MOL file substructure search*
- *Conformer loader*
- *Shape search*

The search examples require that test data has been loaded, and domain indexes have been built. An example molecule loader:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.io.Console;

import openeye.oechem.OEGraphMol;
import openeye.oechem.OEMolBase;
import openeye.oechem.OEInterface;
import openeye.oechem.oechem;
import openeye.oechem.oemolistream;

import java.net.URL;

/*****************************************************************************
MoleculeLoader demonstrate creating and loading molecules from a file
into a database table.
This test table is created prior to loading molecules from the input file.
The domain index is created after molecules are loaded.
*****************************************************************************/

public class MoleculeLoader
{
    public static void moleculeLoader(oemolistream ifs,
                                      String password)
    {
      Connection        conn = null;
      Statement         stmt = null;
      PreparedStatement pstmt = null;

      try
      {
        String ORACLE_SID = null;
        String ORACLE_HOST = "localhost";
        String ORACLE_PORT = "1521";
        String ORACLE_USER = "arnachm1_test";
        if (System.getenv("ORACLE_SID") != null)
          ORACLE_SID = System.getenv("ORACLE_SID");
        if (System.getenv("ORACLE_HOST") != null)
          ORACLE_HOST = System.getenv("ORACLE_HOST");
        if (System.getenv("ORACLE_USER") != null)
          ORACLE_USER = System.getenv("ORACLE_USER");

        String connectURL = "jdbc:oracle:thin:@" + ORACLE_HOST + ":" +
          ORACLE_PORT + ":" + ORACLE_SID;
        conn = Util.createConnection(connectURL,ORACLE_USER,password);

        // First, create the table.
        stmt = conn.createStatement();
        String sql = "create table test (title varchar2(100)," +
          "smiles varchar2(4000) not null,conf blob)";
        stmt.execute(sql);
        System.out.println("Created test table");

        // Next, load the molecules.
        sql = "insert into test(title, smiles) values (?, ?)";
        pstmt = conn.prepareStatement(sql);
        OEMolBase mol = new OEGraphMol();
```

```
      int count = 0;

      while (oechem.OEReadMolecule(ifs, mol))
      {
        String title = mol.GetTitle();
        String ism = oechem.OECreateIsoSmiString(mol);
        pstmt.setString(1, title);
        pstmt.setString(2, ism);
        pstmt.executeUpdate();
        count++;
        if (count % 1000 == 0)
        {
          conn.commit();
          System.out.print("Loaded " + count + " molecules (so far)\r");
          System.out.flush();
        }
      }

      conn.commit();

      System.out.println("Loaded " + count + " molecules (complete)");
      System.out.println("Building domain index");

      // Last, create the domain index.
      stmt = conn.createStatement();
      stmt.execute("create index test_structure_idx on test(smiles) " +
                   "indextype is c$arnachm1.structureIndexType");
      System.out.println("Created index test_structure_idx");
    }
    catch (Exception exception)
    {
      exception.printStackTrace();
    }
    finally
    {
      Util.closeConnnection(conn);
      Util.closeStatement(stmt);
      Util.closeStatement(pstmt);
    }
  }

  public static void main(String[] args)
  {
    URL fileURL = MoleculeLoader.class.getResource("MoleculeLoader.txt");
    if (fileURL == null)
    {
      oechem.OEThrow.Fatal(
        "Unable to open interface file MoleculeLoader.txt");
    }

    oemolistream ifs = null;
    try
    {
      OEInterface itf = new OEInterface(fileURL,"MoleculeLoader", args);
      String inputFilename = itf.GetString("-i");
      String password = "";
      if (itf.HasString("-p"))
      {
```

```
          password = itf.GetString("-p");
        }
        else //prompt for password
        {
          Console cons;
          char [] tmp;
          if ((cons = System.console()) != null &&
              (tmp = cons.readPassword("Password: ")) != null)
          {
            password = new String(tmp);
          }
        }

        ifs = new oemolistream();
        if (!ifs.open(inputFilename))
        {
          oechem.OEThrow.Fatal("Unable to open %s for reading" +
                               inputFilename);
        }

        moleculeLoader(ifs,password);
      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        if (ifs != null)
          ifs.close();
      }
    }
}
```

An example of an exact match search:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.Console;

import openeye.oechem.OEGraphMol;
import openeye.oechem.OEMolBase;
import openeye.oechem.OEInterface;
import openeye.oechem.oechem;
import openeye.oechem.oemolistream;
import openeye.oechem.oemolostream;

import java.net.URL;

/*****************************************************************************
ExactMatch demonstrates exact match searching using the cartridge.
Takes a molecule file query_filename containing one or more query molecules
and searches the 'test' table for exact match molecules. The 'hit' molecules
are written to output_filename.

Assumes the test table and domain index have been created.
(see MoleculeLoader.java)
```

```
*****************************************************************************/

public class ExactMatch
{
    public static void exactMatch(oemolistream ifs,
                                  oemolostream ofs,
                                  String password,
                                  String tableName,
                                  String columnName)
    {
      Connection conn = null;
      PreparedStatement pstmt = null;
      ResultSet rs = null;

      try
      {
        String ORACLE_SID = null;
        String ORACLE_HOST = "localhost";
        String ORACLE_PORT = "1521";
        String ORACLE_USER = "arnachm1_test";
        if (System.getenv("ORACLE_SID") != null)
          ORACLE_SID = System.getenv("ORACLE_SID");
        if (System.getenv("ORACLE_HOST") != null)
          ORACLE_HOST = System.getenv("ORACLE_HOST");
        if (System.getenv("ORACLE_USER") != null)
          ORACLE_USER = System.getenv("ORACLE_USER");

        String connectURL = "jdbc:oracle:thin:@" + ORACLE_HOST + ":" +
          ORACLE_PORT + ":" + ORACLE_SID;
        conn = Util.createConnection(connectURL,ORACLE_USER,password);

        String sql = "select " + columnName + ",title from " + tableName +
          " where " + "c$arnachm1.exactMatch(" + columnName + ", ?, -1) = 1";
        pstmt = conn.prepareStatement(sql);

        OEMolBase mol = new OEGraphMol();

        int queryCount = 0;
        int hitCount = 0;

        while (oechem.OEReadMolecule(ifs, mol))
        {
          String query = oechem.OECreateIsoSmiString(mol);
          pstmt.setString(1, query);
          rs = pstmt.executeQuery();

          while (rs.next())
          {
            String smi = rs.getString(1);
            String title = rs.getString(2);
            mol.Clear();
            oechem.OESmilesToMol(mol,smi);
            mol.SetTitle(title);
            oechem.OEWriteMolecule(ofs, mol);
            hitCount++;
          }
          queryCount++;
        }
```

```
      System.out.println("Found " + hitCount + " exact matches for " +
                         queryCount + " queries");
    }
    catch (Exception exception)
    {
      exception.printStackTrace();
    }
    finally
    {
      Util.closeConnnection(conn);
      Util.closeStatement(pstmt);
    }
  }

  public static void main(String args[])
  {
    URL fileURL = ExactMatch.class.getResource("ExactMatch.txt");
    if (fileURL == null)
    {
      oechem.OEThrow.Fatal("Unable to open interface file ExactMatch.txt");
    }

    oemolistream ifs = null;
    oemolostream ofs = null;
    try
    {
      OEInterface itf = new OEInterface(fileURL,"ExactMatch", args);
      String inputFilename = itf.GetString("-i");
      String outputFilename = itf.GetString("-o");
      String tableName = itf.GetString("-tname");
      String columnName = itf.GetString("-cname");
      String password = "";

      if (itf.HasString("-p"))
      {
        password = itf.GetString("-p");
      }
      else //prompt for password
      {
        Console cons;
        char [] tmp;
        if ((cons = System.console()) != null &&
            (tmp = cons.readPassword("Password: ")) != null)
        {
          password = new String(tmp);
        }
      }

      ifs = new oemolistream();
      if (!ifs.open(inputFilename))
      {
        oechem.OEThrow.Fatal("Unable to open %s for reading " +
                             inputFilename);
      }

      ofs = new oemolostream();
      if (!ofs.open(outputFilename))
```

```
        {
          oechem.OEThrow.Fatal("Unable to open %s for writing " +
                               outputFilename);
        }

        exactMatch(ifs,ofs,password,tableName,columnName);
      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        if (ifs != null)
          ifs.close();
        if (ofs != null)
          ofs.close();
      }
    }
}
```

An example of a similarity search:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.Console;

import openeye.oechem.OEGraphMol;
import openeye.oechem.OEInterface;
import openeye.oechem.oechem;
import openeye.oechem.oemolistream;
import openeye.oechem.oemolostream;

import java.net.URL;

/*****************************************************************************
Similarity demonstrates similarity searching using the data cartridge.
This takes a molecule file containing one of more query molecules and
searches the 'test' table for similar molecules with a Tanimoto >= the
provided cutoff. Each set of results is written to a separate molecule file
in the working directory (sim-hits-XXX.sdf) and the similarity of the
molecules to the query is stored in an SD tag.
*****************************************************************************/

public class Similarity
{

    public static void similarity(oemolistream ifs,
                                  double cutoff,
                                  String password,
                                  String tableName,
                                  String columnName)

    {
      Connection conn = null;
      PreparedStatement pstmt = null;
      ResultSet rs = null;
```

```
try
{
  String ORACLE_SID = null;
  String ORACLE_HOST = "localhost";
  String ORACLE_PORT = "1521";
  String ORACLE_USER = "arnachm1_test";
  if (System.getenv("ORACLE_SID") != null)
    ORACLE_SID = System.getenv("ORACLE_SID");
  if (System.getenv("ORACLE_HOST") != null)
    ORACLE_HOST = System.getenv("ORACLE_HOST");
  if (System.getenv("ORACLE_USER") != null)
    ORACLE_USER = System.getenv("ORACLE_USER");

  String connectURL = "jdbc:oracle:thin:@" + ORACLE_HOST + ":" +
    ORACLE_PORT + ":" + ORACLE_SID;
  conn = Util.createConnection(connectURL,ORACLE_USER,password);

  pstmt = conn.prepareStatement("select " + columnName + ", title, " +
                                "c$arnachm1.similarityScore(1) from " + tableName + " where " +
                                "c$arnachm1.similarity(" + columnName + ", ?, ?, -1, 1) = 1");
  String foo = "select " + columnName + ", title, " +
    "c$arnachm1.similarityScore(1) from " + tableName + " where " +
    "c$arnachm1.similarity(" + columnName + ", ?, ?, -1, 1) = 1";

  System.out.println("sql = " + foo);

  int count = 1;
  String dir = System.getProperty("user.dir");
  OEGraphMol mol = new OEGraphMol();
  OEGraphMol outmol = new OEGraphMol();

  while (oechem.OEReadMolecule(ifs, mol))
  {
    String query = oechem.OECreateIsoSmiString(mol);
    pstmt.setString(1, query);
    pstmt.setDouble(2, cutoff);
    rs = pstmt.executeQuery();

    String outputFilename = dir +
      String.format("/sim-hits-%03d.sdf", (int)count);

    oemolostream oms = new oemolostream(outputFilename);
    int hitCount = 0;

    while (rs.next())
    {
      String smi = rs.getString(1);
      String title = rs.getString(2);
      if (title == null || title.equals(""))
        title = "Hit "+String.valueOf(hitCount+1);
      String tan = String.format("%.3f",rs.getDouble(3));
      outmol.Clear();
      oechem.OESmilesToMol(outmol,smi);
      outmol.SetTitle(title);
      oechem.OESetSDData(outmol,"Tanimoto",tan);
      oechem.OEWriteMolecule(oms, outmol);
      hitCount++;
```

```
      }

      System.out.println("Found " + hitCount + " molecules with sim >= " +
                         cutoff + " for query " + query + " output to " +
                         outputFilename);
      oms.close();
      count++;
    }
  }
  catch (Exception exception)
  {
    exception.printStackTrace();
  }
  finally
  {
    Util.closeConnnection(conn);
    Util.closeStatement(pstmt);
  }
}


public static void main(String[] args)
{
  URL fileURL = Similarity.class.getResource("Similarity.txt");
  if (fileURL == null)
  {
    oechem.OEThrow.Fatal("Unable to open interface file Similarity.txt");
  }

  oemolistream ifs = null;
  try
  {
    OEInterface itf = new OEInterface(fileURL,"Similarity", args);
    String inputFilename = itf.GetString("-i");
    double cutoff = itf.GetDouble("-c");
    String tableName = itf.GetString("-tname");
    String columnName = itf.GetString("-cname");
    String password = "";
    if (itf.HasString("-p"))
    {
      password = itf.GetString("-p");
    }
    else //prompt for password
    {
      Console cons;
      char [] tmp;
      if ((cons = System.console()) != null &&
          (tmp = cons.readPassword("Password: ")) != null)
      {
        password = new String(tmp);
      }
    }

    ifs = new oemolistream();
    if (!ifs.open(inputFilename))
    {
      oechem.OEThrow.Fatal("Unable to open %s for reading " +
                           inputFilename);
```

```
        }

        similarity(ifs,cutoff,password,tableName,columnName);
      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        if (ifs != null)
          ifs.close();
      }
    }
}
```

A substructure search using a SMARTS query:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.Console;

import openeye.oechem.OEGraphMol;
import openeye.oechem.OEMolBase;
import openeye.oechem.OEInterface;
import openeye.oechem.oechem;
import openeye.oechem.oemolostream;

import java.net.URL;

/*****************************************************************************
SubSearchSmarts demonstrates substructure searching using the data cartridge
where the query is an SMARTS string.  This read the query from the command
line and runs a substructure search against the test table. The hits are
written to an output file.
*****************************************************************************/

public class SubSearchSmarts
{
    public static void subSearchSmarts(oemolostream ofs,
                                       String pattern,
                                       String password,
                                       String tableName,
                                       String columnName)
    {
      PreparedStatement pstmt = null;
      ResultSet rs = null;

      Connection conn = null;
      try
      {
        String ORACLE_SID = null;
        String ORACLE_HOST = "localhost";
        String ORACLE_PORT = "1521";
        String ORACLE_USER = "arnachm1_test";
        if (System.getenv("ORACLE_SID") != null)
          ORACLE_SID = System.getenv("ORACLE_SID");
```

```
    if (System.getenv("ORACLE_HOST") != null)
      ORACLE_HOST = System.getenv("ORACLE_HOST");
    if (System.getenv("ORACLE_USER") != null)
      ORACLE_USER = System.getenv("ORACLE_USER");

    String connectURL = "jdbc:oracle:thin:@" + ORACLE_HOST + ":" +
      ORACLE_PORT + ":" + ORACLE_SID;
    conn = Util.createConnection(connectURL,ORACLE_USER,password);
    pstmt = conn.prepareStatement("select " + columnName +
                                  ", title from " + tableName +
                                  " where " + "c$arnachm1.substructure(" +
                                  columnName + ", ?, -1) = 1");
    pstmt.setString(1, pattern);
    rs = pstmt.executeQuery();

    OEMolBase mol = new OEGraphMol();
    int hitCount = 0;

    while (rs.next())
    {
      String smi = rs.getString(1);
      String title = rs.getString(2);
      mol.Clear();
      oechem.OESmilesToMol(mol,smi);
      mol.SetTitle(title);
      oechem.OEWriteMolecule(ofs, mol);
      hitCount++;
    }

    System.out.println("Found " + hitCount + " hits.");
  }
  catch (Exception exception)
  {
    exception.printStackTrace();
  }
  finally
  {
    Util.closeConnnection(conn);
    Util.closeStatement(pstmt);
    Util.closeResultSet(rs);
  }
}


public static void main(String[] args)
{
  URL fileURL = SubSearchSmarts.class.getResource("SubSearchSmarts.txt");
  if (fileURL == null)
  {
    oechem.OEThrow.Fatal(
      "Unable to open interface file SubSearchSmarts.txt");
  }

  oemolostream ofs = null;
  try
  {
    OEInterface itf = new OEInterface(fileURL,"SubSearchSmarts", args);
    String pattern = itf.GetString("-s");
```

```
        String outputFilename = itf.GetString("-o");
        String password = "";
        String tableName = itf.GetString("-tname");
        String columnName = itf.GetString("-cname");

        if (itf.HasString("-p"))
        {
          password = itf.GetString("-p");
        }
        else //prompt for password
        {
          Console cons;
          char [] tmp;
          if ((cons = System.console()) != null &&
              (tmp = cons.readPassword("Password: ")) != null)
          {
            password = new String(tmp);
          }
        }

        ofs = new oemolostream();
        if (!ofs.open(outputFilename))
        {
          oechem.OEThrow.Fatal("Unable to open %s for writing " +
                               outputFilename);
        }

        subSearchSmarts(ofs,pattern,password,tableName,columnName);
      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        if (ofs != null)
          ofs.close();
      }
    }
}
```

A substructure search using a MOL file query:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Clob;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.Console;

import openeye.oechem.OEGraphMol;
import openeye.oechem.OEMolBase;
import openeye.oechem.OEInterface;
import openeye.oechem.oechem;
import openeye.oechem.oemolostream;
```

```
import oracle.sql.CLOB;
import java.net.URL;

/*****************************************************************************
SubSearchMdl demonstrates substructure searching using the data cartridge
where the query is an MDL query.  This read the query from a file and runs
a substructure search against the test table. The hits are written to an
output file.
*****************************************************************************/

public class SubSearchMDL
{
    private static String readMDLQuery(String filename) throws IOException
    {
      BufferedReader reader = null;
      try
      {
        reader = new BufferedReader(new FileReader(filename));
        String line = null;
        StringBuilder stringBuilder = new StringBuilder();
        String ls = System.getProperty("line.separator");
        while ((line = reader.readLine()) != null)
        {
          stringBuilder.append(line);
          stringBuilder.append(ls);
        }
        return stringBuilder.toString();
      }
      finally
      {
        if (reader != null)
          reader.close();
      }
    }

    private static void subSearchMDL(String mdl,
                                     oemolostream ofs,
                                     String password,
                                     String tableName,
                                     String columnName)
    {
      Connection conn = null;
      PreparedStatement pstmt = null;
      ResultSet rs = null;

      try
      {
        String ORACLE_SID = null;
        String ORACLE_HOST = "localhost";
        String ORACLE_PORT = "1521";
        String ORACLE_USER = "arnachm1_test";
        if (System.getenv("ORACLE_SID") != null)
          ORACLE_SID = System.getenv("ORACLE_SID");
        if (System.getenv("ORACLE_HOST") != null)
          ORACLE_HOST = System.getenv("ORACLE_HOST");
        if (System.getenv("ORACLE_USER") != null)
          ORACLE_USER = System.getenv("ORACLE_USER");
```

```
        String connectURL = "jdbc:oracle:thin:@" + ORACLE_HOST + ":" +
          ORACLE_PORT + ":" + ORACLE_SID;
        conn = Util.createConnection(connectURL,ORACLE_USER,password);

        pstmt = conn.prepareStatement("select " + columnName +
                                      ", title from " + tableName +
                                      " where " +
                                      "c$arnachm1.mdl_clob_substructure(" +
                                      columnName + ", ?, -1) = 1");

// Set parameters on prepared statement. In this case it is slightly
// complicated as we have to create a Clob first, set the string on that,
// then set that as the parameter on the prepared statement. This is often
// database dependent.

        Clob clob = oracle.sql.CLOB.createTemporary(conn, false,
                                                    oracle.sql.CLOB.DURATION_SESSION);
        clob.setString(1, mdl);
        pstmt.setClob(1, clob);

        rs = pstmt.executeQuery();

        OEMolBase mol = new OEGraphMol();
        int hitCount = 0;

        while (rs.next())
        {
          String smi = rs.getString(1);
          String title = rs.getString(2);
          mol.Clear();
          oechem.OESmilesToMol(mol,smi);
          mol.SetTitle(title);
          oechem.OEWriteMolecule(ofs, mol);
          hitCount++;
        }
        ofs.close();

        System.out.println("Found " + hitCount + " hits.");

        // Important to free the Oracle CLOB.
        ((CLOB) clob).freeTemporary();

      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        Util.closeConnnection(conn);
        Util.closeStatement(pstmt);
      }
    }

    public static void main(String[] args)
    {
      URL fileURL = SubSearchMDL.class.getResource("SubSearchMDL.txt");
      if (fileURL == null)
```

```
      {
        oechem.OEThrow.Fatal("Unable to open interface file SubSearchMDL.txt");
      }

      oemolostream ofs = null;
      try
      {
        OEInterface itf = new OEInterface(fileURL,"SubSearchMDL", args);
        String inputFilename = itf.GetString("-i");
        String outputFilename = itf.GetString("-o");
        String tableName = itf.GetString("-tname");
        String columnName = itf.GetString("-cname");
        String password = "";

        String mdl = readMDLQuery(inputFilename);

        if (itf.HasString("-p"))
        {
          password = itf.GetString("-p");
        }
        else //prompt for password
        {
          Console cons;
          char [] tmp;
          if ((cons = System.console()) != null &&
              (tmp = cons.readPassword("Password: ")) != null)
          {
            password = new String(tmp);
          }
        }

        ofs = new oemolostream();
        if (!ofs.open(outputFilename))
        {
          oechem.OEThrow.Fatal("Unable to open " + outputFilename +
                               " for writing" +
                               outputFilename);
        }

        subSearchMDL(mdl,ofs,password,tableName,columnName);
      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        if (ofs != null)
          ofs.close();
      }
    }
}
```

An example of populating a blob column with conformers stored in an OEBinary array:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
```

```
import java.sql.ResultSet;
import java.sql.Blob;
import java.io.Console;
import java.io.InputStream;
import java.io.ByteArrayInputStream;

import openeye.oechem.OEMol;
import openeye.oechem.OEMolBase;
import openeye.oechem.OEInterface;
import openeye.oechem.OEFormat;
import openeye.oechem.oechem;
import openeye.oechem.oemolistream;
import openeye.oechem.oemolostream;
import openeye.oechem.OEBinaryIOHandlerBase;

import java.net.URL;

/*****************************************************************************
ConformerLoader demonstrates loading conformers from an OEBinary file
into a database table.  This target table must be created prior to
loading molecules from the input file.  The domain index is created
after molecules are loaded.
*****************************************************************************/

public class ConformerLoader
{
    public static void conformerLoader(oemolistream ifs,
                                       String password,
                                       String tableName,
                                       String columnName)
    {
      Connection        conn = null;
      Statement         stmt = null;
      PreparedStatement pstmt = null;

      try
      {
        String ORACLE_SID = null;
        String ORACLE_HOST = "localhost";
        String ORACLE_PORT = "1521";
        String ORACLE_USER = "arnachm1_test";
        if (System.getenv("ORACLE_SID") != null)
          ORACLE_SID = System.getenv("ORACLE_SID");
        if (System.getenv("ORACLE_HOST") != null)
          ORACLE_HOST = System.getenv("ORACLE_HOST");
        if (System.getenv("ORACLE_USER") != null)
          ORACLE_USER = System.getenv("ORACLE_USER");

        String connectURL = "jdbc:oracle:thin:@" + ORACLE_HOST + ":" +
          ORACLE_PORT + ":" + ORACLE_SID;
        conn = Util.createConnection(connectURL,ORACLE_USER,password);

        int count = 0;
        OEMol mol = new OEMol();
        oemolostream ofs = new oemolostream();
        ofs.SetFormat(OEFormat.OEB);

        String sql = "update " + tableName +
```

```
      " set " + columnName + " = ? where title = ?";

    pstmt = conn.prepareStatement(sql);

    while (oechem.OEReadMolecule(ifs, mol))
    {
      ofs.openstring();
      oechem.OEWriteMolecule(ofs,mol);
      ofs.close();
      byte [] inputBytes = ofs.getByteArray();

      InputStream is = new ByteArrayInputStream(inputBytes);

      String title = mol.GetTitle();
      pstmt.setBinaryStream(1, is, inputBytes.length);
      pstmt.setString(2, title);

      if (pstmt.executeUpdate() == 1)
        count++;
      else
        System.out.println("Molecule '" + mol.GetTitle() + "' not found");

      if (count % 10 == 0)
      {
        conn.commit();
        System.out.print("Loaded " + count + " molecules (so far)\r");
        System.out.flush();
      }
    }

    conn.commit();
    System.out.println("\rLoaded " + count + " molecules (complete)");

    stmt = conn.createStatement();
    sql = "create index test_conf_idx on " +
      tableName + "(" + columnName + ") indextype is c$arnachm1.fastrocsIndexType";
    stmt.execute(sql);
    System.out.println("Created index test_conf_idx");
  }
  catch (Exception exception)
  {
    exception.printStackTrace();
  }
  finally
  {
    Util.closeConnnection(conn);
    Util.closeStatement(stmt);
    Util.closeStatement(pstmt);
  }
}

public static void main(String[] args)
{
  URL fileURL = ConformerLoader.class.getResource("ConformerLoader.txt");
  if (fileURL == null)
  {
    oechem.OEThrow.Fatal(
      "Unable to open interface file MoleculeLoader.txt");
```

```
      }

      oemolistream ifs = null;
      try
      {
        OEInterface itf = new OEInterface(fileURL,"MoleculeLoader", args);
        String inputFilename = itf.GetString("-i");
        String password = "";
        if (itf.HasString("-p"))
        {
          password = itf.GetString("-p");
        }
        else //prompt for password
        {
          Console cons;
          char [] tmp;
          if ((cons = System.console()) != null &&
              (tmp = cons.readPassword("Password: ")) != null)
          {
            password = new String(tmp);
          }
        }

        String tableName = itf.GetString("-tname");
        String columnName = itf.GetString("-cname");

        ifs = new oemolistream();
        if (!ifs.open(inputFilename))
        {
          oechem.OEThrow.Fatal("Unable to open %s for reading" +
                               inputFilename);
        }
        OEBinaryIOHandlerBase hand = ifs.GetBinaryIOHandler();
        hand.Clear();
        oechem.OEInitHandler(hand,
                             oechem.OEBRotCompressOpts(),
                             oechem.OEBDefaultOpts());

        conformerLoader(ifs,password,tableName,columnName);
      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        if (ifs != null)
          ifs.close();
      }
    }
}
```

A shape similarity search example:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Clob;
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.Console;

import openeye.oechem.OEGraphMol;
import openeye.oechem.OEMol;
import openeye.oechem.OEMolBase;
import openeye.oechem.OEInterface;
import openeye.oechem.oechem;
import openeye.oechem.oemolistream;
import openeye.oechem.oemolostream;
import openeye.oechem.OEFormat;

import oracle.sql.CLOB;
import java.net.URL;

/*****************************************************************************
Runs a FastROCS search against the test table using the input query.
This fetches the hits from the database and outputs
the molecules to <output_filename>.
*****************************************************************************/

public class ShapeSearch
{
    private static void shapeSearch(OEMolBase query,
                                    oemolostream ofs,
                                    String password,
                                    String tableName,
                                    String columnName,
                                    float tanCutoff,
                                    int maxHits)
    {
      Connection        conn = null;
      PreparedStatement pstmt = null;
      ResultSet         rs = null;

      try
      {
        String ORACLE_SID = null;
        String ORACLE_HOST = "localhost";
        String ORACLE_PORT = "1521";
        String ORACLE_USER = "arnachm1_test";
        if (System.getenv("ORACLE_SID") != null)
          ORACLE_SID = System.getenv("ORACLE_SID");
        if (System.getenv("ORACLE_HOST") != null)
          ORACLE_HOST = System.getenv("ORACLE_HOST");
        if (System.getenv("ORACLE_USER") != null)
          ORACLE_USER = System.getenv("ORACLE_USER");

        String connectURL = "jdbc:oracle:thin:@" + ORACLE_HOST + ":" +
          ORACLE_PORT + ":" + ORACLE_SID;
        conn = Util.createConnection(connectURL,ORACLE_USER,password);

        pstmt = conn.prepareStatement("select " +
                                      "c$arnachm1.fastrocs_conformation(1) from " + tableName +
                                      " where c$arnachm1.fastrocs(" +
                                      columnName + ", ?, ?, ?, 1) = 1");
```

```
String molstr = "";

int dimension = query.GetDimension();
if (dimension == 3)
{
  oemolostream qfs = new oemolostream();
  qfs.SetFormat(OEFormat.SDF);
  qfs.openstring();
  oechem.OEWriteMolecule(qfs,query);
  qfs.close();
  molstr = qfs.GetString();
}
else
{
  molstr = oechem.OEMolToSmiles(query);
}

Clob clob = oracle.sql.CLOB.createTemporary(conn, false,
                                            oracle.sql.CLOB.DURATION_SESSION);

pstmt.setFloat(2, tanCutoff);
pstmt.setInt(3, maxHits);

if (dimension == 3)
{
  clob.setString(1, molstr);
  pstmt.setClob(1, clob);
}
else
{
  pstmt.setString(1,molstr);
}

rs = pstmt.executeQuery();

OEGraphMol mol = new OEGraphMol();
int hitCount = 0;

oemolistream resfs = new oemolistream();
resfs.SetFormat(OEFormat.SDF);

while (rs.next())
{
  resfs.openstring(rs.getString(1));
  if (oechem.OEReadMolecule(resfs,mol))
  {
    oechem.OEWriteMolecule(ofs,mol);
  }
  else
  {
    oechem.OEThrow.Warning("Unable to read result molecule");
  }

  mol.Clear();
  hitCount++;
}

// Important to free the Oracle CLOB.
```

```
      ((CLOB) clob).freeTemporary();
    }
    catch (Exception exception)
    {
      exception.printStackTrace();
    }
    finally
    {
      Util.closeConnnection(conn);
      Util.closeStatement(pstmt);
    }
  }

  public static void main(String[] args)
  {
    URL fileURL = ShapeSearch.class.getResource("ShapeSearch.txt");
    if (fileURL == null)
    {
      oechem.OEThrow.Fatal("Unable to open interface file ShapeSearch.txt");
    }

    oemolostream ofs = null;
    oemolistream ifs = null;
    try
    {
      OEInterface itf = new OEInterface(fileURL,"ShapeSearch", args);

      String password = "";
      if (itf.HasString("-p"))
      {
        password = itf.GetString("-p");
      }
      else //prompt for password
      {
        Console cons;
        char [] tmp;
        if ((cons = System.console()) != null &&
            (tmp = cons.readPassword("Password: ")) != null)
        {
          password = new String(tmp);
        }
      }

      String outputFilename = itf.GetString("-o");
      ofs = new oemolostream();
      if (!ofs.open(outputFilename))
      {
        oechem.OEThrow.Fatal("Unable to open " + outputFilename +
                             " for writing");
      }

      String inputFilename = itf.GetString("-i");
      ifs = new oemolistream();
      if (!ifs.open(inputFilename))
      {
        oechem.OEThrow.Fatal("Unable to open input file %s " + inputFilename);
      }
```

```
        String tableName = itf.GetString("-tname");
        String columnName = itf.GetString("-cname");
        int maxHits = itf.GetInt("-max_hits");
        float tanCutoff = itf.GetFloat("-tcut");

        OEMol mol = new OEMol();
        while (oechem.OEReadMolecule(ifs,mol))
        {
          shapeSearch(mol,ofs,password,tableName,
                      columnName,tanCutoff,maxHits);
        }
      }
      catch (Exception exception)
      {
        exception.printStackTrace();
      }
      finally
      {
        if (ofs != null)
          ofs.close();
        if (ifs != null)
          ifs.close();
      }
    }
}
```

The directory example_code contains code examples for both Java and Python. The example programs cover substructure search (using both SMARTS patterns and MDL query blocks), similarity search, exact match search and batch molecule loading from a file. See the README.txt files for more information.

CHAPTER

SEVEN

OPEN SOURCE SOFTWARE

The data cartridge includes software (open source) developed by The Apache Software Foundation (http://www.apache.org/). Specifically, unaltered jar-files for log4j, commons lang, commons IO, commons exec, commons pool, commons DBCP, commons CLI, commons collections, Apache web services and Apache XML-RPC are included in this distribution in the lib directory. Additionally, executables from commons procrun (part of the Daemon project) are included in the bin directory. Please see the license at http://www.apache.org/licenses/LICENSE-2.0.txt. You may not use the data cartridge except in compliance with such license. Unless required by applicable law or agreed to in writing, software distributed under such license is distributed on an "AS IS" BASIS WITHOUT WARANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the license for the specific language governing permissions and limitations under such license.

CHAPTER

EIGHT

RELEASE NOTES

8.1 OpenEye Oracle Cartridge v1.0

October 2012

This is the first release of the OpenEye Oracle Cartridge.

INDEX