Choral API Usage

Create Table

Use standard Oracle methods.

The column type of chemical structures must be **CLOB**.

Insert Into Table

Use standard Oracle methods.

Filter out the invalid molecules from the table

The invalid, erroneous molecules must be deleted from the table before the CREATE INDEX statement is executed. The use of isvalidmolecule function is recommended for identifying the invalid molecules.

```
SELECT * FROM <table_name> WHERE isvalidmolecule(<structure_column_name>)=0;
```

Example

```
SELECT + FROM mytable WHERE isvalidmolecule(mol)=0;
```

Create domain index as table_owner

Prerequisite:

Execute privilege on <type>_idxtype

```
CREATE INDEX <index_name> ON <table_name>(<structure_column_name>) INDEXTYPE IS <type>_idxtype;
```

Where

<type> can be one of the type file names present in /data/types/ when the service was initialized

The column to be indexed must be CLOB (Varchar2 is handled as well, but not recommended).

Example

```
CREATE INDEX myidx ON mytable(mol) INDEXTYPE IS choral_owner.sample_idxtype;
```

Use short index names!

Cache loading

The cache is loaded when the first search is executed, but it can be loaded intentionally at any time using the following statements:

```
call load_molecule_cache('index_owner', 'index_name');
call load_fingerprint_cache('index_owner', 'index_name');
```

Both statements are recommended to be executed if the necessary memory is available.

In the case of low memory setup use only the *load_fingerprint_cache* function.

Execute chemical searches as searcher user

Prerequisites:

Execute privilege on <type>_search and <type>_relevance operators.

Select privilege on the table.

Substructure, duplicate, superstructure searches

Search a query structure on a target table

```
SELECT [/*+ <hints> */] * FROM <table_owner>.<table_name>
        WHERE <choral_owner>.<type>_search(<structure_colum_name>, <query_structure>, <search_type>[,
integer])=1 [ORDER BY <type>_relevance(integer)];
```

Search a query structure on a target structure

```
SELECT * FROM DUAL WHERE <choral_owner>.<type>_search(<target_structure>, <query_structure>, <search_type>)
=1;
```

Where

<type> can be one of the type file names present in /data/types/ when the service was initialized

<Search_type> can be:

SUBSTRUCTURE

DUPLICATE

SUPERSTRUCTURE

<hints>: see Adding Hints

Examples

```
SELECT * FROM table_owner.mytable WHERE choral_owner.sample_search(mol, 'CCC', 'SUBSTRUCTURE')=1;

SELECT /*+ DOMAIN_INDEX_SORT() */ * FROM table_owner.mytable WHERE choral_owner.sample_search(mol,
'CCC','SUBSTRUCTURE',1)=1 ORDER BY choral_owner.sample_relevance(1);
```

Search for a limited number of the most relevant hits

Example

```
SELECT * FROM (
        SELECT /*+ DOMAIN_INDEX_SORT() */ * FROM table_owner.mytable WHERE
        choral_owner.sample_search(mol, 'benzene', 'SUBSTRUCTURE',
        1) = 1 ORDER BY choral_owner.sample_relevance(1)
) WHERE rownum <= 10;
```

Full fragment search

Full fragment search can be executed by transforming the query structure in a way that it matches only a full fragment of the target structure and by executing a substructure search on this modified query structure. The transformation adds 's*' query search property to all atoms.

Example

```
SELECT * FROM table_owner.mytable WHERE choral_owner.sample_search(mol, 'CCC','SUBSTRUCTURE','FULLFRAGMENT')
=1;
```

Double bond stereo option: dbsmarkedonly

By default, CIS query matches only with CIS target and TRANS query matches only with TRANS target. If matching of CIS query with both CIS and TRANS targets or matching of TRANS query with both CIS and TRANS targets is aimed, then the query molecule has to be transformed.

The option dbsmarkedonly is provided to accomplish this transformation.

Example

```
SELECT * FROM table_owner.mytable WHERE choral_owner.sample_search(mol, 'C\C=C\C',
'SUBSTRUCTURE','DBSMARKEDONLY')=1;
SELECT * FROM table_owner.mytable WHERE choral_owner.sample_search(mol, 'C\C=C\C',
'SUBSTRUCTURE','DBSMARKEDONLY..FULLFRAGMENT')=1;
```

Similarity search

```
SELECT * FROM <table_owner>.<table_name> WHERE <choral_owner>.<type>_search(<structure_colum_name>,
<query_structure>, 'SIMILARITY'[,integer])<operator><threshold_number> [ORDER BY <type>_relevance(integer)];
```

Where

<type> can be one of the type file names present in /data/types/ when the service was initialized

<Search_type> can be:

<operator> can be: <= or >=

<threshold_number>: number between 0 and 1

Examples

```
SELECT * FROM table_owner.mytable WHERE choral_owner.sample_search(mol, 'CCC','SIMILARITY')>=0.9;

select /*+ DOMAIN_INDEX_SORT() */ * FROM table_owner.mytable
        WHERE choral_owner.sample_search(mol, 'CCC','SIMILARITY',1)>=0.9
        ORDER BY choral_owner.sample_relevance(1);
```

Example for limit

```
SELECT * FROM (SELECT /*+ DOMAIN_INDEX_SORT() */ * FROM table_owner.mytable
        WHERE choral_owner.sample_search(mol, 'CCC','SIMILARITY',1)>=0.9
        ORDER BY choral_owner.sample_relevance(1)) WHERE rownum<=100;
```

Example for limit and offset

```
SELECT * FROM (SELECT /*+ DOMAIN_INDEX_SORT() */ rownum r, t.* FROM table_owner.mytable t
                                        WHERE choral_owner.sample_search(mol, 'CCC','SIMILARITY',1)>=0.9
                                        ORDER BY choral_owner.sample_relevance(1))
        WHERE r BETWEEN 101 AND 200 AND rownum <= 100;
```

Example for selecting similarity value as well

```
SELECT mol,choral_owner.sample_search(mol, 'CCC','SIMILARITY') FROM table_owner.mytable
        WHERE choral_owner.sample_search(mol, 'CCC','SIMILARITY')>=0.9;
```

Reaction search

The following search types are supported not only for molecules but for reactions as well.

- substructure search
- superstructure search
- full fragment search
- duplicate search

Reaction specific query features - like different positions of the reaction arrow - are taken into account. See examples here in Table 2.

Interpreting a molecule string using a specific format

As molecule strings are ambiguous in some cases, it is possible to interpret the given molecule string according to the given molecule format by the following way:

Example

```
'CCC{smiles}'
'CCC{smarts}'
```

In the query structures, the molecule strings which can be interpreted ambiguously like 'CCC' are - by default - handled as SMILES.

Chemical terms

Function chemterm makes possible to calculate chemical terms.

```
chemterm('chemical_term', 'structure')
```

Example

```
SELECT choral_owner.chemterm('mass','CCC') FROM DUAL;
```

Add chemical term column to a table

Example

```
ALTER TABLE mytable ADD (mass number);
UPDATE TABLE mytable SET mass=choral_owner.chemterm('mass', mol)
```

Calculating chemical terms on the standardized structure

Chemical term is calculated on the input structure without standardization. Combine with the standardize method to calculate chemical terms on the standardized structure:

1. Standardize the structure before inserting into the database columns:

   ```
   INSERT INTO <your_table> (mol, ... ) VALUES (sample_standardize(<your structure>), ... )
   ```

2. Store both the not standardized and the standardized structures:

   ```
   INSERT INTO <your_table> (mol,standardized_mol ... ) VALUES (<your structure>, choral_owner.
   sample_standardize(<your structure>, ... )
   ```

3. Standardize on the fly before calling chemical terms:

   ```
   select choral_owner.chemterm(<your_chemical_terms>, choral_owner.sample_standardize
   (<your_structure>));
   ```

Standardize

(Available since version 19.19.)

Calculating the standardized structure

The standardization steps are determined by the molecule type definition. The returned value is a the the standardized structure in MRV format

```
<choral_owner>.<type>_standardize('molecule')
```

where

molecule = a Molecule string

<type> can be one of the type file names present in /data/types/ when the service was initialized

Example:

```
SELECT choral_owner.sample_standardize('C1=CC=CC=C1');
```

Molconvert

Conversion to molecule formats

The use of ChemAxon's MolConverter is supported with some limitations:

```
molconvert('structure','format')
```

where

structure = a Molecule in any of the following formats

format = mrv, mol, rgf, sdf, rdf, csmol, csrgf, cssdf, csrdf, cml, smiles, cxsmiles, abbrevgroup, sybyl, mol2, pdb, xyz, inchi, or name

Example

```
SELECT choral_owner.molconvert('CC', 'mrv') FROM DUAL;
```

Conversion to base64 encoded binary formats (image)

Molecules can be converted to binary image formats (png, jpeg, msbmp, pov, svg, emf, tiff, eps) or other binary formats (pdf) in Base64 encoded form.

Example

```
SELECT choral_owner.molconvert('CC','base64:png') FROM DUAL;
```

isvalidmolecule

The function *isvalidmolecule* defines whether a given chemiical structure string represents a valid molecule or not.

```
isvalidmolecule('structure')
```

where

molecule = a Molecule string or a CLOB type column name storing chemical structures

The output is

1: valid molecule

0: invalid molecule

Example

```
SELECT choral_owner.isvalidmolecule('CC') FROM DUAL;
```

Cost estimation

Cost estimation is automatically set by the installer.

If you want to switch it off, run the sql file found in choral/sql/ folder.:

`choral_disassoc_stats.sql`

Performance Tuning

Below we gathered some steps that can improve the speed of select statements containing chemical search.

Choral service location

The Choral service performing the chemical searches can be installed on the same server as the Oracle server or can be installed on a separate server. If it is installed on a separate server then actions involving heavy network traffic can be affected significantly.

Some examples:

- Chemical index creation
- Substructure searches resulting very large amount of hits, e.g.

  ```
  SELECT COUNT(*) FROM my_big_table WHERE sample_search(mol, 'c1ccccc1', 'SUBSTRUCTURE')=1;
  ```

  Benzene ring is very frequent in organic structures thus in a table with several million structures there can be several million hits.
- Substructure searches requiring functional execution, e.g.

  ```
  SELECT COUNT(*) FROM my_big_table WHERE sample_search(mol, 'c1ccccc1', 'SUBSTRUCTURE')=1 AND id < 100;
  ```

  In this example it is better to retrieve the first hundred IDs and check the substructure condition for each of them separately. But this will result nearly hundred calls to the service, which will be slower if it is on a separate server.

However, if the Oracle Server is heavily used for other than chemical search then it may be better to put the service on a separate machine with a good network connection because high CPU usage of the Oracle Service can slow down the highly parallelized chemical search.

Memory setup

Chemical search can reach its optimal performance if all data required for search are located in the service cache. However, it is possible that there is not enough memory available on the server to store everything in the memory, in this case we would advise to use the low memory setup, which favors storing fingerprints in the cache to storing structural representation of molecules. Fingerprints are needed in each search type, for each chemical search, having the structures available in the memory is usually required only for searches with many hits. There is a memory calculator available, which can be used to get the optimal environment set up in your case.

Notices to memory calculator

Type files to modify are located in folder data/types/ and not in the folder specified by the calculator page.

The calculated Xmx parameter must be copied into files *choral-service.vmoptions* and *run-choral.vmoptions* (not into the file specified on the calculator page).

All other properties calculated by the Xmx calculator must be copied into the configuration file *config/choral.conf* (not into the file specified on the calculator page).

The infix '.runtime' must be omitted from keys presented on the calculator page as the example below shows.

```
com.chemaxon.jchem.psql.label.cachePolicy=LRU
com.chemaxon.jchem.psql.label.cachedObjectCount=1000000
com.chemaxon.jchem.psql.molecule.cachePolicy=LRU
com.chemaxon.jchem.psql.molecule.cachedObjectCount=1000000
com.chemaxon.jchem.psql.fingerprint.cachedObjectCount=1320000
```

Gather statistics

Cost estimation of chemical searches is automatically set up at the installation. However, statistics should be gathered for the table that we are going to search to help Oracle optimizer:

```
call dbms_stats.gather_table_stats(ownname => '<PLAIN USER>', tabname => 'TABLE_NAME', estimate_percent =>
100);
```

Alternatively, statistics can be gathered for the whole schema:

```
call dbms_stats.gather_schema_stats(ownname => '<PLAIN USER>', estimate_percent => 100);
```

Limit the size of the result set

Most of the chemical searches will be really fast if the above two steps are taken, it should take no more than some milliseconds for several million structures. But if the number of hits is very high then it can take longer. In this case it is worth limiting the number of expected results because hits are returned in the order of their relevance and thus the most relevant hits will be returned anyway. In a real time environment the user will not be able to process hundreds of thousands of hits, thus limiting it in order to improve performance is beneficial.

Example of retrieving the hundred most relevant substructure search results:

```
SELECT * FROM (SELECT /*+ DOMAIN_INDEX_SORT() */ * FROM table_owner.mytable
        WHERE choral_owner.sample_search(mol, 'CCC','SUBSTRUCTURE',1)=1
        ORDER BY choral_owner.sample_relevance(1)) WHERE rownum<=100;
```

Adding hints

The speed of the searches can reach its full potential by adding hints to the searches. The hint DOMAIN_INDEX_SORT() is critical to set:

> DOMAIN_INDEX_SORT()
> Important at chemical searches if relevance ordering is used. Doesn’t affect the search speed otherwise.
> Technical background: informs the planner that the search results are already given in the order of relevance so there is no need to fetch all the values and sort it afterwards.

Avoid planning for only chemical search

Cost estimation by the Oracle Query Planner may cost much time compared to the whole search time in some cases of substructure search without other filter conditions, especially in the limited search case. In this case when only substructure search is performed in the select statement it is always better to use the domain index to perform the search then to do a full table scan. Thus, query planning is unnecessary. If a hint is given to the optimizer to use the index cost estimation will still be performed but it can be avoided with a workaround.

Here is a code example to avoid planning by introducing a dummy function and wrapping the query structure into calling this dummy function:

```
CREATE OR REPLACE FUNCTION avoid_planning(v varchar2) RETURN varchar2 IS
BEGIN
  RETURN v;
END;
/

SELECT * FROM test WHERE sample_search(mol,avoid_planning('C'),'SUBSTRUCTURE') = 1;
```

Tuning cost factors for Oracle Query Planner

Choral cartridge provides functions to the Oracle Query Planner to enable estimating the cost of the chemical searches. Currently cost estimation can not be calibrated automatically to the given environment but the factors influencing its behavior can be changed manually.

There are six values in the CHORAL_SETTINGS table of the Choral owner. By updating these values the costs of chemical searches can be influenced.

INDEX_SCREEN_FACTOR
Factor for the cost of fingerprint screening performed using the Choral domain index. Increasing it mildly increases the cost of index usage for substructure and duplicate search.

INDEX_ABAS_FACTOR
Factor for the cost of atom by atom search performed using the Choral domain index. Increasing it significantly increases the cost of index usage for substructure search queries with many hits.

INDEX_SIMILARITY_FACTOR
Factor for the cost of similarity search performed using the Choral domain index. Increasing it increases the cost of index usage for similarity search.

FUNCTION_SCREEN_FACTOR
FUNCTION_ABAS_FACTOR
FUNCTION_SIMILARITY_FACTOR
These are the same factors respectively for functional execution.

These values can be updated to fit the current environment for example by measuring time required for the same query by executing it with index usage or functional execution and setting cost factor values to switch from one to the other at the right moment.

For example, there is a given value for the limit on the ID below where functional and index execution take roughly the same time:

```
SELECT COUNT(*) FROM my_big_table WHERE sample_search(mol, 'c1ccccc1Cl', 'SUBSTRUCTURE')=1 AND id<=<limit
for id>;
```

To find this ID value, one can run the same query with hint for index or functional execution with different ID values and thus get to the value where their time match:

```
SELECT /*+ INDEX(my_big_table my_big_table_chemical_index) */ COUNT(*) FROM my_big_table
        WHERE sample_search(mol, 'c1ccccc1Cl', 'SUBSTRUCTURE')=1 AND id<=<limit for id>;

SELECT /*+ NO_INDEX(my_big_table my_big_table_chemical_index) */ COUNT(*) FROM my_big_table
        WHERE sample_search(mol, 'c1ccccc1Cl', 'SUBSTRUCTURE')=1 AND id<=<limit for id>;
```

Let us say that this value is N. When this value is found then the above described factors can be increased or decreased so that Oracle Query Planner would change the execution plan for the query

```
SELECT COUNT(*) FROM my_big_table WHERE sample_search(mol, 'c1ccccc1Cl', 'SUBSTRUCTURE')=1 AND id<=<limit
for id>;
```

around the value N. The execution plan chosen by the planner can be checked by executing

```
EXPLAIN PLAN FOR SELECT COUNT(*) FROM my_big_table WHERE sample_search(mol, 'c1ccccc1Cl', 'SUBSTRUCTURE')=1
AND id<=<limit for id>;
SELECT * FROM table(dbms_xplan.DISPLAY);
```